UNIVERSIDAD DE BUENOS AIRES
Facultad de Ciencias Exactas y Naturales
Departamento de Matemática

Tesis de Licenciatura

Programación eficiente del torneo argentino de vóley: una aplicación real del *Traveling Tournament Problem*

Joaquín del Priore

Director: Guillermo Durán

18/03/2021

Agradecimientos

A mi familia, en especial a mis viejos, que desde un principio me dieron el apoyo material y no material para poder estudiar esta preciosa carrera. Que supieron entender los tiempos y alegrarse con cada avance y cada tema aprendido, aun sin entender de qué estaba hablando.

A mis amigos, por estar ahí, por bancarme, por hacerme reír y por hacer mi vida más amena.

A Caro, por todo lo que me ayudó a crecer como persona, por ser apoyo constante durante todos mis estudios y por ser la primera en empujarme a cambiarme de carrera.

A Máximo Riso, que despertó en el CBC mi pasión por las matemáticas y por la docencia, pasiones que me acompañan y me acompañarán durante toda mi vida.

A Santiago Gonzalez Zerbo, que con una charla me dio el empujoncito que necesitaba para irme de ingeniería; a Martín Maas que fue el espejo que me dio la bienvenida a exactas.

A Anita Ferrari, que no solo fue una excelente docente sino que también me ayudó en mi búsqueda laboral y de sentido, ofreciéndome consejo y hasta trabajo en mis momentos más complicados.

A Pablo Ferrari que me hizo volver a querer la probabilidad y me ayudó a conseguir la beca para viajar a Rio, a Juan Pablo Pinasco que me enseñó las materias más lindas de la carrera y que al metegol se gana distrayendo al rival. A Marcelo Valdettaro, que llegó a usar una soga y una silla para explicar conceptos de análisis complejo.

A la banda de aplicada, esos compañeros que se convirtieron en amigos y que hicieron que la ida a la facultad fuera un programa divertido. En especial a Naty, por acompañarme desde cálculo avanzado, con la que compartimos todos los TPs de la carrera, siestas en el pasto antes de Real, charlas, peleas, historias de amor y desamor. Te estaré por siempre agradecido.

A Willy, por haber sido el que conectó este grupo de amigos, por haberme iniciado en el maravilloso mundo de la optimización y por su buena onda y calidez ante cualquier duda matemática o extra-matemática. Además por gestionar el increíble viaje a la **ELAVIO** en Lleida en 2019. Agradezco también al tremendo y peligroso grupo que se armó, con Flor, Fede el cirujano, Ivo y Nico.

A Emi, por haberme acompañado en el año más bajo y el año más alto de mi vida, con paciencia y amor, empujándome siempre a hacer la tesis. A Fede Mascialino y Agus López, que me dejaron sacar ideas de sus tesis con una gran fraternidad

académica.

A Juan José Miranda Bront, que me hizo recuperar el amor por la optimización y empezar a estudiar las sutilezas que entran en juego en la implementación de un algoritmo.

A Ine, que me bancó en este año de cuarentena y que me ayudó a cerrar este gran ciclo de mi vida.

Índice general

Índice de figuras

Índice de cuadros

Introducción

En el año 1947, George Dantzig desarrollaba el algoritmo ***simplex***, un método novedoso y sencillo de resolver problemas lineales.

Los problemas lineales son aquellos que pueden ser descriptos con restricciones y objetivos lineales respecto de las variables. Se podría decir que son, en cierto sentido, los problemas más simples que uno puede construir.

Aún así, rápidamente se encontraría el enorme potencial en estos problemas que con modelados sencillos resolvían cuestiones cruciales de la vida cotidiana. Cuestiones como encontrar caminos de distancia mínima o asignar personas a realizar diferentes tareas era ahora mucho más fácil.

Además, con los años (y aún hoy en día, de manera exponencial) llegarían cada vez más avances, tanto en el sentido de algoritmos más sofisticados como en el de máquinas más poderosas que ayudarían a que fuera mucho más rápido resolver este tipo de problemas.

Figura 1: Un joven George Dantzig

Hoy en día la programación lineal y mixta es utilizada en los más diversos campos de conocimiento. Las herramientas disponibles para resolverlos, incluso las gratuitas son cada vez más potentes y las máquinas tienen una capacidad de cómputo que crece día a día.

Esto hace que pueda cualquier persona resolver problemas tan variados como complicados. Siempre y cuando, claro está, sea capaz de hacer un buen modelado del problema entre manos.

Un buen modelado incluye también la capacidad de traducir un problema que en apariencia no es lineal a uno que sí lo es. Logrado esto, se puede aplicar el ya nombrado algoritmo ***simplex*** más algunas otras herramientas modernas y encontrar óptimos o casi óptimos en tiempos muy cortos de ejecución.

Sports scheduling

Uno de los tantos campos en los que se usó la programación lineal y mixta es en el de los deportes. Quien haya visto la película «Moneyball»(2011) sabe que la matemática y el deporte tienen una estrecha relación desde hace años. En particular, el *sports scheduling* se ocupa de decidir quién juega contra quién, cuándo y dónde buscando algún objetivo específico.

Para esto se vale principalmente de métodos de programación entera, *constraint programming* y heurísiticas de varios tipos.

El *sports scheduling* ha sido aplicado en los últimos años, por ejemplo, en la liga de fútbol de Chile [14], las eliminatorias sudamericanas del mundial de fútbol de la FIFA [15], en ligas de básquet [17] y de *hockey* sobre hielo [16] en Estados Unidos.

También ha sido utilizado recientemente en el torneo masculino de básquet en Argentina [5]. El presente trabajo, de hecho, toma como punto de partida la investigación realizada para resolver este último problema.

De los distintos problemas que ataca el *sports scheduling* hay una familia especialmente interesante por su simplicidad a la hora de formularlo y su complejidad a la hora de resolverlo. Se trata de un problema usual en el deporte: armar un campeonato donde cada equipo juegue contra el resto dos veces, una vez de local y otra de visitante (llamado un *double round robin*), minimizando la distancia total viajada por los equipos.

Llamado *traveling tournament problem* (o **TTP**), propuesto por primera vez por Kelly Easton, George Nemhauser y Michael Trick en 2001 [1], reúne dos condiciones muy comúnes en los campeonatos a través de los distintos países y deportes: un torneo con formato *double round robin* y el deseo de viajar lo menos posible.

Estos dos objetivos, el primero uno de factibilidad y el segundo de optimización, son fuerzas tirando en direcciones opuestas y hacen de este un problema sumamente interesante y sobre todo difícil de resolver. Mientras que el problema del viajante (**TSP** por sus siglas en inglés) puede ser resuelto sin problemas para instancias de varios miles de ciudades el **TTP** se vuelve prácticamente imposible con un poco más de 10 equipos.

Cualquier estrategia, cualquier heurística para resolverlo será, entonces, bienvenida.

La liga argentina de Voleibol

Y de todos los deportes, de todas las ligas del mundo, nos va a interesar principalmente la del Voleibol en Argentina. El torneo principal (LVA) es un torneo de dos etapas: la fase regular con formato *double round robin* y una segunda fase de *play off* en formato eliminatorio.

En la primera fase, como es lógico, hay un deseo de minimizar distancias. Esto convierte automáticamente al problema en un **TTP**.

¿Cómo se arma el *fixture* hasta ahora? Aprovechando que las fechas se juegan en días apareados (jueves y sábado o viernes y domingo) y que los equipos se pueden dividir en parejas (aunque no siempre, como veremos), se resuelve un **TTP** para parejas de equipos y parejas de fechas. Esto hace que se gane en complejidad (hay la mitad de variables) pero se pierda, claro está, en optimalidad.

Luego de analizar ventajas y desventajas, propondremos un nuevo sistema, más parecido al de la *NBA*, donde los partidos pueden jugarse cualquier día de la semana y se rompe el concepto de parejas. Esto incurrirá en mejoras en la distancia viajada por todos los equipos durante el torneo, manteniendo además restricciones y pedidos específicos de los equipos.

Contenido

En el Capítulo 1 estudiaremos la programación matemática. Definiremos la clase de problemas y construiremos una solución: el método *simplex*.

Además, presentaremos los problemas enteros y mixtos con otra propuesta de resolución: el *Branch & Bound*.

En el Capítulo 2 presentaremos el *Traveling Tournament Problem*, desde su definición, sus casos especiales y sus variantes. Veremos también algunas estrategias para facilitar la resolución de problemas específicos.

En el Capítulo 3 veremos el problema de la liga de voleibol argentina como una aplicación práctica de lo discutido en los dos capítulos anteriores. Se analizará la estrategia de resolución utilizada hasta ahora para armar el *fixture* y luego se propondrá una nueva estrategia, detallando los modelos matemáticos, las ventajas y desventajas.

En el Capítulo 4 veremos los resultados numéricos de aplicar la nueva estrategia para el armado de *fixtures*. Para eso compararemos lo ya hecho en las temporadas 2017/2018, 2018/2019 y 2019/2020 con nuestros *fixtures* hipotéticos.

Cada una de las temporadas tiene características propias particulares que nos hará tener que cambiar la forma de atacarlo.

En el Capítulo 5, último capítulo, daremos las conclusiones del trabajo y nombraremos los posibles pasos para seguir estudiando el tema.

Un pequeño apartado sobre nomenclaturas

¿Voleibol? ¿Voleybol? ¿Volleyball? ¿Voley? ¿Volley? ¿Cuál es la forma correcta de nombrarlo? ¿Hay alguna correcta?

Para decidirnos quizás sea pertinente remontarnos al origen de la palabra.

En 1895, en Holyoke, Massachusetts, EEUU, William G. Morgan inventaba un juego llamado *mintonette*, por ser un derivado del *badminton*. El *mintonette* se jugaba con cualquier cantidad de jugadores y una red en el medio. El objetivo era pasar una pelota de un lado a otro lado de la red, tras una cantidad de golpes con las manos no limitada.

Tras ver que el juego se jugaba con golpes de volea, se lo empezó a conocer rápidamente como *volley ball* que luego pasaría a ser *volleyball*, su nombre actual.

La palabra inglesa *volley* viene del francés *volée* cuyo origen está en el verbo latín *volāre* y significa "Golpear algo (una pelota usualmente) antes de que toque el suelo". Golpear una pelota en pleno vuelo, podríamos decir. El ADN de nuestro deporte.

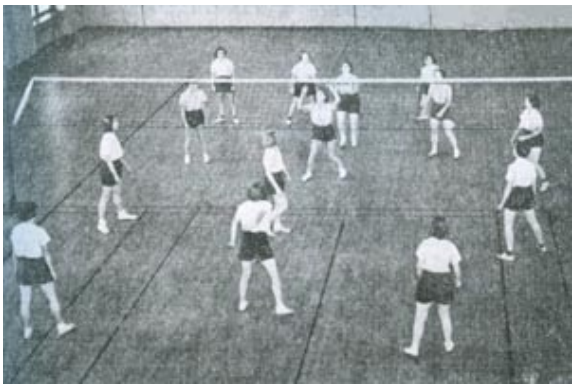


Figura 2: Un juego de *mintonette*, antecesor del moderno Vóley

Pero, ¿cómo nombrarlo en castellano?. En lugar de traducir el nombre y llegar a horribles resultados como "pelota voladora", "pelotavolea" ("balonvolea" existe increíblemente) se escogió en general el camino del extranjerismo adaptado llegando a soluciones como "voleibol", "vóleibol", "volibol" o "vólibol" según la región. Es decir, se intentó reproducir la fonética adaptada a las reglas de escritura de nuestra lengua. Las "y" intermedias y las "ll" quedan entonces descartadas.

Para definirme entre estas opciones recurriré entonces a la **ACLAV**, que en su mismo nombre nos da la solución: **Asociación de Clubes de la Liga Argentina de Voleibol**. Notar que, al no llevar tilde, la palabra "voleibol" está acentuada en la última sílaba, diferenciándola del caso de "vóleibol", también ampliamente utilizada en la Argentina.

Así es que, solo por el bien de la consistencia, utilizaré de ahora en más la forma **voleibol** para referirme al deporte, alternando con la abreviación **vóley** sobre la cual, por suerte, no hay ninguna discusión o duda de cómo escribirla.

Capítulo 1

Programación matemática

La programación matemática u optimización matemática es la técnica de encontrar la mejor opción de un conjunto de alternativas. Es una de las ramas más apasionantes de la matemática y una con gran aplicación al mundo real. Sin muchas veces saberlo, aplicamos técnicas de optimización a diario en nuestra vida cotidiana, ya sea eligiendo un camino para ir a un lugar, acomodando cajas en un baúl, jugando un juego o haciendo transacciones financieras. Incluso uno puede argüir que las acciones más desinteresadas y altruistas llevan intrínsecamente una optimización. Al fin y al cabo, para lograr un mundo mejor, tenemos que definir primero qué significa "mejor" y más importante, qué cosas podemos y no podemos hacer. Y a esto, básicamente, se resume un problema de optimización.

La idea básica es esta: tenemos un conjunto de alternativas "posibles" y queremos encontrar la mejor opción (o las mejores opciones). Nuestras alternativas disponibles va a ser un conjunto A al cual llamaremos "región de factibilidad" y vamos a tener definida una función $f : A \mapsto \mathbb{R}$ comúnmente llamada "función objetivo" la cual vamos a querer minimizar o maximizar.

Es decir (en el caso de minimización por ejemplo), queremos hallar $x_o \in A$ tal que $f(x_0) \leq f(x) \quad \forall x \in A$, sin importar, a priori, si existe más de un x que lo cumpla. En este caso diremos que el óptimo se alcanza en x_0 con un valor de $f(x_0)$.

Dicho esto, es importante notar que un problema de maximización puede ser fácilmente convertido en uno de minimización definiendo $\hat{f} = -f$. El óptimo de esta minimización va a ser óptimo del problema original, ya que $\hat{f}(x_0) \leq \hat{f}(x) \iff -f(x_0) \leq -f(x) \iff f(x_0) \geq f(x)$ para cualquier x.

Por lo que todo lo que se diga de minimización es replicable a maximización y usaré de ahora en más la minimización para hablar del problema general.

Cuando estamos en $\mathbb{R}^n$ vamos usualmente a definir la región de factibilidad mediante una serie de restricciones sobre las variables

$$
\begin{aligned}
&Minimizar\ f \\
Sujeto\ a:\ &R_1 \\
&R_2 \\
&\vdots \\
&R_m
\end{aligned}
$$

Donde las restricciones pueden estar dadas por ecuaciones o inecuaciones.

La forma de f y de las R_i van a definir el tipo de problema que estamos tratando, siendo uno de los más típicos la programación lineal u optimización lineal, donde tanto f como las R_i son lineales respecto de las variables. La optimización lineal es tan importante y ha sido tan ampliamente estudiada que suele nombrarse a toda la otra inmensa familia de problemas como problemas "no lineales" a secas.

Como dice un viejo dicho, esto es como separar al universo en bananas y "no bananas".

Pero la razón de esta clasificación es que los problemas lineales tienen sorprendentes propiedades de las cuales podemos hacer uso, como veremos a continuación.

1.1. Optimización lineal

Los problemas de optimización o programación lineal tienen la característica de que la región de factibilidad está dada por ecuaciones e inecuaciones lineales y la función objetivo también es lineal en las variables del problema.

Se suele escribir el problema generalizado lineal de la siguiente manera:

$$
\begin{aligned}
Minimizar\ & \sum_{i=1}^{n} c_i x_i \\
Sujeto\ a:\ & \sum_{i=1}^{n} a_{1i} x_i \leq b_1 \\
& \quad \vdots \\
& \sum_{i=1}^{n} a_{mi} x_i \leq b_m \\
& \quad x_i \geq 0 \quad \forall i \in \{1, \ldots, n\}
\end{aligned}
$$

Es fácil ver que cualquier problema lineal donde las restricciones no son inecuaciones con menor o igual pueden reducirse a esta la forma estándar. Por la naturaleza de estos problemas, pareciera no tener sentido a priori tener desigualdades estrictas. De cualquier modo, vamos a trabajar de ahora en más con conjuntos cerrados, por lo que pedimos que en las restricciones también valga la igualdad.

Así definido el problema, tenemos la gran ventaja de poder escribirlo en modo matricial, mucho más compacto:

$$\begin{aligned} &Minimizar\ c^T x \\ &Sujeto\ a:\ Ax \leq b \\ &\qquad\qquad x \geq 0 \end{aligned}$$

Con $c, x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ y $b \in \mathbb{R}^m$. Además se entiende que la última línea significa que cada coordenada de x es mayor o igual a 0. Esta forma es llamada también la "estándar".

¿Por qué nos importan estos problemas? ¿Qué tienen de especial? Las regiones formadas por estas restricciones forman un polítopo (generalización de un polígono) convexo y, como veremos en breve, siempre vamos a poder hallar un óptimo en alguno de los vértices del polítopo (si este tiene algún vértice y si existe algún óptimo), transformando así el problema a uno con finitos puntos. Esto hace que el problema sea muchísimo más fácil de resolver y más importante: nos da la posibilidad de descubrir óptimos globales, algo en general imposible en los problemas "no lineales".

Veámoslo brevemente.

Vamos primero a definir formalmente qué es un vértice.

Definición 1. *Dado $C \subseteq \mathbb{R}^n$, $C \neq \emptyset$ cerrado y convexo, diremos que $x \in C$ es un **punto extremo** de C si no existen $x_1, x_2 \in C$ distintos y $\lambda \in (0,1)$ tales que $x = \lambda x_1 + (1-\lambda)x_2$*

Unos párrafos atrás pusimos un condicional sobre si un polítopo tenía vértices o puntos extremos. ¿No tienen todos los polítopos puntos extremos?

Bueno, no siempre. Basta ver que $C = \{(x,y) \in \mathbb{R}^2 / x \leq 0\}$, semiplano de $\mathbb{R}^2$, por lo tanto un polítopo, no tiene puntos extremos.

Sin embargo, resulta ser que los casos en los que no tienen puntos extremos son poco interesantes para nosotros y fácilmente caracterizables: básicamente son los conjuntos que contienen una recta.

Pero para poder demostrar este hecho vamos a tener que dar algunas definiciones y teoremas sobre los **hiperplanos separadores**

Definición 2. *Sea $p \in \mathbb{R}^n$ un vector no nulo y b un número real, llamaremos a las soluciones de la ecuación lineal $p_1x_1 + p_2x_2 + \cdots + p_nx_n = b$ un **hiperplano** de $\mathbb{R}^n$ y lo notaremos $H(p,b)$. Esto es, $H(p,b) = \{x \in \mathbb{R}^n \,/\, p.x = b\}$*

Definición 3. *Decimos que un hiperplano $H(p,b)$ **separa** los conjuntos X e $Y \subseteq \mathbb{R}^n$ si para todo $x \in X$ y para todo $y \in Y$ se cumple que $p.x \leq b \leq p.y$. Además, cuando decimos que H separa al conjunto X del punto x, en realidad nos estamos refiriendo a que separa X del conjunto $\{x\}$.*

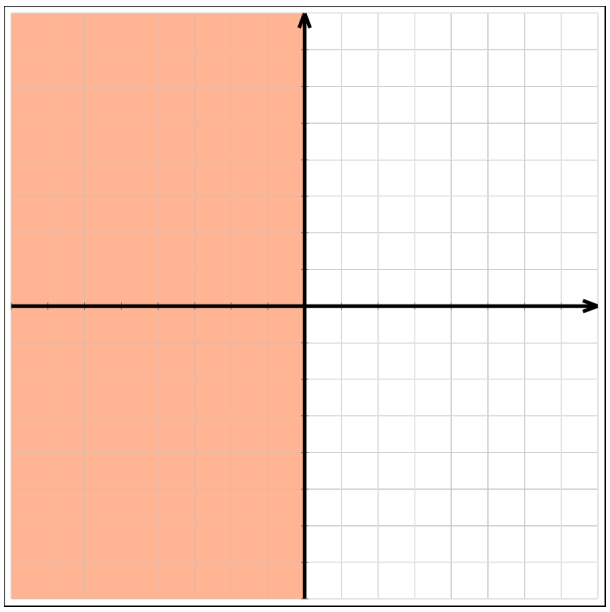


Figura 1.1: Semiplano izquierdo

Definición 4. *Decimos que $H(p, b)$ es un **hiperplano soporte** o que **soporta** al conjunto X si se da que X está completamente de un lado de H y H toca a X en al menos un punto. Esto es, si se cumple que $p.x \leq b$ o $p.x \geq b$ para todo $x \in X$ y además existe un punto $x_0 \in X$ con $p.x_0 = b$, es decir, $x_0 \in H(p, b)$*

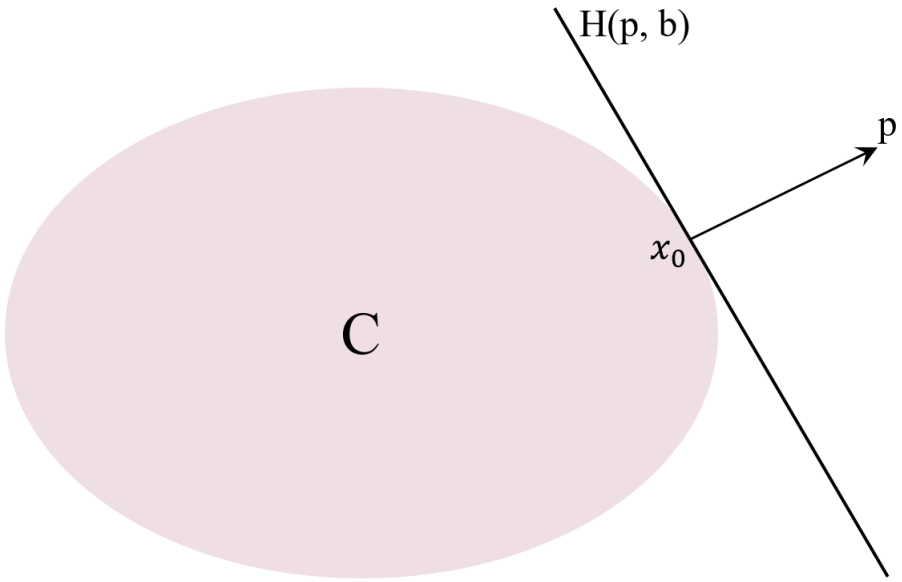


Figura 1.2: El conjunto convexo C siendo soportado por el hiperplano $H(p, b)$, con $b = p.x_0$

Teorema 1. *Sea $C \subseteq \mathbb{R}^n$ un conjunto cerrado convexo y no vacío. Entonces para cada punto $\overline{x}$ que no pertenezca a C existe un hiperplano $H(p, b)$ soporte de C que separa a C de x*

Demostración
Sea $\overline{x} \notin C$ y sea $\hat{x}$ el punto de C más cercano a $\overline{x}$. Es fácil demostrar que este punto existe, solo haciendo uso del hecho que C es cerrado y no vacío.

Sea ahora $p = \hat{x} - \overline{x}$ y sea $b = p.\hat{x}$. Vamos a ver que entonces $H(p, b)$ soporta a S y separa a S de $\overline{x}$.

Primero veamos que $p.\overline{x} < b$. Tenemos $p.\hat{x} - p.\overline{x} = p.(\hat{x} - \overline{x}) = p.p$. Como $\hat{x} \neq \overline{x}$, se tiene que $p \neq 0$, y por lo tanto $p.p > 0$. Por lo tanto $p.\hat{x} - p.\overline{x} > 0$, esto es, $p.\overline{x} < p.\hat{x} = b$, que es lo que buscábamos.

Ahora veamos que todo C se encuentra "del otro lado" de H. Es decir, que para todo $x \in S$ se tiene $p.x \geq b$. Sea $x \in C$ y asumamos por un instante que $p.x < b$. Para cada $\lambda \in (0, 1)$ definamos $x^\lambda = (1 - \lambda)\hat{x} + \lambda x$. Como C es convexo y $x, \hat{x} \in C$, tenemos que $x^\lambda \in C$ para todo λ. Vamos a mostrar que para valores suficientemente chicos de λ tendremos $d(\overline{x}, x^\lambda) < d(\overline{x}, \hat{x})$, una contradicción, ya que supuestamente $\hat{x}$ minimizaba la distancia de $\overline{x}$ a C.

Veamos primero

$$\begin{aligned} x^\lambda - \overline{x} &= (1 - \lambda)\hat{x} + \lambda x - \overline{x} \\ &= (\hat{x} - \overline{x}) + \lambda(x - \hat{x}) \\ &= p + \lambda(x - \hat{x}) \end{aligned}$$

Ahora veamos que $d(\overline{x}, x^\lambda) < d(\overline{x}, \hat{x})$, o lo que es lo mismo, que $||\overline{x} - x^\lambda||^2 < ||\overline{x} - \hat{x}||^2$

$$\begin{aligned} ||\overline{x} - x^\lambda||^2 &= (x^\lambda - \overline{x}).(x^\lambda - \overline{x}) \\ &= \left[p + \lambda(x - \hat{x})\right] . \left[p + \lambda(x - \hat{x})\right] \\ &= p.p + 2\lambda p.(x - \hat{x}) + \lambda^2 ||x - \hat{x}||^2 \\ &= ||\hat{x} - \overline{x}||^2 + \lambda g(\lambda) \end{aligned}$$

Donde $g(\lambda) = 2(p.x - p.\hat{x}) + \lambda||x - \hat{x}||^2$. Y como habíamos que $p.x < b = p.\hat{x}$, tenemos que $\lim_{\lambda \to 0} g(\lambda) < 0$ y por lo tanto para valores chicos de λ vamos a tener que $||\overline{x} - x^\lambda||^2 < ||\hat{x} - \overline{x}||^2$.

Pero aquí logramos la contradicción que queríamos llegar, que $d(\overline{x}, x^\lambda) < d(\overline{x}, \hat{x})$. Como esto no puede pasar, concluimos que $p.x \geq b$ para todo $x \in C$. Además, es obvio que $\hat{x} \in H(p, b)$. □

Teorema 2. *Sea C un conjunto convexo, cerrado, no vacío y sea $\overline{x}$ un punto en la frontera de C. Entonces existe un hiperplano soporte de C que contiene a $\overline{x}$.*

Demostración:
Como $\overline{x}$ está en la frontera de C, para cada $n \in \mathbb{N}$ la bola abierta $B(\overline{x}, \frac{1}{n})$ contiene un punto $x_n \notin C$.Notar que $\lim x_n = \overline{x}$.

Por el teorema anterior podemos afirmar que para cada uno de estos x_n podemos armar un hiperplano H con un $p_n \neq 0 \in \mathbb{R}^n$ tal que

$$\forall x \in C \quad p_n.x \geq p_n.x_n$$

Esto es

$$\forall x \in C \quad p_n.(x - x_n) \geq 0$$

Asumamos sin pérdida de generalidad que $||p_n|| = 1$. Con esto ubicamos a nuestra sucesión p_n sobre la esfera unitaria de $\mathbb{R}^n$, que es compacta. Por lo que la sucesión $\{p_n\}$ tiene una subsucesión convergente que llamaremos $\{p_{n_k}\}$. También llamemos al límite $\overline{p}$, que cumplirá, lógicamente $||\overline{p}|| = 1$. En particular, $\overline{x} \neq 0$.

Tenemos entonces $\{x_{n_k} \to \overline{x}\}$ y $\{p_{n_k} \to \overline{p}\}$ y

$$\forall x \in C \quad p_{n_k}.(x - x_{n_k}) \geq 0$$

Por lo tanto

$$\forall x \in C \quad \overline{p}.(x - \overline{x}) \geq 0$$

y

$$\forall x \in C \quad \overline{p}.x \geq \overline{p}.\overline{x} \geq 0$$

Es decir, podemos tomar el hiperplano $H(\overline{p}, b)$ con $b = \overline{p}.\overline{x}$. Este hiperplano cumple todo lo pedido. □

Ahora sí podemos caracterizar a los conjuntos que no tienen puntos extremos. Para eso vamos a usar el concepto de hiperplano soporte para un punto del conjunto convexo.

Teorema 3. *Sea $C \subseteq \mathbb{R}^n$ un conjunto no vacío, convexo y cerrado. Entonces C tiene un punto extremo $\iff$ C no contiene una recta.*

Demostración:

- ($\implies$) Sea $C \subseteq \mathbb{R}^n$ no vacío convexo y cerrado y sea $\overline{x}$ un punto extremo de C. Vamos a suponer que C contiene una recta, con el propósito de llegar a una contradicción.

  Si así fuera, entonces existen $x \in C$ y $d \in \mathbb{R}^n$ tales que $\{x + \lambda d : \lambda \in \mathbb{R}\} \subseteq C$ Definamos entonces para cada $n \in \mathbb{N}$

  $$x_n = (1 - \frac{1}{n})\overline{x} + \frac{1}{n}(x + nd)$$

  x_n es una combinación convexa entre dos elementos de C: el primero es el punto extremo, el segundo es un punto de la recta. Por lo tanto $x_n \in C$ para cada n (C es convexo). Además, C es cerrado, por lo que tenemos que el límite también está en C.

  $$\lim_{n\to\infty} x_n = \lim_{n\to\infty} \overline{x} + d + \frac{1}{n}(x - \overline{x}) = \overline{x} + d$$

  Cambiando un signo y siguiendo la misma lógica llegamos a que $\overline{x} - d$ también pertenece a C. Pero esto implica que $\overline{x}$ no es un punto extremo de C ya que

  $$\overline{x} = \frac{1}{2}(\overline{x} + d) + \frac{1}{2}(\overline{x} - d)$$

Esto es una contradicción. La cual se da por haber asumido que existía una tal recta. Por lo tanto, C no contiene una recta.

- ($\Longleftarrow$) Supongamos ahora que C no contiene una recta y veamos por inducción en la dimensión n que entonces existe un punto extremo en C

  1) Si $C \subset \mathbb{R}$, el resultado es trivial. Al no contener una recta, está acotado por algún lado y por lo tanto cuenta con un ínfimo o supremo. Al ser cerrado, este pertenece a C y es el extremo.

  n) Ahora supongamos que esto vale para $\mathbb{R}^n$ y mostremos que vale par $\mathbb{R}^{n+1}$

  Sea $C \subset \mathbb{R}^{n+1}$, C no contiene una recta. Por lo que debe de tener algún punto frontera x_0. Por ser C convexo y por el anterior teorema del hiperplano soporte, existe un hiperplano que pasa por x_0 y deja a C de un lado del hemiespacio. Sea $H_{x_0} = \{x \in \mathbb{R}^{n+1} : a^T x = \alpha\}$ para algún $a \in \mathbb{R}^{n+1}$ y algún $\alpha \in \mathbb{R}$ que cumple que $a^T x \leq a^T x_0$ para cada $x \in C$.

  Ahora veamos el conjunto $C \cap H_{x_0}$. Este conjunto es cerrado por ser intersección de cerrados, convexo por la misma razón y no vacío (x_0 pertenece a él). No contiene una recta pues C no lo hacía y es un subconjunto de $\mathbb{R}^n$. Podemos entonces usar la hipótesis inductiva y decir que contiene un punto extremo $\overline{x}$. Es claro que $\overline{x} \in C$ pero, ¿es un punto extremo de C?

  Supongamos que puedo escribir a $\overline{x}$ como combinación convexa de dos puntos x_1 y x_2. Vamos a concluir que esos puntos deben ser ambos iguales a $\overline{x}$:

  $$\overline{x} = \lambda x_1 + (1 - \lambda) x_2$$

  Y entonces tenemos

  $$a^T \overline{x} = \lambda a^T x_1 + (1 - \lambda) a^T x_2$$

  Ahora bien, como $x_1, x_2 \in C$ y $\overline{x} \in H_{x_0}$ que es un hiperplano separador, tenemos que $a^T x_1 \leq a^T \overline{x}$ y $a^T x_2 \leq a^T \overline{x}$. Por lo que para que se cumpla la igualdad, necesitamos que se cumpla también la igualdad $a^T x_1 = a^T \overline{x} = a^T x_2$. Pero esto significa que x_1 y x_2 pertenecen a $C \cap H_{x_0}$, y $\overline{x}$ era un punto extremo allí por lo que nos queda la única opción $x_1 = \overline{x} = x_2$ y por lo tanto aquella no era realmente una combinación convexa y $\overline{x}$ es, en efecto, un punto extremo de C □

Ahora que tenemos este resultado podemos enunciar el que más nos incumbe:

Teorema 4. *Sea $C = \{x \in \mathbb{R}^n \ / \ Ax \leq b\}$ y sea el problema de optimización lineal P: minimizar$\{c^T x : x \in C\}$. Si C tiene un punto extremo y P un óptimo entonces existe un óptimo que también es punto extremo de C.*

Demostración:
Sea α^* el valor en el óptimo y consideremos el conjunto de todos los puntos donde se alcanza el óptimo $O = \{x \in C / c^T x = \alpha^*\}$. C tiene un punto extremo por hipótesis y gracias al teorema anterior sabemos que no contiene una recta. Al ser O un subconjunto de C, tampoco contiene una recta. Además O no es vacío pues P tenía una solución por hipótesis y es claramente cerrado y convexo. Por lo que, de nuevo por teorema anterior, O debe tener un punto extremo $\overline{x}$

Ahora, ¿será $\overline{x}$ también un punto extremo de C?

Sean $x_1, x_2 \in C$ y $\lambda \in (0, 1)$ con

$$\overline{x} = \lambda x_1 + (1 - \lambda) x_2$$

luego

$$c^T \overline{x} = \lambda c^T x_1 + (1 - \lambda) c^T x_2$$

Y como $x_1, x_2 \in C$ y α^* es el valor óptimo en C se sigue que $c^T x_1 \leq c^T \overline{x} = \alpha^*$ y $c^T x_2 \leq c^T \overline{x} = \alpha^*$. Pero entonces para que se cumpla la igualdad se debe tener $c^T x_1 = c^T x_2 = c^T \overline{x} = \alpha^*$. O sea, $\overline{x}$ se escribe como combinación convexa de dos elementos de O. Pero $\overline{x}$ era un punto extremo de O, por lo que $x_1 = x_2 = \overline{x}$.

Por lo que, en efecto, $\overline{x}$ es un punto extremo de C.

□

Esto nos da entonces un resultado crucial, que diferencia este grupo de problemas de cualquier otro: basta recorrer los finitos extremos del polítopo para encontrar el óptimo (siempre y cuando sepamos que el problema tiene óptimo).

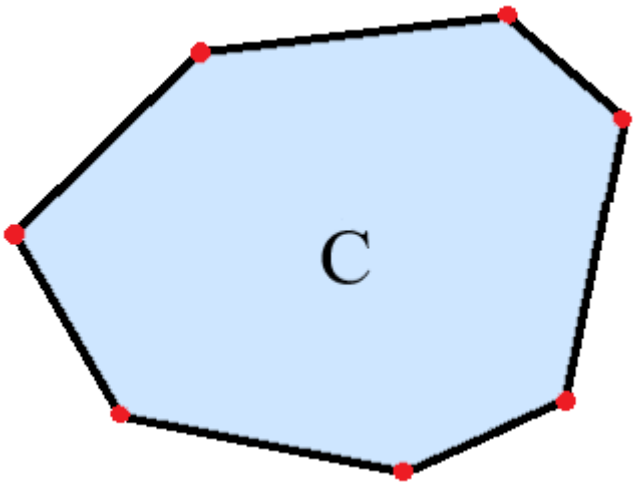


Figura 1.3: Los vértices, en rojo, son todos los posibles óptimos de cualquier función lineal que se aplique sobre $C \subset \mathbb{R}^2$

Además es evidente, al tratarse de finitos puntos, que podemos asegurar fácilmente que un óptimo es global, algo impracticable en la mayoría de los problemas "no lineales".

Ahora bien, sería un poco ingenuo de nuestra parte si creemos que por tener finitos valores para analizar, el problema se convierte en uno fácil de resolver. Cada

vez que uno agrega una variable al problema se suma una dimensión al espacio donde vive el polítopo, y cada restricción nueva puede sumar muchos nuevos vértices a la vez. Cuando estamos lidiando con casos de la vida real, estos problemas se vuelven muy grandes muy rápido, y listar y evaluar cada vértice no parece una tarea razonable (el solo hecho de encontrar la **cantidad** de vértices en un polítopo es un problema muy difícil, estudiado en la combinatoria poliedral y es completo para la clase de complejidad **PP**).

Entonces ¿Cómo encontramos los vértices? ¿en qué orden los recorremos? Aquí entra en juego el método *simplex*.

1.2. El método *simplex*

Para explicar el método *simplex* debemos primero tener una caracterización más apropiada de los vértices o puntos extremos. Una caracterización que nos permita pasar del mundo geométrico al algebraico.

Para empezar, sea P nuestro problema de programación lineal, vamos a pasar a escribirlo de forma canónica:

$$\begin{aligned} &Minimizar\ c^T x \\ &Sujeto\ a:\ Ax = b \\ &\qquad\qquad x \geq 0 \end{aligned}$$

Cualquier problema lineal puede escribirse de esta forma. Cualquier restricción dada por desigualdad puede llevarse a una igualdad agregando una variable nueva: $a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b \longrightarrow a_1x_1 + a_2x_2 + \cdots + a_nx_n + \mathbf{s} = b$ siempre y cuando pidamos que esta nueva variable cumpla $s \geq 0$. También observemos que podemos suponer que $A \in \mathbb{R}^{m\times n}$ con $m \leq n$ y que $rang(A) = m$ sin pérdida de generalidad. En el caso en que no sea así, simplemente tendremos que identificar y eliminar las restricciones redundantes.

Ahora, elijamos un conjunto de índices $B \subset \{1, 2, \ldots, n\}$ de tamaño m, correspondientes a m columnas linealmente independientes de la matriz. Podemos entonces pensar que la matriz A es una concatenación de $A_B \in \mathbb{R}^{m\times m}$ con las columnas correspondientes a los índices B y A_N con el resto de las columnas. Si hacemos lo propio con los índices de x, obtenemos:

$$\begin{aligned} A &= \begin{bmatrix} A_B \mid A_N \end{bmatrix} \\ x &= \begin{bmatrix} x_B \mid x_N \end{bmatrix} \end{aligned}$$

Podemos entonces dar la siguiente

Definición 5. *Si $A = \begin{bmatrix} A_B \mid A_N \end{bmatrix}$, A_B inversible, $x = \begin{bmatrix} x_B \mid x_N \end{bmatrix}$ y se cumple que $A_B.x_B = b$ y $x_N = 0$ decimos que x es una **solución básica**. Si además cumple que $x_B \geq 0$, diremos que x es una **solución básica factible**.*

¿Cómo se conectan estos conceptos algebraicos con los anteriores, geométricos? De la siguiente manera:

Teorema 5. *Sea* $A = \begin{bmatrix} A_1 \mid A_2 \end{bmatrix}$ *y* $x_1 > 0$ *tal que cumple* $A_1.x_1 = b$*. Entonces las columnas de* A_1 *son linealmente dependientes si y solo si* $x = \begin{bmatrix} x_1 \mid 0 \end{bmatrix}$ *no es un punto extremo del conjunto* $C = \{x \in \mathbb{R}^n / A.x = b \ , \ x \geq 0\}$*.*

Demostración:

- ($\Longrightarrow$) Si las columnas de A_1 son linealmente dependientes, existe un vector $c \neq 0$ tal que $A1.c = 0$. Entonces

$$A_1 \overbrace{(x_1 + \epsilon c)}^{v_1} = b$$
$$A_1 \underbrace{(x_1 - \epsilon c)}_{w_1} = b$$

  Ya que todos los elementos en x_1 son positivos podemos tomar un ϵ suficientemente chico para lograr que $v_2 > 0$ y $w_2 > 0$. Pero entonces es claro que $x = \begin{bmatrix} x_1 \mid 0 \end{bmatrix}$ es suma convexa de $v = \begin{bmatrix} v_1 \mid 0 \end{bmatrix}$ y $w = \begin{bmatrix} w_1 \mid 0 \end{bmatrix}$, que son ambos puntos del conjunto C. Por lo que x no es un punto extremo.

- ($\Longleftarrow$) Si x no es punto extremo, existen v, w y $\lambda \in (0, 1)$ tales que $x = \lambda v + (1 - \lambda)w$, con $A.v = b$ y $A.w = b$.

  Entonces

$$0 = \overbrace{\lambda}^{>0} \overbrace{v_2}^{\geq 0} + \overbrace{(1 - \lambda)}^{>0} \overbrace{w_2}^{\geq 0} \quad \Longrightarrow \quad v_2 = w_2 = 0$$

  Por lo que nos queda $x = \lambda v_1 + (1 - \lambda)w_1$. Y como $b = A.v = A_1.v_1 = A_1.x_1$ tenemos finalmente $A_1.(x_2 - v_2) = 0$. Y como tenemos que x_2 no puede ser igual a v_2, la resta no es 0 y por lo tanto las columnas de A_1 deben ser linealmente dependientes.

□

Parafraseando el teorema: $x = \begin{bmatrix} x_1 \mid x_2 \end{bmatrix}$ es un punto extremo si y solo si A_1 tiene columnas linealmente independientes. Es importante notar que no necesariamente el teorema nos da puntos factibles. Nos da $x_1 > 0$ con una cantidad r de coordenadas, potencialmente menor a m (mayor no, pues queremos que A_1 tenga columnas *l.i.*. Pero es fácil ver que podemos agregar columnas hasta formar una base de $\mathbb{R}^n$ (recordar que A tiene rango m) y así conseguir nuestra base B, nuestra matriz A_B y nuestra **solución básica factible** $x = \begin{bmatrix} x_B \mid x_N \end{bmatrix}$.

Ya que $A.x = b$, tenemos que si x es una solución básica factible, existe un conjunto de índices B tal que cumple simultáneamente

- $x_B = A_B^{-1} b$
- $x_N = 0$

Además, si para un dado conjunto de índices ocurren esas dos cosas y además $x_B \geq 0$, entonces $x = \left[x_B \mid x_N\right]$ se trata de una solución básica factible.

Con lo cual, en resumen, basta con ir eligiendo conjuntos B de índices de columnas linealmente independientes de tamaño m y verificar que $x_B = A_B^{-1} b$ tiene todas coordinadas no negativas. Con esto estaremos visitando todos los puntos extremos del polítopo y el mejor valor que encontremos será el óptimo del problema.

Simplex además hace esto de una manera muy inteligente, visitando sólo los vértices que necesita visitar. Ya que cambiar solo un índice de B equivale a moverse a un vértice vecino, *simplex* irá cambiando un índice a la vez, moviéndose de vecino en vecino, y solo si sabe que la función objetivo mejora. Esto sumado al hecho de que un mínimo local va a ser además global hace que el método *simplex* sea muy rápido para encontrar soluciones.

Y podría naturalmente alzarse ahora una pregunta: ¿cuán rápido?

1.2.1. Complejidad algorítmica

Si hablamos de complejidad algorítmica, debemos decir que la del método *simplex* es muy mala. En el peor caso la cantidad de operaciones que deben hacerse es exponencial. Esto es porque en un caso malo deberá recorrer absolutamente todos los vértices hasta encontrar el óptimo. Para mostrar esto se usa el cubo de Klee-Minty, un hipercubo unitario de dimensión variable cuyos vértices han sido levemente perturbados.

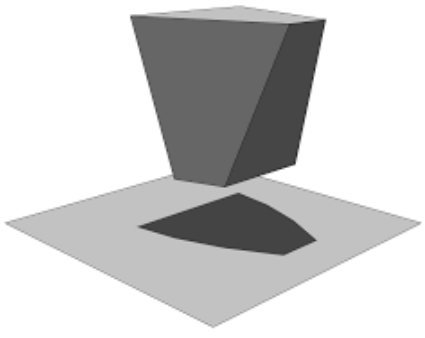


Figura 1.4: Cubo de *Klee-Minty* en 3 dimensiones

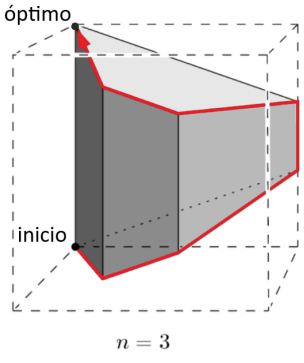


Figura 1.5: Todos los vértices deben ser recorridos para llegar al óptimo

Para este polítopo, el algoritmo *simplex* deberá recorrer todos los vértices antes de llegar al óptimo, convirtiendo el problema en uno exponencial (recordemos que el cubo d-dimensional tiene 2^d vértices). Incluso para cada diversa variación que se ha encontrado a *simplex*, se ha propuesto también un cubo donde se recorran todos los vértices.

Pero entonces, ¿debemos desechar el algoritmo *simplex* por tener mala complejidad? En absoluto. Los tiempos promedio de ejecución del algoritmo son muy buenos. Tanto que *simplex* sigue siendo preferido a los llamados "métodos de punto interior" que, aunque su complejidad es polinomial, suelen ser peores en promedio.

Tal vez solo necesitemos que nuestro *solver* identifique si el problema es un cubo de *Klee-Minty* y lo recorra en sentido contrario...

1.3. Optimización lineal entera y mixta

Ahora bien, para muchos problemas de la vida real nos interesa poder hacer modelos con variables enteras. Hay muchos casos donde la integridad es intrínseca al problema y no podemos aceptar como solución que $3,2$ camiones se asignen a una ruta, o que para decidir si jugar o no un partido la respuesta sea $0,7$. Estas últimas, donde debemos decidir entre *sí* o *no* se suelen modelar con variables binarias (0 o 1) que no son más que variables enteras acotadas.

Entran entonces en escena los llamados problemas lineales enteros, donde se pide que las variables sean enteras o más en general los mixtos, donde algunas variables son reales y otras enteras:

$$\begin{aligned} &Minimizar\ c^T x + d^T y \\ &Sujeto\ a:\ Ax + Dy \leq b \\ &\qquad\qquad x \geq 0 \\ &\qquad\qquad y_i \in \mathbb{Z}_{\geq 0}\ \forall i \end{aligned} \tag{1.1}$$

Pero, ¿sirve *simplex* para resolver esto? Bueno...en principio no. Todas las técnicas y teoremas anteriormente nombrados se basan fuertemente en que estamos en $\mathbb{R}^n$ y no funcionan en este nuevo espacio. Lo que sí podemos hacer es diseñar algunos meta-algoritmos, que se hagan de *simplex* para ir resolviendo el problema.

Para eso, veamos algunas definiciones que nos van a servir para entender mejor nuestro problema.

Lo primero a notar es que, si tomamos el conjunto

$$C = \{x \in \mathbb{R}^n_{\geq 0}\,,\, y \in \mathbb{Z}^p_{\geq 0} \,/\, Ax + Dy \leq b\}$$

y al conjunto relajado

$$C_r = \{x \in \mathbb{R}^n_{\geq 0}\,,\, y \in \mathbb{R}^p_{\geq 0} \,/\, Ax + Dy \leq b\}$$

es obvio que $C \subseteq C_r$. Esto nos servirá, como veremos más adelante, para obtener una cota de nuestro problema. Al estar contenido un conjunto en otro, se ve claramente que al optimizar sobre C_r (resolver el problema "relajado") nos va a devolver un óptimo mejor (o con suerte, igual) al del conjunto C.

Y tal vez surjan unas dudas: ¿puede darnos el mismo resultado? ¿puedo de alguna forma resolver el problema relajado y llegar al óptimo del problema general?
Para eso necesitamos algunas definiciones.

Definición 6. *Sea un conjunto $C \in \mathbb{R}^N$, se define la cápsula convexa de C como el menor conjunto convexo que contiene a C. Equivalentemente se puede definir como la intersección de todos los conjuntos convexos que contienen a C o, más interesantemente, el conjunto de todas las combinaciones convexas de los puntos de C.*

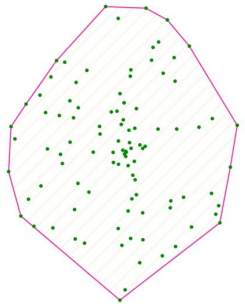


Figura 1.6: La cápsula convexa de un conjunto discreto en $\mathbb{R}^2$

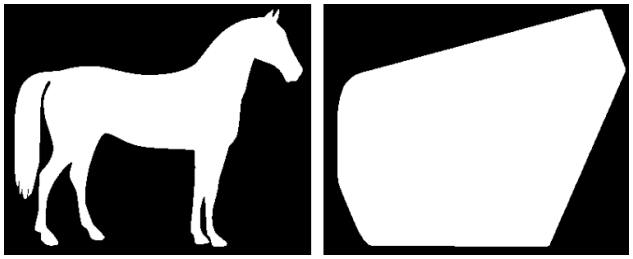


Figura 1.7: La cápsula convexa de la figura de un caballo en $\mathbb{R}^2$

Con esta definición queda claro que, a priori, C_r (la relajación lineal) no tiene por qué ser la cápsula convexa de C (con parte entera). De acuerdo, C_r es un conjunto convexo que contiene a C pero tal vez no sea el más chico.

¿Qué pasaría si consiguiéramos la cápsula convexa de C, $Conv(C)$? Para responder esto usamos dos resultados:

Teorema 6. *Todos los puntos extremos de $Conv(C)$ son puntos extremos del conjunto C original.*

Demostración
Recordemos que los puntos extremos de un conjunto convexo son puntos que no son combinación convexa de otros puntos del mismo conjunto. Sea x un punto extremo de $Conv(V)$ pero no extremo en C. Entonces x es combinación convexa de dos puntos x_1, $x_2 \in C$ (pertenezca x a C o no). Pero x_1 y x_2 van a pertenecer a $Conv(C)$ haciendo que x no sea un extremo de $Conv(C)$, lo cual es un absurdo. □

Teorema 7 (Krein-Milman)**.** *Todo conjunto compacto y convexo en un espacio euclideano es igual a la cápsula convexa de sus puntos extremos.*

Con lo cual tendríamos que si conseguimos la cápsula convexa de C, vamos a

tener

$$
\begin{aligned}
&Min\{c^T x + d^T y \ , \ [xy] \in C\} = \\
&Min\{c^T x + d^T y \ , \ [xy] \in Extremos(C)\} = \\
&Min\{c^T x + d^T y \ , \ [xy] \in Extremos(Conv(C))\} = \\
&Min\{c^T x + d^T y \ , \ [xy] \in Conv(C)\}
\end{aligned}
$$

¿De qué nos sirve esto? Si tenemos un problema mixto, factible y acotado, y además tenemos una caracterización de la cápsula convexa del conjunto de factibilidad podemos correr *simplex* sobre este nuevo conjunto como si fuera una optimización lineal, pero con la seguridad de que el óptimo que hallemos va a pertenecer al conjunto original, es decir, las variables que tengan que ser enteras lo serán.

La gran contra es que encontrar un conjunto de restricciones que caractericen a la cápsula convexa del conjunto de factibilidad de un problema mixto es muy difícil. En general resulta ser tan difícil como resolver el problema mismo.

Pero no todo está perdido. Podemos usar otras técnicas que tiendan a buscar extremos con variables enteras.

1.3.1. *Branch & Bound*

El método de *branch & bound* es un tipo de programación dinámica, donde vamos a intentar dividir el problema mayor en una serie de sub-problemas y de llevar una memoria de resultados obtenidos para tomar decisiones futuras.

Supongamos que tenemos un problema de programación entera, aunque podemos hacer exactamente los mismos pasos para el caso de programación mixta.

$$
\begin{aligned}
&Minimizar \ d^T y \\
&Sujeto \ a: \ Dy = b \\
&\qquad\qquad y_i \in \mathbb{Z}_{\geq 0} \quad \forall i
\end{aligned}
\tag{P}
$$

Lo primero que haremos es considerar la *relajación lineal* del problema. Esto es:

$$
\begin{aligned}
&Minimizar \ d^T y \\
&Sujeto \ a: \ Dy = b \\
&\qquad\qquad y_i \in \mathbb{R}_{\geq 0} \quad \forall i
\end{aligned}
\tag{P_0}
$$

es decir, las coordenadas de y son reales no negativos. Lo primero para observar es que el óptimo de este problema va a ser mejor que el original. Esto es porque, si llamamos $C = \{y \in \mathbb{Z}^n_{\geq 0} \,/\, Dy = b\}$ y $C_0 = \{y \in \mathbb{R}^n_{\geq 0} \,/\, Dy = b\}$ es claro que $C \subseteq C_0$ por lo que cualquier óptimo en C será factible en C_0 aunque no necesariamente al revés.

Otro dato a observar, a veces olvidado, la solución de P_0 puede no estar ni cerca del problema P. Pero seguro nos va a servir como cota.

Los pasos a seguir serán entonces los siguientes: resolvemos con *simplex* el problema relajado P_0. Si en mi solución y^* todas sus coordenadas son enteras entonces resuelve mi problema original y no tengo que seguir. Si no pasa esto, entonces existe un índice $i \in \{1, \ldots, n\}$ para el cual la coordenada y_i^* no es entera. Llamemos $C_1^- = \{y \in C_0 \,/\, y_i \leq \lfloor y_i^* \rfloor\}$ y $C_1^+ = \{y \in C_0 \,/\, y_i \geq \lceil y_i^* \rceil\}$ y pasamos a resolver los nuevos problemas

$$min\{c^T x + d^T y \,, con\, [x\ y] \in C_1^-\} \tag{P_1}$$
$$min\{c^T x + d^T y \,, con\, [x\ y] \in C_1^+\} \tag{P_2}$$

Un par de puntos a tener en cuenta:

- estas soluciones no pueden sino empeorar la solución de P_0, ya que $C_1^- \subseteq C_0$ y $C_1^+ \subseteq C_0$,
- las soluciones que obtengamos van a ser distintas a la anterior ya que en ambos casos la variable y_i está obligada a ser distinta de y_i^*,
- al resolver estos dos problemas no perdemos ninguna solución factible del problema original P

Siguiendo recursivamente con esta estrategia vamos a crear un árbol de problemas donde cada problema tiene un óptimo peor al de su padre pero está "más cerca" de ser una solución entera en las variables y.

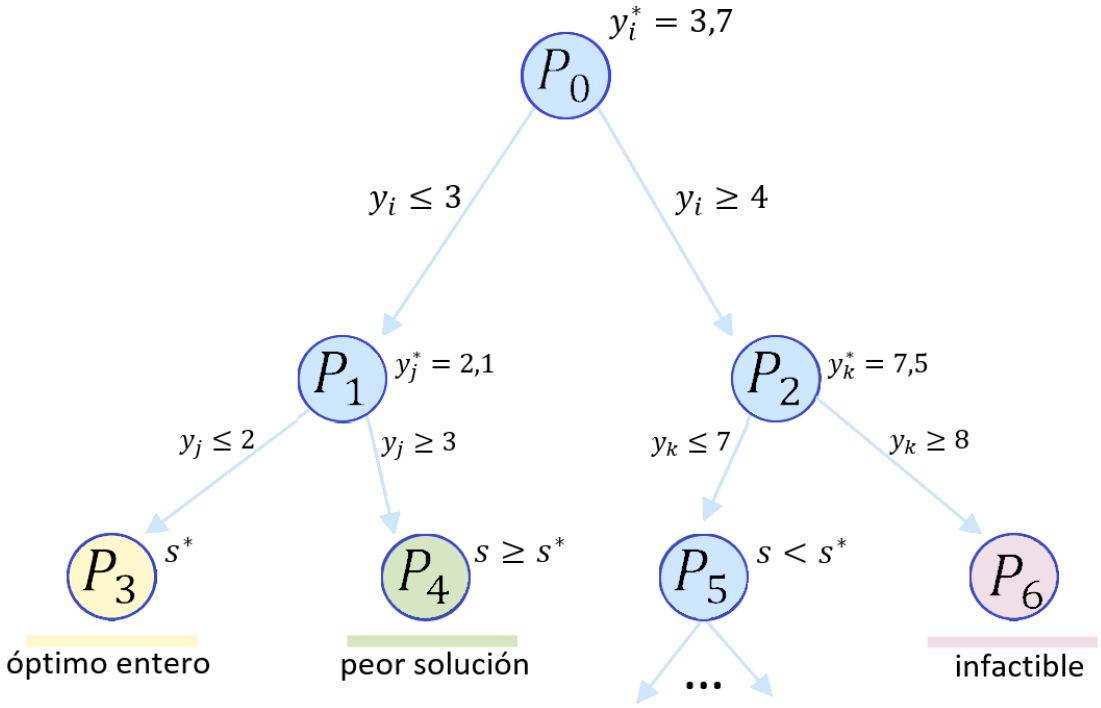
Cada vez que ramifiquemos un problema y resolvamos los nuevos subproblemas nos vamos a encontrar con uno de los siguientes escenarios:

1. El problema tiene solución entera y por lo tanto es factible del problema original. Ramificar sólo empeoraría la solución, por lo que frenamos esa rama.
2. El problema es infactible, no hay solución, cualquier ramificación seguiría siendo infactible.
3. La solución del problema no es entera por lo que podemos seguir ramificando.

Este esquema funcionaría perfectamente y una vez recorridas todas las opciones del árbol obtendríamos una o más hojas con soluciones enteras y podríamos elegir la mejor. Solo una cuestión: el árbol crece exponencialmente y ya con pocas variables se convertiría en impráctico.

Aquí es donde vamos a usar inteligentemente la información ya obtenida. La idea es simple, una vez que encontramos la primera solución entera vamos a guardar su valor. Llamémoslo $\mathbf{s}^*$. Al ser una solución factible, el óptimo de P tiene que tener un valor mejor que este, por lo que nos va a servir como cota superior. Además, si recordamos que ramificar empeora la solución podemos decidir ramificar inteligentemente. En el paso **3)**, podemos preguntarnos si tiene sentido seguir o no. Si la solución de nuestro subproblema (que además no es entero) ya es peor que $\mathbf{s}^*$ no tiene ningún sentido ramificar y podemos cortar esa rama.

Figura 1.8: El algoritmo de *branch & bound*



A su vez, a medida que vayamos encontrando soluciones enteras, vamos a ir actualizando el valor de $\mathbf{s}^*$. Al final, cuando no podamos seguir ramificando, tendremos que $\mathbf{s}^*$ es el valor del óptimo en el problema original P.

La pregunta de dónde ramificar es clave para este problema. Suponiendo que en el óptimo de un subproblema tenemos varias coordenadas no enteras, ¿cuál elegir para ramificar? (también podría ramificarse no en una variable)

Para esto hay distintas estrategias, algunas de ellas:

- Ramificar en la variable "menos entera", es decir, la que tiene un valor en el óptimo con parte entera más cercana a 0,5.

- Llevar un historial para cada variable de cuánto afectó en la función objetivo al elegirla para ramificar. Con esa información, ir eligiendo la mejor. Observar que al principio no se va a tener ninguna información pero se sabrá cada vez más con el paso de las iteraciones.

- Como el caso anterior pero testear cuánto va a servir elegir una variable **antes** de ramificar. Esto puede ser muy pesado computacionalmente, pero se puede testear solo sobre algunas variables sospechadas de ser mejores.

1.3.2. Desigualdades válidas y planos cortantes

La idea de usar planos cortantes es la de "recortar" el conjunto de factibilidad hasta que el extremo óptimo tenga valores enteros en las variables que queremos que así lo sean.

Primero definamos algunos conceptos:

Definición 7. *Sea P un problema de optimización mixta. Decimos que una restricción r es una* ***desigualdad válida*** *si al agregarla al problema no se elimina ninguna solución factible.*

Definición 8. *Sea P un problema de optimización mixta. Sea C el conjunto de factibilidad de P y sea C_r el conjunto de su relajación lineal. Sea x_0 un punto de C_r pero no de C (usualmente el óptimo de la relajación lineal). Decimos que una restricción r es un* ***plano cortante para x_0*** *si es una desigualdad válida del problema y además elimina a x_0.*

Los planos cortantes pueden ayudar muy fuertemente a resolver un problema de programación lineal mixta. Le permiten al *solver* trabajar sobre un conjunto mucho más acotado de factibilidad sin perder soluciones.

Conocer la estructura del problema puede ser muy útil a la hora de agregar desigualdades válidas. Aunque uno crea que "no se está agregando información nueva", lo cierto es que puede hacer que el *solver* itere muchas veces menos, llegando a un óptimo factible en un tiempo considerablemente menor. Aún así, existen técnicas que permiten encontrar planos cortantes de manera general, sin tener en cuenta la estructura del problema.

Una de esas técnicas son los *cortes de Gomory*

Cortes de Gomory

Sea

$$\begin{aligned} &Minimizar\ c^T x \\ &Sujeto\ a:\ Ax = b \\ &\qquad\qquad x_i \in \mathbb{Z}_{\geq 0} \quad 1 \leq i \leq n \end{aligned} \tag{P}$$

Un problema lineal entero (el resultado servirá también para problemas mixtos) y sea

$$\begin{aligned} &Minimizar\ c^T x \\ &Sujeto\ a:\ Ax = b \\ &\qquad\qquad x_i \in \mathbb{R}_{\geq 0} \quad 1 \leq i \leq n \end{aligned} \tag{P_r}$$

su relajación lineal. Llamemos x^* al óptimo de este último. Por *simplex* sabemos que si elegimos una base $B \subseteq \{1, \ldots, n\}$ de A obtenemos

$$x_B = A_B^{-1}b - A_B^{-1}A_N x_N$$

Donde $N = \{1, \ldots, n\} \setminus B$. Sea x^* la solución óptima y supongamos que existe $i \in B$ con $x_i^* \notin \mathbb{Z}$ (si no, tenemos la solución del problema original). Por la ecuación anterior sabemos que cualquier solución factible del problema entero cumple $x_B = A_B^{-1}b - A_B^{-1}A_N x_N$. En particular su coordenada i va a ser entera:

$$A_{i\cdot}^{-1}b - \sum_{j \in N} A_{i\cdot}^{-1}A_{\cdot j}x_j \in \mathbb{Z}$$

El resultado se mantiene entero si resto la parte entera de $A_{i\cdot}^{-1}b$ (por que es un entero) y si resto la parte entera de $A_{i\cdot}^{-1}A_{\cdot j}$ multiplicada por cada x_j (recordemos que si estamos en una solución, cada x_j es entero).

$$A_{i\cdot}^{-1}b - \lfloor A_{i\cdot}^{-1}b \rfloor - \sum_{j \in N} \left(A_{i\cdot}^{-1}A_{\cdot j} - \lfloor A_{i\cdot}^{-1}A_{\cdot j} \rfloor\right) x_j \in \mathbb{Z}$$

Pero como $0 \leq a - \lfloor a \rfloor < 1$ cualquiera sea el a, nos queda algo menor a 1, menos una suma de cosas positivas. por lo que

$$A_{i\cdot}^{-1}b - \lfloor A_{i\cdot}^{-1}b \rfloor - \sum_{j \in N} \left(A_{i\cdot}^{-1}A_{\cdot j} - \lfloor A_{i\cdot}^{-1}A_{\cdot j} \rfloor\right) x_j < 1$$

Y al ser entero, podmeos además decir

$$A_{i\cdot}^{-1}b - \lfloor A_{i\cdot}^{-1}b \rfloor - \sum_{j \in N} \left(A_{i\cdot}^{-1}A_{\cdot j} - \lfloor A_{i\cdot}^{-1}A_{\cdot j} \rfloor\right) x_j \leq 0$$

o

$$\sum_{j \in N} \left(A_{i\cdot}^{-1}A_{\cdot j} - \lfloor A_{i\cdot}^{-1}A_{\cdot j} \rfloor\right) x_j \geq A_{i\cdot}^{-1}b - \lfloor A_{i\cdot}^{-1}b \rfloor$$

Esta última es una desigualdad válida para P pero violada por x^*. Recordemos que x^* tiene todas sus variables $x_j = 0 \; \forall j \in N$ y $x_i = A_{i\cdot}^{-1}b$ no entera. Por lo que nos queda $0 \geq A_{i\cdot}^{-1}b - \lfloor A_{i\cdot}^{-1}b \rfloor > 0$ que es un absurdo.

Entonces hallamos un plano cortante para x^* del problema P. Observar que se pueden crear distintos planos cortantes dependiendo de por cuál coordenada no entera de x^* se decida empezar. Existen distintas estrategias para decidir cuál es el mejor corte.

Otra cosa a notar es que si restamos la igualdad $x_i + \sum_{j \in N} A_{i\cdot}^{-1}A_{\cdot j}x_j = A_{i\cdot}^{-1}b$ al corte, nos queda

$$x_i + \sum_{j \in N} \lfloor A_{i\cdot}^{-1}A_{\cdot j} \rfloor x_j \leq \lfloor A_{i\cdot}^{-1}b \rfloor$$

.

Este corte, además, tiene lado derecho entero, por lo que al agregarlo al problema P nos suma una variable de holgura a la que podemos pedir integralidad, pudiendo iterar el mismo procedimiento bajo las mismas hipótesis.

De hecho, Gomory demostró [18] que tomando ciertas decisiones este es un algoritmo que finaliza en finitos pasos, hallando una solución entera.

Sin embargo, Gomory mismo nos alerta sobre esta técnica: puede ser muy lenta y muy inestable numéricamente.

No fue hasta mediados de los noventa donde este pensamiento se dio vuelta. Gracias a Gérard Cornuéjols y algunos colegas que descubrieron que combinar cortes con *branch & bound* resultaba en muy buenos resultados. [19]

1.3.3. *Branch & Cut*

El concepto general de un algoritmo de *branch & cut* es el de usar *branch & bound* pero ayudándose con cortes antes de cada ramificación. Esto es:

1. Resolver la relajación lineal del problema. Obtener x^*.
2. Buscar planos cortantes que eliminen a x^*. Agregar las restricciones al problema.
3. Volver al punto 1 y repetir las veces que se considere necesario.
4. Ramificar y volver al punto 1 para cada uno de los subproblemas.

Lo que logramos es agregar un refinamiento antes de ramificar, lo cual nos ayudará a encontrar mejores soluciones más rápido. Sin embargo, como encontrar planos cortantes puede resultar complicado hay que buscar una estrategia para balancear esta dificultad con la mejora que potencialmente nos traerá.

1.3.4. *Branch & Price*

El método de *branch & price* va a valerse de nuevo del *branch & bound* pero en lugar de intercalar con la técnica de planos cortantes lo hará con la de generación de columnas.

La generación de columnas se suele usar cuando los problemas son especialmente grandes. En estos casos se asume que la mayor parte de las variables va a ser no básica en el óptimo, por lo que pueden asumirse iguales a cero y no tenerlas en cuenta explícitamente.

Lo que se hace, por lo tanto, es empezar eligiendo un subconjunto de variables del problema original (el problema **maestro**) para ser tenidas en cuenta explícitamente. Una vez elegidas, se resuelve el problema solo para esas variables (el problema maestro restringido) y se crea un subproblema, llamado ***pricing problem***. Este problema auxiliar va a decidir cuál es la columna óptima para ingresar a la base y es un problema de optimización lineal en sí mismo.

Una vez que se decidió cuál(es) columna(s) va(n) a entrar a la base se agrega(n) al problema maestro restringido y se itera. Hay que tener en cuenta que en cada iteración el subproblema asociado cambia, por lo que debe resolverse desde cero.

El bucle se va a cortar cuando no haya más columnas con costo reducido negativo. Esto querrá decir que el óptimo del problema maestro restringido no puede mejorar y las columnas no tenidas en cuenta son efectivamente no básicas para el problema maestro. Y aquí entra la parte *branch* del nombre: la ramificación. Si la solución encontrada no es entera se dividirá como en casos anteriores en dos subproblemas y se volverá a empezar, teniendo en cuenta los valores óptimos obtenidos.

Con todas estas herramientas podremos atacar nuestro problema específico, aunque necesitaremos un potente *solver* y más importante, un buen modelo que refleje el problema que pretendemos resolver.

Capítulo 2

El Traveling Tournament Problem

El *traveling tournament problem*, comúnmente abreviado **TTP**, es una clase de problemas propuesto por Kelly Easton, George Nemhauser y Michael Trick en 2001 [1].

Inspirados en la liga de *Baseball* de los Estados Unidos (MLB), observaron que los equipos querían minimizar la distancia viajada, pero a su vez mantener ciertas restricciones tradicionales del armado de *fixtures*. Por ejemplo, es improbable que un equipo quiera jugar 5 partidos seguidos de visitante, aún cuando esto casi con certeza ayudará a disminuir la distancia viajada al final del torneo. A su vez, al estar fuertemente relacionados por el torneo, no son obvias las decisiones a tomar para que todos los equipos logren sus objetivos sin romper la estructura. ¿Qué pasa si cada equipo arma el torneo de la manera más egoísta posible? ¿puede esto resultar en un *fixture* bueno para todos?

Quizás para responder esto y para entender el **TTP** en general primero sea necesario hablar de un problema asociado, el **TSP**.

2.1. El problema del viajante de comercio

El problema del viajante de comercio (o **TSP** por sus siglas en inglés) es un problema clásico de la programación matemática que busca, dado un conjunto de ciudades y las distancias entre ellas, encontrar una ruta de distancia mínima que visite todas las ciudades y regrese a la ciudad de origen. De ahí su nombre: es el circuito que querría tomar un comerciante que quiere pasar una sola vez por cada ciudad y regresar luego a su casa.

Otra forma de pensar el problema es como un problema de grafos. Si pensamos que tenemos un grafo pesado no dirigido, con cada vértice representando una ciudad y cada arista el camino entre las ciudades, el problema pasa a ser el de encontrar un ciclo Hamiltoniano con peso mínimo. Usualmente se modela con un grafo completo (todos los vértices están conectados), poniendo un peso suficientemente grande sobre una arista si el correspondiente camino no existiese.

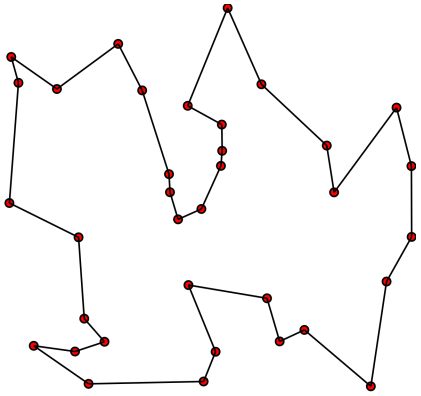


Figura 2.1: Ruta de distancia mínima para un conjunto de 35 ciudades

El **TSP** es un problema *NP-hard*, aun restringiendo la distancia a una euclidiana sobre el plano, o sacando la restricción de visitar solo una vez cada ciudad. Sin embargo, ha sido ampliamente estudiado y perfeccionado debido a sus aplicaciones en problemas como la recolección de basura, el tendido eléctrico o en lo que nos compete, ayudar a resolver problemas de armado de *fixtures*.

2.2. El TTP

En cierta forma, el problema del **TTP** parece ser una mezcla del **TSP** y de los problemas de *Sports Scheduling* con restricciones sobre los patrones de localías.

La programación entera resuelve muy bien el **TSP** mientras que con *Constraint Programming* se resuelven bien los problemas de factibilidad de torneos con restricciones en las localías. Para poner algún ejemplo, en 2004 se resolvió el **TSP** para 24.978 ciudades de Suecia [2], y problemas de patrones de localías se han resuelto con *Constraint Programming* para torneos de más de 30 equipos [3].

Es decir, ambas ramas están avanzadas en sus respectivos campos, y atacan de maneras exitosa sus respectivos problemas. Pero el **TTP** parece escapársele a las dos, ya que mezcla elementos de una y de otra. Tanto es así, que incluso instancias pequeñas, digamos de 6 equipos, son difíciles de resolver.

Es por esto y por lo simple de su formulación que esta clase de problemas supone un gran atractivo tanto para el campo teórico como para el práctico. Y es que, luego de su presentación formal, se encontró gran cantidad de ejemplos para su aplicación en la vida real.

Para poder aplicarlo necesitamos torneos que tengan un sistema de *doble round robin*, esto es, un torneo donde cada equipo juegue contra cada rival exactamente dos veces: una de local y otra de visitante. Ejemplos que presentan este sistema son la MBL, la liga Argentina de Basquet, la de vóley entre otros.

El **TTP** intenta en estos casos resolver lo siguiente: Dados **n** equipos, con **n** par, hallar un *fixture* de tipo *double round robin*, donde se minimicen las distancias recorridas por los equipos. Además, se debe cumplir que la cantidad de partidos consecutivos que un equipo juega de visitante esté siempre entre L y U, constantes fijas del problema.
Así queda presentado formalmente:

Input:

- Cantidad de equipos **n** par.
- Límites inferior y superior para el tamaño de los viajes $1 \leq \mathbf{L} \leq \mathbf{U}$.
- Matriz de distancias $\mathbf{D} \in \mathbb{R}^{n \times n}$.

Output:

- Un *fixture* de tipo *double round robin* que minimiza la distancia recorrida por todos los equipos, donde ningún equipo juega de visitante menos de **L** ni más de **U** partidos consecutivos

Aunque lo más usual es fijar $L = 1$ y U según la disposición de los equipos (3 o 4 en general), los cambios en L y U llevarán a diferentes e interesantes problemas. $L = U = 2$, por ejemplo transforma el problema en uno trivialmente infactible, ya que resulta imposible que cualquier equipo juegue sus $n - 1$ impar partidos de visitante en viajes de a 2. $L = U = 1$ es otro caso interesante. Si cada equipo debe regresar a su ciudad luego de cada partido de visitante (aun jugando dos partidos de visitante consecutivos), la distancia total recorrida por cada equipo se vuelve constante, y el problema se convierte en uno de solo factibilidad. Además, es fácil notar que, por desigualdad triangular, el valor de la distancia total recorrida por los equipos:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} d(i, j)$$

(incluyendo casos de D no simétrica) resulta una **cota superior** al problema, para cualquier L y U que lo vuelvan factible. Encontrar cotas a la solución del problema resulta muy útil a la hora de la ejecución de un algoritmo de resolución. Si las cotas son suficientemente buenas, pueden acelerar el proceso de búsqueda.

Una **cota inferior** se encuentra también fácilmente. Si llamamos, para un n y D, la distancia total recorrida en el óptimo como $S_{L,U}$ ($+\infty$ si es infactible) entonces tenemos que $S_{1,U} \leq S_{1,U'}$ cuando $U \leq U'$. Más aún, $S_{L,U} \leq S_{L',U'}$ si $L \geq L'$ y $U \leq U'$.

Es por esto que el caso $L = 1$, $U = n - 1$ se vuelve muy interesante, ya que $S_{1,n-1}$ va a representar una cota inferior universal del problema. Y aquí es cuando retomamos el **TSP**. Si permitimos que cada equipo viaje cuanto quiera, cada uno

va a intentar realizar un único circuito visitando una vez cada rival, minimizando la distancia. Esto es, resolver el **TSP**. Pero el **TSP** tiene el foco en un solo viajante, y no en su relación con el resto. Rápidamente entrarán en conflicto unos equipos con otros: cuando el equipo A visita al equipo B, B debería jugar de local en esa fecha y por lo tanto no estar de viaje. Esto vuelve rápidamente el problema en uno más difícil de lo que parece. Incluso en casos donde el **TSP** es trivial, resolver el **TTP** puede resultar desafiante.

Un caso ejemplar de esto son las llamadas *Instancias circulares*. En ellas, los equipos están dispuestos en un círculo, numerados del 0 al $n-1$. Y se cumple

$$d(i,j) = \text{mín}\{i-j \pmod n\ ,\ j-i \pmod n\}$$

El **TSP** en este caso es obvio (y único), y aún así no está claro que sea fácil resolver el **TTP** asociado.

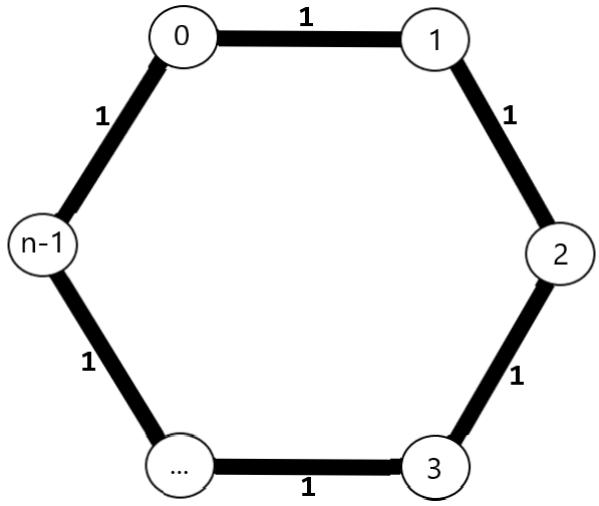


Figura 2.2: Instancia circular del **TSP** con n ciudades

¿Qué otros ejemplos de uso hay del *Traveling Tournament Problem*?

2.3. Instancias del TTP

Según la matriz de distancia, según la cantidad de equipos y restricciones extras se consiguen nuevas familias de instancias, algunas de ellas ampliamente estudiadas por la comunidad científica. Ya sea por su interés puramente teórico o por su específica aplicación práctica a torneos existentes, cada una de estas instancias y sus resoluciones ayudan a entender el problema y a ampliar las herramientas para poder atacarlo.

2.3.1. Distancia constante

Quizás la instancia más simple de todas, se trata de crear una matriz de distancias con puros unos exceptuando los ceros de la diagonal principal.

Esta instancia está lejos de ser aplicable a un problema real, para el cual los n equipos necesitarían habitar un espacio de al menos $n-1$ dimensiones. Sin embargo aporta interesantes conocimientos sobre el problema en general.

Por ejemplo, Ribeiro y Urrutia [6] demostraron que minimizar la distancia para el **TTP** de distancia constante era equivalente a maximizar la cantidad de *breaks* por equipo.

El concepto de *break* es clave en el *sports scheduling* en general. Decimos que un equipo tiene un *break* si juega dos partidos consecutivos de local (*break* de local) o dos partidos consecutivos de visitante (*break* de visitante).

Algo interesante a notar es que, cuando se miran todos los equipos de un torneo, el número de *breaks* de local es siempre igual al número de *breaks* de visitante, por lo que minimizar (o maximizar) uno u otro es equivalente.

En general se intenta minimizar el número de *breaks* para cada equipo. Es comprensible que se desee, por comodidad, por equidad competitiva, que la cantidad de *breaks* sea lo más cercana posible a cero, es decir, que la alternancia sea lo más perfecta posible (aunque esa alternancia perfecta en este tipo de torneos es matemáticamente imposible). Sin embargo, es obvio que el minimizar distancia y minimizar *breaks* llevan a la función objetivo en direcciones opuestas. Lo que no es obvio es que en el caso de la distancia constante, los problemas son *exactamente* opuestos, como demostraron Ribeiro y Urrutia. Esto es, minimizar distancias es lo mismo que maximizar *breaks*.

Así es que, aunque parezca un problema puramente teórico, el planteamiento de un **TTP** de distancia constante puede ser utilizado para problemas bien concretos donde se busque maximizar *breaks* de visitante como una estrategia para disminuir las distancias recorridas.

En el caso de **TTP** sin restricciones en el largo de viajes posibles, además Ribeiro y Urrutia demuestran que el óptimo es $n^2 + \frac{n}{2} - 1$ para n equipos. Para el caso donde el tamaño máximo de viaje permitido es 3, se han estudiado y se siguen estudiando casos de hasta 24 equipos.

2.3.2. Circulares

Las ya nombradas instancias circulares son otro ejemplo de instancias teóricas interesantes. Su más sorprendente punto es la dificultad que tiene comparado con su equivalente **TSP**. Es el contraejemplo utilizado a la idea de que de un **TSP** sencillo se obtiene un **TTP** también sencillo [1].

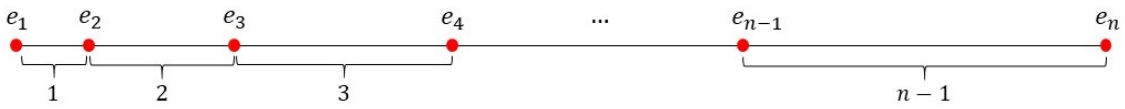
Las instancias circulares con n par han sido estudiadas encontrando mejores soluciones progresivamente. Pero para n tan pequeño como 12 (y con un tamaño máximo permitido de viaje igual a 3) ya hay una diferencia entre la mejor cota inferior (388) y la mejor solución factible encontrada (404). Aún está abierta la pregunta de si este es el óptimo o no.

2.3.3. Lineales

Similar a la circular, la idea aquí es que los n equipos van a estar distribuidos a lo largo de una recta, con una distancia igual a 1 para cada par de equipos adyacentes [7]. Es decir, $D(i, j) = |j - i|$.

Además hay una versión de distancia creciente. Los equipos, al igual que antes, están distribuido sobre una recta pero además la distancia va aumentando en 1 cada vez. Esto es, $D(i, i + 1) = i$. Este modelo refleja, en cierto sentido aunque artificial, el concepto de equipos "lejanos" o "cercanos". Veremos en el problema del vóley que esta idea aparece entre los equipos de Buenos Aires y algunos equipos muy al norte o al sur del país.

Figura 2.3: n equipos dispuestos sobre una recta, con distancia creciente



2.3.4. Galaxias

Una de mis instancias favoritas, el **TTP** entre galaxias. En el límite entre las instancias teóricas y las prácticas, estas instancias imaginan un torneo a disputarse entre la tierra y otros 39 exoplanetas de distintas galaxias [8].

Las distancias aquí se miden en años luz por lo que hallar un óptimo en distancias es crítico. Puede marcar la diferencia entre que el equipo campeón sean nietos de los que jugaron el primer partido en lugar de bisnietos.

Desde el punto de vista de la investigación, el problema presenta un desafío interesante al ser la matriz de distancia una con distancias en tres dimensiones, algo que no ocurre en ninguna otra instancia del **TTP**.

Aquí de nuevo, a partir de $n = 12$ no se tiene seguridad de haber llegado a un óptimo. Sin embargo, se siguen estudiando hasta hoy en día. En 2018, por ejemplo, Hirano, Abe e Imahori encontraron mejoras para n igual a 18, 24, 32 y 36.

Quizás cuando logremos construir las naves y lancemos la propuesta del picadito interestelar ya habremos encontrado el óptimo del problema y podremos minimizar las distancias.

2.3.5. *National League*

La *National League* (NL), constituida por la mitad de los equipos presentes en la *Major League Baseball* (Liga mayor de béisbol de los Estados Unidos), cuenta con 16 equipos. Introducido por Easton, Nemhauser y Trick en el *paper* que dio inicio a la idea de *Traveling Tournament Problem* [1], el problema llamado "NLn", con n

un número par entre 4 y 16 consiste en resolver el **TTP** para los primeros n equipos de la *National League*.

Las reglas son:

- Se debe crear un *double round robin* en $2(n-2)$ fechas.
- $L = 1, U = 3$. Esto es, los viajes no pueden ser de longitud mayor a 3. El caso $U = \infty$ también se estudia.
- No puede haber "repetidores", es decir, A vs. B seguido inmediatamente de B vs. A
- El objetivo es minimizar distancias.
- Las distancias son de vuelo, son simétricas, y se supone que cada equipo comienza y finaliza el torneo en casa.

Existen instancias ya resueltas, como por ejemplo NL6, con un óptimo igual a 23916 Siendo el @ significa que el equipo jugará de visitante. Por ejemplo ATL juega

Fecha	ATL	NYM	PHI	MON	FLA	PIT
1	@FLA	@PHI	NYM	PIT	ATL	@MON
2	@PHI	PIT	ATL	@FLA	MON	@NYM
3	MON	@FLA	PIT	@ATL	NYM	@PHI
4	NYM	@ATL	@MON	PHI	@PIT	FLA
5	@PIT	PHI	@NYM	FLA	@MON	ATL
6	@MON	@PIT	FLA	ATL	@PHI	NYM
7	@NYM	ATL	MON	@PHI	PIT	@FLA
8	PIT	MON	@FLA	@NYM	PHI	@ATL
9	PHI	FLA	@ATL	@PIT	@NYM	MON
10	FLA	@MON	@PIT	NYM	@ATL	PHI

Cuadro 2.1: Óptimo para el NL6

de visitante contra FLA en la fecha 1.

Observar que de los 5 partidos a jugar de visitante, todos excepto NYM los juegan en grupos de 2 y 3, el máximo permitido. No solo es, sino que *parece* ser el óptimo.

Para el resto de los n se siguen encontrando mejoras. En julio del 2019, por ejemplo, Ben Sliméne logró hallar mejores soluciones en el caso $U = \infty$ para NL10, NL12, NL14 y NL16.

2.3.6. *National football league*

Otro de los torneos donde se aplica el **TTP** es la liga de fútbol americano de los Estados Unidos, la *NFL*.

También basado en la distancia por aire entre ciudades, el torneo (o al menos la instancia estudiada) cuenta con un total de 32 equipos. Esta instancia es obviamente demasiado grande para el **TTP** pero sirve como experimento.

En este caso en particular, se cuenta con un buen generador de **TSP** entre las ciudades, lo cual hace el trabajo mucho más fácil a la hora de hallar óptimos.

Igual que antes, instancias con menos que 32 equipos se suelen nombrar con "NFLn", siendo n la cantidad de equipos. Para NFL32 todavía hay unos 80 mil kilómetros de diferencia entre la mejor cota inferior y la mejor solución factible hallada. De más está decir que es una distancia enorme. Si el óptimo es tan bueno como la cota inferior valdrá la pena seguir estudiándolo para ahorrarse esa cantidad de kilómetros

2.3.7. Super14

En 1996 se crea el **Super12**, un campeonato que reunía 12 equipos de los 3 países con posiblemente el mejor rugby del mundo: Australia, Nueva Zelanda y Sudáfrica. El campeonato es considerado en casi todos lados como el más competitivo de rugby en el mundo.

En 2006 se decide agregar dos equipos al torneo, pasando a llamarse **Super14**. En 2011 la cantidad de equipos pasa a ser 15, y ya cansados de cambiar de nombre decidieron llamar al campeonato **SuperRugby**, nombre independiente de la cantidad de equipos y que mantiene hasta hoy.

En el 2016, luego de haber mostrado Argentina su potencial en la copa mundial de rugby (había obtenido el cuarto puesto el año anterior, que se sumaba al tercer puesto del 2007), al país se le permitió sumar un equipo al **SuperRugby**. Así nace la primera franquicia de rugby profesional en Argentina, los *jaguares*.

Además se admitió en el campeonato a los *sunwolves*, sumando entonces al **SuperRugby** dos países: Argentina y Japón.

De todas formas, el torneo que se toma como ejemplo para resolver el **TTP** es el del año 2009 [9], por lo que ni Argentina ni Japón tienen el honor de tener un equipo entre los 14 estudiados.

Como un pequeño apartado, el **SuperRugby** estuvo cerca de ser el objeto de estudio de esta tesis, pero antes de empezar se encontraron dos contras muy fuertes.

- La primera es que en el rugby es crítico el tiempo de espera entre partido y partido. Al ser un deporte de contacto y donde el físico se pone tan al límite, jugar más de un partido por semana resulta casi inconcebible. Esto hace prácticamente imposible hacer viajes muy largos, ya que el descanso entre partido haría las estadías muy costosas, perdiendo el sentido original de abaratar viajes.

- La segunda es que los equipos neozelandeses estaban poco interesados en generar viajes largos. Dadas las cortas distancias, preferirían volver a casa entre partido y partido, y posiblemente ignorarían un *fixture* con viajes óptimos.

Esto muestra como a veces es tan importante conocer la parte humana y específicamente deportiva como la parte matemática y computacional. De nada sirve conseguir una solución óptima si al final será rechazada por los equipos.

2.3.8. Torneo de fútbol Brasileño

En el torneo de 2003 de fútbol de Brasil, el *"Campeonato Brasileiro de Futebol"*, 24 equipos realizaron un *double round robin*. Urrutia y Ribeiro vieron la posibilidad de resolver un **TTP** para este campeonato [10]. Encontrando una solución factible con un valor de 506433 en 2004, la cual fue luego mejorada por ellos mismos en 2005 con un valor de 503158, y más tarde llevada a 500756 en 2007 por Van Hentenryck y Vergados. El problema continúa abierto.

2.4. Variantes del TTP

Pero además de las instancias teóricas como pueden ser las circulares, que sirven para estudiar el problema, establecer cotas o simplemente como divertimento matemático, existen las variantes de tipo *práctico*. O más o menos práctico, como veremos en un ejemplo en particular. A pesar de que el planteo del **TTP** contempla diversos casos, gracias al parámetro en la cantidad de equipos (aunque obligatoriamente par) y a los parámetros L y U de largo de viaje, a veces queda corto para contemplar otros casos de torneos reales. Torneos que, por alguna u otra razón, no caen precisamente en la definición de **TTP**, pero entran en un campo de similitud con el **TTP**. ¿Podemos adaptar el **TTP** para que contemple estos casos?

2.4.1. Torneos con valor de *matchup*

El *double round robin*, aunque es una estructura de torneo ampliamente utilizada, a veces puede no ser la estructura buscada para un campeonato específico. El beneficio de esta forma es que presenta una idea de igualdad para todos los equipos: todos van a tener los mismos rivales (exceptuándose a uno mismo) y además tendrán la posibilidad de jugar de local y visitante. La localía, aunque a veces resulte extraño, es tomada en casi todos los deportes como un factor determinante a la hora de buscar equidad en la competencia.

Sin embargo, a veces el *double round robin* no es adecuado para el torneo que se tiene en mente. El principal motivo es el del largo del torneo: si uno sigue este esquema, con n equipos obtiene $(n-1)n$ partidos. Dependiendo del n y la cantidad de días que uno tenga disponible, puede ser necesario querer jugar más o menos partidos. En el caso de querer alargarlo muchas veces se logra creando una segunda fase de eliminatorias que además puede tener motivos deportivos. Pero a veces, como veremos en nuestro problema del vóley, no es suficiente.

Incluso hay casos en los que se va a optar por un *round robin* usual, o un *quadruple round robin*, este último preservando todo la equidad competitiva mencionada del *double round robin*.

En el caso de la liga profesional de béisbol japonesa, por ejemplo, 6 equipos se disputan 120 partidos entre ellos, en un total de 8 rondas [13].

Lo que se va a hacer es simplemente cambiar la restricción que pide que cada encuentro se realice exactamente una vez por una que pida el valor de *matchup* fijado para ese encuentro:

$$\sum_{k \in Fechas} x_{ijk} = C_{ij} \qquad \forall\, i, j \in Equipos,\ i \neq j$$

Donde x_{ijk} es una variable binaria que decide si el equipo i juega contra el equipo j en la fecha k, C_{ij} es una cantidad prefijada de encuentros entre i y j, con i de local. Notar que el *double round robin* tiene un valor de *matchup* de 1 para cada par de equipos, el *quadruple round robin* un valor de 2, y en el *round robin* simple se va a cumplir que

$$C_{ij} + C_{ji} = 1 \qquad \forall\, i, j \in Equipos,\ i \neq j$$

Notar que el valor de C_{ij} va a estar prefijado, y en este último u otros casos donde no haya simetría va a haber que decidir los valores de alguna manera. ¿Podemos delegar al *solver* a que tome esta decisión? Esto será estudiado en capítulos posteriores, cuando se vean las estrategias utilizadas para el torneo de vóley del torneo 2019/2020.

2.4.2. Torneos con tiempo relajado

Uno de los factores que más afecta a la factibilidad de los torneos con estructura de **TTP** es el del tiempo. Recordemos que en el **TTP** clásico, dados n equipos, cada uno de ellos jugará $2(n-1)$ partidos. Es decir, habrá efectivamente $2(n-1)$ fechas.

Esto es porque no existe el concepto de fecha libre, en cada fecha del torneo cada equipo juega un partido. Sin embargo, puede sernos útil querer agregar fechas al campeonato, donde los equipos pueden o no jugar un partido. Esto puede ayudarnos con la factibilidad o con el óptimo de varias maneras.

Por un lado, nos permite incluir fechas vacías, descansos para equipos específicos. Es cierto que en el **TTP** clásico el concepto de "fecha" no tiene por qué ser (casi nunca lo es) igual al concepto de día, y es posible incorporar descansos, o días sin juegos entre fecha y fecha. Pero si nuestra intención es que esos descansos o fechas libres se den para equipos específicos y no para todos, las fechas extra pueden ser de gran utilidad.

Por otro lado, podemos querer por algún motivo *desacoplar* los partidos. Con el **TTP** clásico va a ocurrir que todos jueguen su k-ésimo partido en la k-ésima fecha. Esto, aunque a nivel competencia puede ser buscado, a nivel de factibilidad, cuando entren en juego peticiones específicas de los equipos, puede volverse demasiado restrictivo. Agregando fechas en cambio, podemos comenzar a tener partidos entre dos equipos, donde uno estará jugando su quinta fecha y otro su sexta fecha por ejemplo.

Día	1	2	3	4
Equipo 1	@Equipo 2	@Equipo 3	Equipo 4	
Equipo 2	Equipo 1		@Equipo 3	@Equipo 4
Equipo 3	@Equipo 4	Equipo 1	Equipo 2	
Equipo 4	Equipo 3		@Equipo 1	Equipo 2

Cuadro 2.2: Un *fixture* con tiempo relajado. El día 3 el **Equipo 1** juega su tercera fecha contra el **Equipo 4**, que está a su vez jugando su segunda fecha

¿Cómo logramos esto? En general vamos a querer que la cantidad de fechas siga siendo un múltiplo de $n-1$ (por ejemplo $3(n-1)$) y pediremos, en lugar que haya un partido por fecha, que haya *al menos* un partido por fecha:

$$\sum_{j \in Equipos\, i \neq j} x_{ijk} + x_{jik} \leq 1 \qquad \forall\, i \in Equipos,\ \forall k \in Fechas$$

Donde, de nuevo, la x_{ijk} decide si el equipo i recibe al equipo j en la fecha k y por supuesto, el conjunto $Fechas$ es un conjunto con tiempo relajado.

Esta variante del **TTP** es muy utilizada en problemas reales y ha sido nombrada lo suficiente para lograr su nombre propio. Suele llamársela en la literatura del tema **TRTTP**, por sus siglas en inglés (*Time Relaxed Traveling Tournament Problem*) [11].

Quizás solo quede una pregunta girando, ¿cuán grave es que las fechas estén desacopladas? ¿está bien que, por ejemplo, no jueguen todos su último partido al mismo tiempo?

Otra vez, veremos más adelante cómo aplicar este método y cómo solucionar estas cuestiones en nuestro problema del vóley.

2.4.3. Torneos con un *schedule* parcial

Otro caso muy común en estos torneos, o en el contexto del *sports scheduling* en general es el de tener que completar un *schedule* parcial [12]. Por alguna razón podemos tener que se han fijado ciertos encuentros o ciertas localías y debemos completar el torneo manteniendo esos partidos prefijados. Esto puede deberse a eventos televisivos, a pedidos de los propios equipos, a motivos específicamente deportivos.

Cualquiera sea el motivo, se van a tener que solucionar dos cosas. La primera, ¿es *factible* un torneo de *double round robin* donde se respeten estos nuevos pedidos? Nada dice a priori que se pueda. Si el *schedule* parcial no se ha hecho de forma inteligente, las nuevas restricciones podrían entrar en conflicto consigo mismas. Y aquí puede aparecer un problema interesante: ¿se puede encontrar un torneo factible *lo más fiel posible* al *schedule* parcial? Aunque va a ser un problema de optimización, definitivamente va a ser discutible qué significa "lo más fiel posible" en este contexto.

Si en cambio nos convencemos que es factible podremos pasar a la segunda parte: ¿cuál es el *mejor* torneo que contenga estos partidos prefijados? Donde, aquí sí, seguimos midiendo la mejoría en términos de distancia global.

Algo a tener en cuenta: la solución no puede menos que empeorar en estos casos. Al **TTP** libre le estamos agregando restricciones, el conjunto de factibilidad estará contenido en el conjunto original por lo que a lo sumo obtendremos un resultado igual de bueno que sin las restricciones.

¿Cómo procedemos a resolver el nuevo problema? Supongamos que tenemos un conjunto F de triadas (i, j, k) que nos exigen que el equipo i reciba al equipo j en la fecha k (esto podría ser también menos estricto y pedir, por ejemplo, solo que i se enfrente a j en la fecha k, sin importar localía).

Entonces agregamos a las restricciones del **TTP** las nuevas restricciones

$$x_{ijk} = 1 \qquad \forall (i, j, k) \in F$$

Donde la variable binaria x_{ijk} decide si el equipo i recibe al equipo j en la fecha k.

Notar que al hacer esto estamos agregando una variable al problema y luego fijando su valor. Esto podría no ser una buena idea porque estamos agregando dimensiones a un problema cuando de hecho no las necesita.

Sin embargo, cualquier *solver* moderno realiza un preprocesamiento donde elimina las variables con valores prefijados, por lo que no hace falta preocuparse demasiado por esta forma aparentemente ineficiente de escribir el modelo.

De hecho, esta es otra razón por la cual se puede elegir prefijar valores y completar. Cuando el problema es muy grande y muy difícil de resolver, puede servir encontrar de manera heurística partidos que deberían jugarse y luego completar alrededor. Si la manera en la que se hizo la heurística es buena, uno puede esperar poco empeoramiento en optimalidad habiendo ganado mucho en tiempos de ejecución.

2.4.4. Torneos basados en rondas

Hasta ahora no hemos hablado del orden o la forma en la que se van a jugar los partidos. No pedimos ninguna estructura más allá de que cada equipo se enfrente a cada otro equipo dos veces, una de local y otra de visitante. Esto es,

$$\sum_{k \in Fechas} x_{ijk} = 1 \qquad \forall i, j \in Equipos\,,\ i \neq j$$

Donde la variable binaria x_{ijk} decide si el equipo i recibe al equipo j en la fecha k.

Ni siquiera estamos pidiendo que los equipos i y j no se enfrenten consecutivamente alternando localías, una restricción muy pedida en este tipo de torneos.

Lo que se suele usar en estos casos es el concepto de *rondas*. Esto es, partimos el torneo en 2 (o más, si además estamos en caso de *quadruple round robin* u otros valores de *matchup*) rondas. Cada ronda tendrá un subconjunto de fechas tal que

$F_1 \cup F_2 = Fechas$ y pediremos que los equipos i y j se enfrenten una vez en cada ronda. Esto es

$$\sum_{k \in F_1} x_{ijk} + x_{jik} = 1 \qquad \forall i, j \in Equipos\,,\ i \neq j$$

$$\sum_{k \in F_2} x_{ijk} + x_{jik} = 1 \qquad \forall i, j \in Equipos\,,\ i \neq j$$

No se busca en estos casos que haya simetrías como pueden ser los torneos espejados o el esquema francés por ejemplo.

Esta formato puede ser muy útil para torneos donde a mitad del mismo se prevea un intervalo de descanso (vamos a ver que esto ocurre en el torneo de vóley). En estos casos, parece ser razonable que los equipos lleguen con la misma cantidad de partidos jugados y con la experiencia de haberse enfrentado a todo el resto de los equipos.

Además, es muy común que a mitad de temporada se realicen partidos *all stars* con los mejores jugadores o mini torneos entre los mejores equipos del campeonato. Para que esto puede tener un sentido más real, es una buena idea que para ese entonces se haya cerrado una *ronda* y que la tabla de posiciones por ejemplo refleje resultados parciales más "justos".

Capítulo 3

El problema del torneo de Vóley

El torneo de la primera división de voleibol masculino de Argentina (**LVA** por "Liga de voleibol argentina", o alternativamente "Liga argentina de voleibol") es la competencia de máxima categoría de voleibol del país. La organización del torneo corre a cuenta, desde 2003, de la **ACLAV** (Asociación de Clubes Liga Argentina de Voleibol). Al jugarse usualmente entre noviembre y abril del año siguiente, para referirnos a una temporada específica nombramos dos años consecutivos. En su primera edición de 2003/2004 (antes no era organizado por **ACLAV**) hubieron 12 equipos participantes, pero este es un número que cambia temporada a temporada.

Temporada	Cantidad de equipos
2003-04	12
2004-05	12
2005-06	12
2006-07	12
2007-08	12
2008-09	11
2009-10	11
2010-11	12
2011-12	12
2012-13	10
2013-14	11
2014-15	10
2015-16	11
2016-17	11
2017-18	11
2018-19	10
2019-20	9

Cuadro 3.1: Cantidad de equipos por temporada, desde el 2003 hasta la actualidad.

Estos cambios (sobre todo en la paridad) hará que más adelante tengamos que modificar estrategias para crear el *fixture*.

La liga de voleibol argentina se compone de dos partes:

- La **fase regular**
- Los ***play-offs***

La **fase regular** es un *double round robin*, es decir, juegan todos contra todos dos veces, una vez de local y otra de visitante. Los 22 partidos a disputar (en el caso de 12 equipos) se dividen en grupos de a dos que se juegan en un jueves y sábado o viernes y domingo consecutivos, los denominados "*weekend*". Con lo cual estamos hablando de 22 semanas de juego. Los equipos irán quedando ordenados, según puntaje, en una tabla de posiciones. Al final de esta primera fase, los 8 primeros equipos pasarán a la fase de ***play-off***, quedando el resto eliminado.

Estos 8 equipos jugarán los cuartos de final, el 1º contra el 8º, el 2º contra el 7º, y así sucesivamente. Los cuatro ganadores jugarán las semifinales y los ganadores de estas, la final. El ganador de la final se consagrará campeón de la liga. Todos los certámenes de *play-off* se juegan al mejor de 5 partidos, lo cual hace que no haya demasiadas sorpresas en los resultados.

Pero ese no es el tema de este trabajo.

Es importante notar que en la fase de *play-offs* no hay aportes para hacer desde el lado del *Sports scheduling*; los partidos ya están definidos y no se pueden cambiar. Incluso el patrón de localías queda definido a partir de la tabla de posiciones de la fase regular. Es por esto que nos centraremos en la primera parte, la fase regular, donde el torneo podría llevarse a cabo de múltiples maneras.

3.1. Viejo enfoque

En 2007, gracias al trabajo conjunto de Flavia Bonomo, Andrés Cardemil, Guillermo Durán, Javier Marenco y Daniela Sabán [4] con la **ACLAV**, se comenzó a utilizar una nueva estrategia para la creación del *fixture*. La idea era intentar resolver el **TTP** pero de una manera que fuera ejecutable en un tiempo razonable y siguiendo las restricciones exigidas por la asociación.

El planteo se hizo entonces de la siguiente manera. Para evitar resolver un **TTP** con 12 equipos, que puede volverse muy complicado o directamente imposible ante pequeñas variaciones, se juntan los equipos en parejas por proximidad. Luego los pares de equipos (A_i, B_i), con $1 \leq i \leq \frac{n}{2}$ pueden tratarse como un nuevo equipo para cada i. Los partidos pasan a jugarse también de a pares, en los nombrados "*weekend*". Estas fechas dobles suelen ser en los pares de días jueves-sábado o viernes-domingo. Lo que se logra con esto es reducir tanto el número de fechas como el de equipos a la mitad.

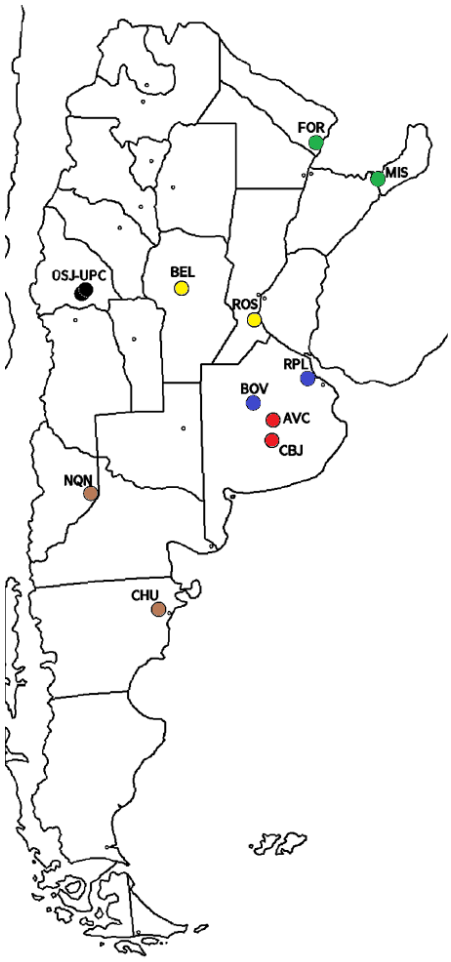


Figura 3.1: Los 12 equipos de la liga, emparejados por proximidad

Pero ¿qué significa que el equipo doble i visite al equipo doble j en la fecha doble k? Esto quedará siempre bien definido: A_i juega contra B_j, mientras que B_i lo hará contra A_j en el primer día de la fecha. En el segundo día los rivales se intercambiarán, jugando A_i contra A_j y B_i contra B_j (Ver cuadro 3.2). Es importante notar que, en el caso ideal en que la distancia entre los equipos de una misma pareja sea muy chica (o cero), visitar a ambos es prácticamente lo mismo que visitar a uno solo, volviendo el modelo de parejas un modelo que refleja muy bien la realidad.

¿Puede entonces modelarse el problema como un **TTP** de 6 equipos?
Sí. El concepto de partido está bien definido, la matriz de distancias queda también unívocamente definida, así como también el concepto de localía. Lo único que falta para que una solución de este nuevo **TTP** se pueda llevar a una solución del problema original es agregar una fecha intraparejas, donde A_i juegue contra B_i dos veces, alternando localía, para todo i.

Fecha **k** - día 1 (jueves)	Fecha **k** - día 2 (sábado)
A_i vs B_j B_i vs A_j	A_i vs A_j B_i vs B_j

Cuadro 3.2: La pareja (A_i, B_i) enfrenta a la pareja (A_j, B_j) en la fecha **k**

¿Qué se puede hacer en el caso en que los equipos no son pares?
En este caso (efectivamente, varios años se han tenido 11 equipos) un equipo quedará sin pareja. O lo que es lo mismo, se puede pensar que está emparejado a un equipo ficticio f y tratar al resto igual que antes. El único cambio que habrá es que cuando a un equipo le toque jugar contra f tendrá una fecha libre. Esto ocurrirá

dos veces para cada equipo en el torneo: cuando visite al equipo sin pareja y cuando este lo visite. En la matriz de distancia se asumirá que el equipo sin pareja está a distancia 0 de f y así se calcularán el resto de las distancias. En la fecha intraparejas, el equipo desemparejado tendrá una doble fecha libre.

Alcances y limitaciones

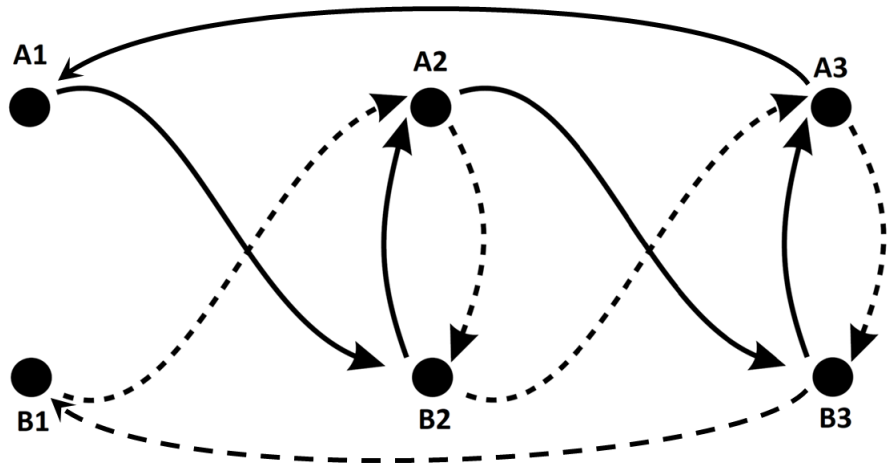
Este sistema de parejas se viene utilizando desde la temporada 2007-08 con muy buenos resultados. Al reducirse el problema a un **TTP** con 6 equipos, los tiempos de corrida son muy bajos, y el problema suficientemente maniobrable.

Sin embargo nos encontramos con varias limitaciones. La primera y más importante es que la solución óptima que se encuentre bajo esta estrategia no será el óptimo del problema original casi seguramente. La única forma de acercar los dos problemas es si las distancias intraparejas son cercanas a cero, y ni aun así quedaría claro que un óptimo implique el otro.

Otro de los problemas es que con esta estrategia estamos limitados a realizar viajes donde visitamos una cantidad par de equipos (exceptuando el viaje para el partido intrapareja). La posibilidad de visitar 3 equipos y luego volver queda totalmente descartada (salvo en el caso de tener una cantidad de equipos impar).

Se suma también que los partidos intraparejas se hacen sí o sí un mismo *weekend*, además de todos a la vez. Estos partidos, posiblmente sean entre equipos de una misma ciudad, partidos entre rivales"clásicos". Aunque quizás extrañamente deportivamente deseable, no tenemos opción de no hacerlo.

Figura 3.2: La pareja (A_1, B_1) visitando a la pareja (A_2, B_2), luego a la pareja (A_3, B_3) y finalmente volviendo a casa



Por último, es importante recordar que hasta aquí estamos restringidos a que los partidos se jueguen en los llamados *weekend*, que corresponden a un jueves y sábado usualmente. De aquí obtenemos que, en caso que la pareja (A_1, B_1) visite a la pareja (A_2, B_2) y luego a la (A_3, B_3) (ver Figura 3.2), que en el problema original equivale a que tanto A_1 como B_1 hagan un viaje visitando 4 equipos antes de volver a casa, estos dos equipos estarán fuera de casa durante el transcurso de dos *weekends* completos. Esto es, al menos **10** días fuera de casa para jugar 4 partidos. En general, para jugar $2n$ partidos afuera, necesitamos $3 + 7(n-1)$ días afuera. Esto incurre en gastos de hotel y otras cosas que, aunque no están medidos por nuestro modelo, son indeseados a la hora de armar un buen *fixture*.

3.2. Nuevo enfoque

Y he aquí la nueva estrategia. Hemos nombrado antes que la resolución de un **TTP** puro para 12 equipos puede resultar en algo muy difícil de resolver. Cuando uno encima agrega restricciones específicas de los clubes, fechas en las que no se puede jugar, campeonatos internacionales, requerimientos televisivos, el problema rápidamente se vuelve inmanejable. Pero dando uso de unas cuantas herramientas podemos acercarnos a una solución deseada.

La estrategia seguida en este trabajo va a ser una similar a la seguida en el armado de *fixture* para la Liga Nacional de Básquet en Argentina (LNB) [5].

Denominado allí como **TP-TRTTP** (por sus siglas en inglés: *Trip Preferences-Time Relaxed Traveling Tournament Problem*) será una variante del ya mencionado **TRTTP** (ver página 34), donde además de ser relajado en el tiempo hay "viajes preferidos" para hacer. Estos viajes serán proporcionados previo al armado del fixture por los equipos mismos.

Además, se resuelve el problema en dos etapas: una primera en la que se decide la secuencia de partidos a jugar junto con la localía y una segunda donde se distribuyen los partidos ya definidos en los días en que se lleva a cabo el torneo.

La primera gran diferencia de la nueva estrategia es que los equipos ya no estarán emparejados. Y lo que es aún más crítico, se elimina el concepto de *weekend* y pasa a poder jugarse cualquier día de la semana. Un sistema más parecido al de la **NBA**. Esto traerá mucha flexibilidad a la hora de organizar viajes: un equipo puede aprovechar estar fuera de casa y jugar más seguido. Aquí contamos con la suerte que el voleibol permite que se jueguen partidos con poco descanso entre uno y otro, algo que otros deportes no pueden hacer (ver página 32).

Por otro lado, esta decisión traerá también muchas más variables a la hora de crear un modelo matemático. Donde antes había 22 fechas habrá ahora alrededor de 150 *días* tomando ese papel. Este aumento en la cantidad de variables resulta absolutamente crítico.

Se decide entonces partir el problema en dos, el primero siendo un problema de optimización, casi igual al **TTP** pero relajado en el tiempo, el segundo un problema

básicamente de factibilidad. Vamos a verlo.

Primera etapa

En la primera etapa se resuelve una relajación del **TTP** con $3(n-1)$ fechas (con un factor de 3 en lugar de 2). Esto permite desacoplar los partidos y que solo en caso de viaje se juegue lo suficientemente seguido. La matriz de distancia es obtenida en principio por la distancia geodésica entre los estadios de los equipos. El conjunto de viajes posibles es administrado por los mismos equipos. Esto permite dos cosas: disminuir abruptamente la cantidad de viajes posibles a analizar y dejar de manera implícita posibles restricciones de los equipos sobre los viajes a realizar, evitando luego rechazos de futuros *fixtures*.

Segunda etapa

Una vez obtenido el resultado de la primera etapa se pasa a la segunda etapa, que distribuirá los partidos a lo largo de los días. Una cosa a observar: las fechas (es decir, quién juega contra quién y en qué orden) ya están definidas y por lo tanto la distancia a recorrer por los equipos. Esto es, el óptimo que queríamos obtener, el problema original ya lo tenemos. En este paso eso no se modificará. Lo único que puede pasar a partir de ahora es que la solución que tenemos no sea factible. Es por esto que, esencialmente, el segundo problema es de factibilidad. Lo que se va a hacer es distribuir las fechas de la solución anterior a lo largo de los días del torneo, intentando satisfacer todas las restricciones impuestas sobre los días. Aquí entrarán en juego restricciones que no tenían sentido en el primer modelo, porque no existía el concepto de "día". Restricciones sobre días televisables, sobre días donde ciertos estadios están inhabilitados pueden fácilmente resultar en que las fechas designadas anteriormente no puedan ser ubicadas en ningún día.

Si esto ocurriera, será necesario repetir el proceso, comenzando por el primer modelo. Vamos entonces a escribir los modelos matemáticos:

3.2.1. Primer modelo

Conjuntos y parámetros

Sean, para un n dado (en nuestro caso será $n = 12$) el conjunto de equipos $\textbf{Equipos} = \{1, 2, ..., n\}$ y el conjunto de fechas $\textbf{Fechas} = \{1, 2, ..., 3(n-1)\}$.

Llamaremos a veces a la cantidad de elementos del conjunto de equipos como $|Equipos|$ y la cantidad de fechas como $|Fechas|$ con el fin de simplificar la lectura.

Sea, además, para cada $i \in Equipos$ el conjunto de viajes $\textbf{Viajes}_\textbf{i}$, donde cada viaje $v \in Viaje$ tiene una lista de equipos ordenada, sin repeticiones y sin contener al equipo i. Notaremos $|v|$ a la longitud de la lista, es decir, la cantidad de equipos que visita i realizando el viaje v.

Llamaremos a la unión de todos los viajes $\textbf{Viajes} = \bigcup_{i=1}^{n} Viajes_i$.

Recordemos que estos viajes van a ser los proporcionados por los equipos

Variables

Para cada $i, j \in Equipos$ con $i \neq j$ y para cada $k \in Fechas$, definimos las variables binarias x_{ijk}.

$$x_{ijk} = \begin{cases} 1 & si\ el\ equipo\ \boldsymbol{i}\ juega\ de\ local\ contra\ el\ equipo\ \boldsymbol{j}\ en\ la\ fecha\ \boldsymbol{k} \\ 0 & si\ no \end{cases}$$

Además, definamos para cada $i \in Equipos$, para cada $v \in Viajes_i$ y para cada $k \in Fechas$ las variables binarias y_{ivk}. Donde

$$y_{ivk} = \begin{cases} 1 & si\ el\ equipo\ \boldsymbol{i}\ comienza\ el\ viaje\ \boldsymbol{v}\ en\ la\ fecha\ \boldsymbol{k} \\ 0 & si\ no \end{cases}$$

Con estas dos familias de variables estamos en condiciones de definir las restricciones:

Restricciones

1. Cada partido se juega exactamente una vez de local y una vez de visitante.

$$\sum_{k \in Fechas} x_{ijk} = 1 \qquad \forall i, j \in Equipos,\ i \neq j$$

2. Cada equipo juega a lo sumo un partido por fecha.

$$\sum_{j \in Equipos,\, i \neq j} x_{ijk} \leq 1 \qquad \forall i \in Equipos,\ \forall k \in Fechas$$

3. Los partidos de visitante salen de los viajes disponibles.

$$x_{ijk} = \sum_{v \in Viajes_j} y_{jv,k-pos(v,i)+1} \qquad \forall i, j \in Equipos,\ i \neq j,\ \forall k \in Fechas$$

   Donde se entiende que la suma se hace sobre los viajes de **j** que tienen en su lista de destinos al equipo **i**.
   Además, pos(v,i) es una función que toma un viaje $\mathbf{v} \in Viajes$ y un equipo **i** perteneciente a la lista de destinos de **v** y devuelve su posición en dicha lista.

4. No puede comenzar un viaje que no tenga suficientes fechas para concluirlo.

$$y_{ivk} = 0 \qquad \forall i \in Equipos,\ \forall v \in Viajes,\ \forall k, k + |v| \geq |Fechas|$$

Agregamos también restricciones extra que servirán para la segunda etapa:

5. Cada equipo juega su primer partido dentro de las primeras 3 fechas.

$$\sum_{j \in Equipos, j \neq i} \sum_{k=1}^{3} x_{ijk} \geq 1 \qquad \forall i \in Equipos$$

6. Todos juegan su último partido en la última fecha (salvo uno, en caso de imparidad).

$$\sum_{i,j \in Equipos, j \neq i} x_{ij|Fechas|} \geq \frac{|Equipos| - 1}{2}$$

7. Ningún equipo pasa más de 3 fechas sin jugar.

$$\sum_{j \in Equipos, j \neq i} \sum_{l=0}^{3} x_{ij,k+l} + x_{ji,k+l} \geq 1 \qquad \forall i \in Equipos,\ \forall k \in Fechas,\ k+3 \leq |Fechas|$$

8. Ningún equipo pasa más de 5 fechas sin jugar de visitante al menos una vez.

$$\sum_{j \in Equipos, j \neq i} \sum_{l=0}^{6} x_{ji,k+l} \geq 1 \qquad \forall i \in Equipos,\ \forall k \in Fechas,\ k+6 \leq |Fechas|$$

9. Cada equipo descansa la fecha anterior o posterior a un viaje.

$$\sum_{j \in Equipos, j \neq i} x_{ij,k-1} + x_{ji,k-1} + x_{ij,k+|v|} + x_{ji,k+|v|} \leq 2 - y_{v,k}$$

$$\forall i \in Equipos,\ \forall v \in Viajes_i,\ \forall k \in Fechas,\ k \neq 1,\ k+|v| \leq |Fechas|$$

10. Cada equipo, después de hacer un viaje, juega al menos un partido de local en las 4 fechas subsiguientes.

$$\sum_{j \in Equipos, j \neq i} \sum_{l=0}^{3} x_{ij,k+|v|+l} \geq y_{vk}$$

$$\forall i \in Equipos,\ \forall v \in Viajes_i,\ \forall k \in Fechas,\ k+|v|+3 \leq |Fechas|$$

Función objetivo

Finalmente definimos la función objetivo, en este caso a minimizar, que son los kilómetros recorridos por todos los equipos durante todo el torneo.

$$f.o. = \sum_{i \in Equipos} \sum_{v \in Viajes} \sum_{k \in Fechas} y_{ivk}.kms(v)$$

Donde kms(v) es una función del conjunto de Viajes a $\mathbb{R}$ que devuelve la cantidad de kilómetros a recorrer cuando el equipo asociado a v visita la lista de destinos asociado a v y regresa.

Es importante notar que la función a optimizar mide la distancia global, es decir la suma de las distancias recorridas por todos los equipos.

Con este primer modelo vamos a obtener para cada equipo una lista de $2(|Equipos|-1)$ rivales, las localías de cada partido, y sublistas que indican cuáles partidos de visitante corresponden al mismo viaje (ya que podría haber partidos consecutivos de visitante que no estén en el mismo viaje). Con toda esta información pasamos al segundo modelo.

3.2.2. Segundo modelo

El segundo modelo tomará los resultados del primero como *input* para devolver luego la distribución de los días. Es decir, mientras que el primer modelo ya definió que el equipo **i** debe jugar contra el equipo **j** de visitante en su **k**-ésima fecha, el segundo modelo definirá exactamente en qué día se jugará.

Es importante notar que este segundo modelo podría ser solo un problema de factibilidad. Lo que realmente nos importaba optimizar, la distancia, ya está hecho y las decisiones tomadas de aquí en adelante no cambiarán al óptimo.

Conjuntos y parámetros

Usaremos el mismo conjunto de **Equipos** y, aunque usemos de nuevo el conjunto **Fechas**, será ahora $Fechas = 2(n-1)$. Esto es, representa las fechas de hecho de los equipos que juegan contra alguien, sin la relajación del problema anterior. Es decir, no habrá ahora fechas libres.

Algo muy importante a observar: Debido a cómo se resolvió el problema anterior, la fecha k-ésima de un equipo puede no corresponderse con la k-ésima de otro. Por ejemplo, puede ser que el equipo i juegue contra el equipo j de local en su tercera fecha mientras que el equipo j visite al i en su segunda fecha. Esto no es, en principio, un problema.

Para cada equipo $i \in Equipos$ sumaremos los conjuntos $\boldsymbol{InicioV_i} \subseteq Fechas$, las fechas donde el equipo i comienza un viaje y $\boldsymbol{FinV_i} \subseteq Fechas$, las fechas donde el equipo i finaliza un viaje.

Además, sea el conjunto de días $\boldsymbol{Días} = \{1, 2, ..., D\}$ para algún **D** deseado como duración del torneo. De nuevo, utilizaremos $|Días|$ para la cantidad de días, con el propósito de facilitar la lectura. Como subconjunto tendremos a $\boldsymbol{DíasT} \subseteq \boldsymbol{Días}$ los días televisados (en nuestro ejemplo los jueves) que son días en los que debe haber sí o sí algún partido para pasar por televisión.

Podría agregarse también un conjunto de enfrentamientos "relevantes" para la televisión. Puede ocurrir que se quiera los jueves televisar algún partido importante

y no *cualquier* partido. En nuestro caso supondremos que todos los partidos tienen la misma relevancia televisiva.

M será un parámetro que especificará la máxima cantidad de días que puede estar un equipo sin jugar un partido (8 en nuestro caso).

Variables

Vamos a definir unas nuevas variables binarias z_{ikt} $\forall i \in Equipos,\ \forall k \in Fechas,\ \forall t \in Días$. Donde

$$z_{ikt} = \begin{cases} 1 & si\ el\ equipo\ \mathbf{i}\ juega\ su\ fecha\ \mathbf{k}\ en\ el\ día\ \mathbf{t} \\ 0 & si\ no \end{cases}$$

Recordemos que, para un $i \in Equipos$ cuál es su fecha k, contra quién juega, quién es local y si está de viaje o no son cosas que vienen fijadas por la solución del problema anterior.

Restricciones

1. A cada partido se le asigna un día

$$\sum_{t \in Días} z_{ikt} = 1 \qquad \forall i \in Equipos,\ \forall k \in Fechas$$

2. Para cada equipo, no puede haber más de T partidos sin jugar entre fecha y fecha

$$\sum_{t=t_0}^{t_0+T} z_{i,k+1,t} \geq z_{i,k,t_0} \qquad \forall i \in Equipos,\ \forall k \in Fechas \setminus \{|Fechas|\},\ \forall t_0 \in Días$$

3. Si el equipo está en viaje, juega exactamente día por medio

$$z_{i,k,t} = z_{i,k+1,t+2} \qquad \forall i \in Equipos,\ \forall k \in Fechas,\ \forall t \in Días$$

   Siempre y cuando el equipo i esté en el mismo viaje en las fechas k y $k+1$

4. Cada equipo descansa al menos dos días antes y después de cada viaje

$$z_{i,k-1,t-1} + z_{i,k-1,t-2} + z_{i,k,t} \leq 1$$
$$\forall i \in Equipos,\ \forall k \in InicioV_i \setminus \{1\},\ \forall t \in Días \setminus \{1, 2\}$$

$$z_{i,k+1,t+1} + z_{i,k+1,t+2} + z_{i,k,t} \leq 1$$
$$\forall i \in Equipos,\ \forall k \in FinV_i \setminus \{|Fechas|\},\ \forall t \in Días \setminus \{D-2, D-1\}$$

5. Los partidos en contra se corresponden unos con otros

$$z_{ik_{ij}^l t} = z_{jk_{ji}^l t} \qquad \forall i, j \in Equipos, i \neq j,\ l \in \{1, 2\},\ \forall t \in Días$$

Siendo k_{ij}^l la fecha del equipo i en la que se enfrenta con el equipo j. Si es la primera o la segunda vez que se enfrentan los equipos lo define el valor de l

6. Debe haber al menos un partido en los días televisados

$$\sum_{i \in Equipos} \sum_{k \in Fechas} z_{ikt} \geq 1 \qquad \forall t \in DíasT$$

7. Todos los equipos juegan su última fecha el mismo día

$$z_{i,|Fechas|,t} = z_{j,|Fechas|,t} \qquad \forall i, j \in Equipos,\ i \neq j,\ \forall t \in Días$$

Función objetivo

En este caso, por lo ya mencionado, la función objetivo no es en realidad muy relevante. Podríamos poner una función objetivo constante y no tendríamos mayores problemas. Básicamente se volvería un problema de factibilidad.

Pero, ya que en la realidad las soluciones no son de hecho equivalentes agregamos una dimensión a explorar en el objetivo. Los equipos sumarán como información en qué días prefieren jugar, lo cual puede pensarse también como información de en qué días prefieren **no** jugar. Incluso podrían informar con toda una escala de preferencia para cada día y pesarlos con esos valores en la función objetivo.

Volviendo a este caso: el conjunto de días preferidos por el equipo i será DP_i y por lo tanto la función objetivo nos quedará:

$$f.o. = \sum_{i \in Equipos} \sum_{k \in Fechas} \sum_{t \in DP_i} z_{ikt}$$

La cual trataremos de maximizar.

Notar que cada día preferido que se elija (su variable asociada z_ikt valga 1) sumará 1 a la función objetivo.

Esto recae en dos puntos:

- Un día preferido por un equipo que pidió 30 días vale lo mismo que uno de un equipo que pidió un día.

  Esto puede solucionarse fácilmente pesando distinto según la cantidad de días preferidos por equipo o fijando la cantidad de días preferidos que puede pedir cada equipo.

- Aun cuando esté discriminado, al *solver* le dará lo mismo asignar 20 días preferidos a un equipo y 0 a otro o 10 y 10.

  Resolver esto no es tan sencillo porque hay que incurrir en funciones objetivo cuadráticas. Pero de nuevo recordemos que la función objetivo en este paso no es lo que más nos interesa del problema.

Complicaciones

Al hacer las primeras corridas de la segunda etapa se encontraron varios problemas con las soluciones halladas. En particular dos que pasaban muy a menudo:

- el campeonato que ofrecía la solución era demasiado corto
- el torneo con las restricciones pedidas era infactible (problema que surge mayormente con la solución del primero, como veremos).

El primer problema surge un poco naturalmente de las restricciones impuestas por el problema. Por más que la restricción 4 agrega descansos antes y después de los viajes, las restricciones 2 y 3 piden respectivamente que ningún equipo esté más de T partidos entre fecha y fecha y que jueguen día por medio cuando están de viaje. Esto obliga (más cuánto mejor es la solución de la primera etapa) a que los partidos se jueguen muy próximos entre sí. Por eso es que habrá una tendencia natural hacia devolver torneos de muy corta duración. O al menos cortos comparados con ediciones anteriores.

Por suerte la solución para esto fue relativamente sencilla.

Lo que se hizo fue crear unas variables binarias p_t y u_t que representaran, respectivamente, el primer y último día del campeonato.

$$p_t = \begin{cases} 1 & \textit{si el campeonato comienza el día } \mathbf{t} \\ 0 & \textit{si no} \end{cases}$$

$$u_t = \begin{cases} 1 & \textit{si el campeonato finaliza el día } \mathbf{t} \\ 0 & \textit{si no} \end{cases}$$

Luego se crean cuatro familias de restricciones extra:

- Una que obliga al campeonato a durar más de M días:

$$\sum_{t=d}^{d+M-1} p_t + u_t \leq 1 \qquad \forall d \in \{1, \ldots, D - M\}$$

  Es decir, en cada ventana de M días no pueden estar el fin y el inicio del torneo.

- Otra que me asegure unicidad del primer y último día del campeonato

$$\sum_{t \in Días} p_t = 1$$

$$\sum_{t \in Días} u_t = 1$$

- Otra que liga las variables p para que valgan lo que queremos:

$$\sum_{t \in Días_{\leq d}} \left(n.p_t - \sum_{i \in Equipos} z_{i,1,t} \right) \geq 0 \qquad \forall d \in Días$$

Con esto pedimos que, dado un día d si algún equipo jugó su primer partido antes de d, entonces p_t debe valer 1 para algún $t \leq d$.

Esto es, si llamamos t_0 al único día para el cual $p_{t_0} = 1$ se tiene

$$t_0 \leq \min_i \{t \in Días / z_{i,1,t} = 1\}$$

Y además

$$p_t \leq \sum_{i \in Equipos} z_{i,1,t} \qquad \forall t \in Días$$

Con esto obligamos a que p_t solo valga 1 si ese día algún equipo juega su primera fecha.

Esto nos deja entonces que

$$t_0 \geq \min_i \{t \in Días / z_{i,1,t} = 1\}$$

Con lo cual el mínimo, o sea el primer partido jugado, se alcanza en t_0, el día donde la variable de primer día vale 1.

- Y por último, otra que liga las variables u:

$$u_t = z_{i,|Fechas|,t} \qquad \forall t \in Días\,,\, \forall i \in Equipos$$

Esta es mucho más sencilla. Al jugar todos los equipos la última fecha el último día, el u_t que queremos que valga 1 es el del t donde cualquier equipo juegue el último partido. O lo que es equivalente, todos los equipos.

Observar que este conjunto de restricciones deja redundante la que obligaba que el último partido se jugara en simultáneo.

Por otro lado, si tuviéramos n impar, habrá que restringir está restricción solo a los $n-1$ equipos que jueguen su última fecha el último día. El equipo que tenga fecha libre habrá jugado necesariamente su última fecha un día anterior a este, por lo que u_t seguirá estando bien ligada.

Una cosa a observar es que es una restricción dura. Si el *solver* no encuentra un campeonato de más de M días directamente devolverá que el problema es infactible.

Podría buscarse una versión alternativa donde la función objetivo se ocupe de maximizar la cantidad de días del campeonato. Sin embargo, el *solver* resuelve muy rápidamente esta segunda etapa y las variaciones posibles para M son realmente muy pocas. Basta con correr con algunos valores para encontrar el límite de factibilidad.

El segundo problema es justamente el de la factibilidad. Forzando al campeonato a durar más de M días, o incluso sin hacerlo, ¿qué es lo que puede volverlo infactible? ¿es el problema intrínsecamente imposible de resolver o hemos sido demasiado severos a la hora de restringirlo?

Al observar de nuevo las restricciones en estos casos, nos podemos dar cuenta que hay restricciones que no son tan primordiales como otras. Si i enfrenta a j el día t, es forzoso que j enfrente a i el mismo día. Ahora, si un equipo no logra descansar dos días después de un viaje...quizás no sea tan grave.

Al ver que una de las principales causas de infactibilidad era el descanso post y pre viaje se optó por volver esta una restricción *soft*. Es decir, una restricción deseable pero no imprescindible. Para esto se agregan las variables binarias de holgura (pueden ser reales entre 0 y 1 para no forzar al *solver*) s^{pre} y s^{post} para todo i, k y t.

Con ellas, las ecuaciones del punto 4 quedarán:

$$z_{i,k-1,t-1} + z_{i,k-1,t-2} + z_{i,k,t} \leq 1 + s^{pre}_{ikt}$$
$$\forall i \in Equipos,\ \forall k \in InicioV_i \setminus \{1\},\ \forall t \in Días \setminus \{1,2\}$$

$$z_{i,k+1,t+1} + z_{i,k+1,t+2} + z_{i,k,t} \leq 1 + s^{post}_{ikt}$$
$$\forall i \in Equipos,\ \forall k \in FinV_i \setminus \{|Fechas|\},\ \forall t \in Días \setminus \{D-2, D-1\}$$

Al ser el lado izquierdo siempre menor o igual a 2, si las variables de holgura valen 1 permiten que pase cualquier cosa.

Como queremos de todas maneras evitar esto lo más posible agregamos un peso grande para estas variables en la función objetivo.

La nueva función objetivo quedará entonces:

$$f.o. = \sum_{i \in Equipos} \sum_{k \in Fechas} \sum_{t \in DP_i} \left(z_{ikt} + M_1 . s^{pre}_{ikt} + M_2 . s^{post}_{ikt} \right)$$

Para unos M_1 y M_2 elegidos para representar la gravedad de violar esta restricción. Por supuesto podrían pesarse distinto, si se quisiera para, equipos, fechas o días distintos.

Decíamos que si estas variables se encienden permiten cualquier cosa. Pero incluso permiten que se juegue un partido inmediatamente después o inmediatamente antes del viaje. Esto puede ser indeseado. Si queremos permitir un solo día de descanso pero no **cero** podemos agregar las restricciones

$$z_{i,k-1,t-1} + z_{i,k,t} \leq 1$$

$$z_{i,k+1,t+1} + z_{i,k,t} \leq 1$$

Para los i, k, t de antes y asegurarnos que esto no suceda.

Agregando esta flexibilidad a las restricciones se lograron muy buenos resultados. Los problemas que eran infactibles para la distribución de días pasaron a ser factibles y solo sacrificando un (y no los dos) día de descanso. Además, al estar las variables de holgura fuertemente pesadas, solamente se quebraba el descanso lo mínimo indispensable, una o dos veces por campeonato para los equipos en conjunto.

Capítulo 4

Resultados

4.1. Liga 2018/2019 (10 equipos)

Lo primero que se hizo fue trabajar con un campeonato lo más estándar posible: 10 equipos, *double round robin*, la cantidad es par, por lo que se trata casi de un **TTP** perfecto. Digo "casi" porque hay varios puntos que nos alejan de un **TTP** clásico: los viajes posibles no se definen por su largo sino por extensión, hay más fechas de las necesarias, hay descansos, hay restricciones de *breaks*, etc. Sin embargo, no tenemos que preocuparnos por ejemplo de las asimetrías que traen los otros torneos estudiados.

Un paso importante para el primer modelo es el de la generación de viajes. En lugar de crear todos los viajes posibles a realizar por un equipo (con límite inferior y superior en la cantidad de destinos antes de regresar), los viajes van a venir como input al *solver*, de parte de los mismos equipos. Esto sirve para dos cosas:

- Ayudar al *solver*, preprocesando y filtrando viajes que posiblemente no estén en la solución final.
- Evitando el rechazo del *fixture*, donde aparezcan viajes que los equipos no quieran hacer o directamente no harán.

Por ejemplo, si **River** (Bs As) tuviera que hacer un viaje visitando a **Gigantes del sur** (Neuquén) y luego **Monteros** (Tucumán), es posible que decidan pasar por casa en el medio, rompiendo el propósito del viaje de ahorrar kilómetros (ver figura 4.1)

Además de esta forma quedan contemplados casos imposibles de ver para alguien externo, donde pueden entrar en juego contratos con agencias o incluso cuestiones personales de los jugadores.

Más adelante veremos además algunos casos donde **todos** los viajes (con límite de cantidad de visitas) son tenidos en cuenta, para poder usar como comparación.

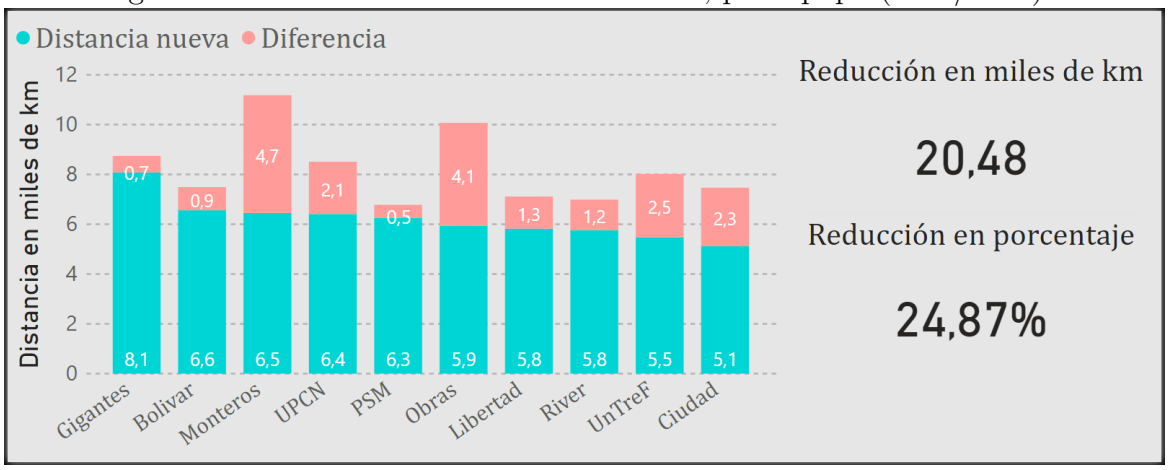
Figura 4.1: Los equipos de la liga de Vóley de la temporada 2018-19

Ya que no contamos con los datos reales de los equipos, los viajes posibles van a ser generados de forma automática, teniendo en cuenta viajes que tienen sentido al mirar el mapa y sobre todo los viajes que se dieron en el *fixture* real, asegurándonos que nuestra solución sea por lo menos tan buena como la original.

Para esto hubo que obtener y procesar los partidos asignados en la temporada y su orden, lo cual fue hecho en su mayor parte a través de la página de la Asociación de Clubes Liga Argentina de Voleibol, www.aclav.com. Además, esto nos permite calcular las distancias reales recorridas en la temporada, para poder compararlas con nuestros resultados.

Una vez corrido el primer modelo, se obtienen los siguientes resultados:

Figura 4.2: Cambio en la distancia recorrida, por equipo (2018/2019)



Se logra una mejora de aproximadamente **24.87 %** total, reduciendo en más de 20 mil kilómetros la distancia recorrida. Algo muy positivo a observar es que todos los equipos reducen los kilómetros viajados. Esto podría tranquilamente no suceder; el objetivo es global, y a priori nos da igual que un equipo reduzca 10 mil kilómetros o que cada uno reduzca mil. De hecho, veremos en unos casos siguientes que pasa algo del estilo. Se podría buscar un resultado similar para todos los equipos, pero no es el objetivo. Además de que convertiría al problema en un problema cuadrático.

Los equipos que más redujeron los km respecto del *fixture* original son **Monteros** y **Obras**, con más de un **40 %** de mejora cada uno. Podemos ver que **Monteros** viajó muchísimo esta temporada, posiblemente por estar muy alejado del resto de los equipos.

Quizás surja la duda, ¿por qué **Obras** y **UPCN** no viajaron lo mismo (o muy parecido) originalmente, si el *fixture* se hizo con un sistema de parejas? En el torneo publicado por *ACLAV* se pueden ver irregularidades, fechas cambiadas, posiblemente por pedidos o problemas de los equipos.

En esto gana el nuevo *fixture*, que puede tener en cuenta estos cambios **individuales**, sin tener que afectar la pareja entera y aún buscando minimizar las distancias.

La solución del primer modelo (o al menos **una** solución, siempre se puede iterar si el resultado no es deseable) puede verse en la tabla 4.1. En amarillo se destacan los viajes realizados por cada equipo.
En términos de dinero, si aproximamos el costo de ómnibus como 150 $/km (algo así como 2 **USD**/km) tenemos un ahorro total de más de 3 millones de pesos. Este es un gran ahorro dada la cantidad de equipos. Este dinero, en lugar de ser ahorrado, podría ser usado por ejemplo para que todos los viajes pasen a ser en avión en lugar de en ómnibus (o al menos los que puedan hacerse en avión).

Fecha	CIU	GIG	LIB	MON	OBR	BOL	PSM	RIV	U3F	UPC
0				U3F				@UPC	@MON	RIV
1					RIV			@OBR	@UPC	U3F
2	@PSM	@BOL	UPC	RIV	U3F	GIG	CIU	@MON	@OBR	@LIB
3	@LIB		CIU		@UPC	@RIV		BOL		OBR
4	@GIG	CIU	@BOL	@RIV	PSM	LIB	@OBR	MON		
5	@BOL	LIB	@GIG	@U3F		CIU	@UPC		MON	PSM
6	MON			@CIU		UPC				@BOL
7	UPC		MON	@LIB				U3F	@RIV	@CIU
8	GIG	@CIU	RIV		@PSM		OBR	@LIB	UPC	@U3F
9		@U3F	OBR		@LIB	@PSM	BOL	UPC	GIG	@RIV
10	U3F	@RIV	BOL	OBR	@MON	@LIB	UPC	GIG	@CIU	@PSM
11	@U3F	PSM	@RIV	BOL		@MON	@GIG	LIB	CIU	
12			@U3F	@OBR	MON	PSM	@BOL		LIB	
13	LIB		@CIU	@UPC		RIV		@BOL		MON
14		RIV	@PSM		@BOL	OBR	LIB	@GIG		
15		@MON		GIG	@U3F		RIV	@PSM	OBR	
16	OBR	@LIB	GIG	UPC	@CIU		U3F		@PSM	@MON
17	@MON	@PSM	U3F	CIU	@RIV	@UPC	GIG	OBR	@LIB	BOL
18	@UPC				BOL	@OBR	@RIV	PSM		CIU
19	@OBR	MON	@UPC	@GIG	CIU		@U3F		PSM	LIB
20		U3F	@OBR	@BOL	LIB	MON			@GIG	
21	RIV	UPC		@PSM		U3F	MON	@CIU	@BOL	@GIG
22		OBR			@GIG					
23		@OBR			GIG					
24	@RIV	@UPC	PSM			@U3F	@LIB	CIU	BOL	GIG
25	BOL			PSM		@CIU	@MON			
26	PSM	BOL	@MON	LIB	UPC	@GIG	@CIU	@U3F	RIV	@OBR

Cuadro 4.1: Fase 1 de la resolución de la liga 2018/2019

Tomemos como ejemplo el caso de **UnTreF** y comparemos los gastos a día de hoy. Según el *fixture* dado por el modelo anterior, el equipo debía recorrer un total de 8.550 km por ruta, que con nuestra cuenta anterior nos da un gasto aproximado de **$1.282.500** en ómnibus.

Con el nuevo *fixture* tenemos tres viajes múltiples:

- **Monteros** (Túcuman)-**UPCN-Obras** (San Juan),
- **PSM-Libertad** (Santa Fe),
- **Gigantes** (Neuquén)-**Bolivar** (Buenos Aires).

Tomando valores de www.aerolineas.com.ar a la fecha, tenemos que el vuelo BSAS-San Miguel de Tucumán con vuelta San Juan-BSAS cuesta unos $17.000 por persona, el tramo BSAS-Rosario unos $4.500 y BSAS-Neuquén unos $7.000.

Si calculamos que viajan 20 personas (jugadores titulares y suplentes más equipo técnico) y agregamos todos los tramos restantes en ómnibus (Desde Tucumán a San Juan por ejemplo) obtenemos un costo total de unos **$1.002.000**. Es decir, aún unos **$280.000** por debajo del costo del *fixture* anterior.

Una vez corrido el primer modelo, falta distribuir los partidos en los días utilizables, lo cual será definido por el segundo modelo. Algo ya dicho pero que vale la pena recordar: una vez resuelto el primer modelo, la parte de optimización ya está resuelta. Los viajes, los kilómetros que obtuvimos, la mejora respecto del torneo pasado ya está fija. Lo único que queda ver es la factibilidad de la solución. ¿Es posible distribuir los partidos en los días que tenemos? ¿es posible que los equipos realicen esos viajes en ese orden?

En principio el segundo modelo solo va a mirar factibilidad, aunque para ayudar a la solución, vamos a permitir algunas cosas indeseadas, penalizándolas luego en la función objetivo.

Por ejemplo: las restricciones de dos días de descanso antes y después de un viaje son importantes, pero no tiene sentido desestimar un *fixture* entero si un solo equipo descansa un día en lugar de dos después de algún viaje. Podemos agregar variables de holgura en estos casos con mucho peso en la función objetivo. Incluso podemos pesarlas de forma discriminada según equipo y según viaje (teniendo en cuenta longitud, por ejemplo).

En la Figura 4.1 podemos ver los partidos distribuidos en los primeros días de enero, **Libertad** haciendo un viaje a Buenos Aires y de vuelta por Santa Fe, **Monteros** en un viaje a visitar a los dos equipos de San Juan y **River** en viaje hacia el sur. Como habíamos especificado, los partidos que se juegan en viaje se juegan exactamente cada dos días. Observar que en estos casos sí se respetan los dos días de descanso antes y después de cada partido. Mientra tanto, **Ciudad** y **Bolivar** juegan la copa Libertadores intercalada entre el torneo, con un día de descanso antes y después de la copa.

	2019-01-06	2019-01-07	2019-01-08	2019-01-09	2019-01-10	2019-01-11	2019-01-12	2019-01-13
Ciudad	@UnTreF		Libertadores	Libertadores		Libertad		
Gigantes								River
Libertad		@River		@UnTreF		@Ciudad		@PSM
Monteros			@Obras		@UPCN			
Obras			Monteros					
Bolivar			Libertadores	Libertadores		River		
PSM								Libertad
River		Libertad				@Bolivar		@Gigantes
UnTreF	Ciudad			Libertad				
UPCN					Monteros			

Figura 4.3: Fragmento de la segunda fase de resolución para la temporada 2018/2019

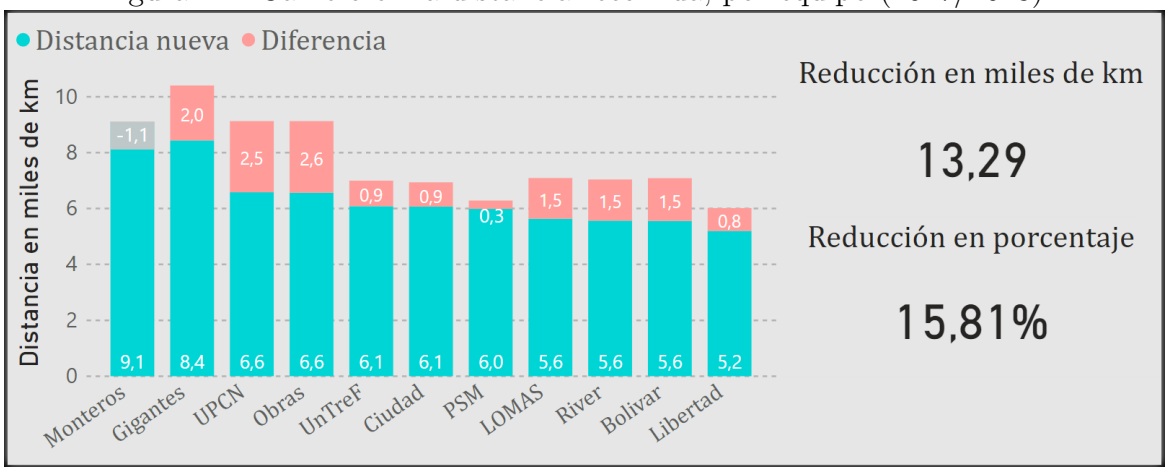
4.2. Liga 2017/2018 (11 equipos)

Para la temporada 2017/2018 se pierde la simetría que teníamos en el campeonato anterior. Esto se va a ver reflejado en algunas restricciones, aunque pocas. La primera es que en lugar de pedir que todos los equipos jueguen su último partido en la última fecha vamos a pedir que lo hagan 10 equipos, dejando un grado de libertad para decidir qué equipo tiene libre la última fecha ampliada.

En otro aspecto en el que cambia el problema es en el más obvio: hay ahora muchas más variables para analizar. En donde más se replica esto es en la cantidad de viajes generados. Todos los viajes que ya teníamos deberían ver la opción de agregar el onceavo equipo a cualquiera de sus posibles viajes, en cualquiera de todas las combinaciones posibles. Además de, claro está, agregar todos los viajes posibles del nuevo equipo.

Esto hará que los tiempos de cómputo se alarguen. Para llegar a este caso específico se corrió durante 6 días llegando a un **12 %** de gap.

Figura 4.4: Cambio en la distancia recorrida, por equipo (2017/2018)



Aquí se ve que los resultados no son tan buenos como en el caso anterior, llegando incluso **Monteros** a empeorar su situación respecto al campeonato original. Podríamos pedir como restricción que todos los equipos mejoren la cantidad de kilómetros viajados respecto al anterior pero no tendría mucho sentido. Después de todo, estos casos solo sirven como comparación y en la creación de un nuevo campeonato no va a estar disponible el *fixture* alternativo.

De todas formas la reducción sigue siendo buena, de más del **15 %**, siendo **Obras** el equipo con mayor reducción porcentual, con casi un **30 %**.

Respecto al caso nombrado en el anterior ejemplo, aquí sí **UPCN** y **Obras** tienen recorridos casi gemelos, tanto en el *fixture* nuevo como en el original.

El resultado de la fase 1 para este campeonato se puede ver en el cuadro 4.2. De nuevo, los viajes están resaltados en amarillo.

Podemos observar cómo se cumple la restricción de equidad deportiva de que todos los equipos jueguen el último partido a la vez. Por supuesto, al ser impares, hay un equipo que jugará su último partido antes. En este caso sería **Gigantes del sur**. Algo interesante a notar es que esto lo decide el *solver*, cuando antes ya venía predifinido por las parejas. El equipo sin pareja, o con una pareja virtual, tendría fecha libre el último día.

Fechas	CIU	GIG	LIB	MON	OBR	BOL	PSM	RIV	U3F	UPC	LOM
0	@PSM	UPC					CIU			@GIG	
1	@LIB	RIV	CIU		@PSM	U3F	OBR	@GIG	@BOL		
2		U3F	OBR	@UPC	@LIB	RIV	LOM	@BOL	@GIG	MON	@PSM
3	@U3F		LOM		BOL	@OBR			CIU		@LIB
4		@MON	GIG		PSM	@UPC	@OBR			BOL	
5	LIB		@CIU	BOL		@MON	@UPC		LOM	PSM	@U3F
6			@RIV		UPC			LIB		@OBR	
7		MON	@LOM	@GIG	RIV		U3F	@OBR	@PSM		LIB
8	@GIG	CIU	@U3F	@PSM	@BOL	OBR	MON	@UPC	LIB	RIV	
9			PSM	RIV	@LOM		@LIB	@MON			OBR
10	GIG	@CIU			@U3F				OBR		
11	LOM	@U3F	UPC		@RIV	@PSM	BOL	OBR	GIG	@LIB	@CIU
12	OBR	@RIV	BOL	U3F	@CIU	@LIB	UPC	GIG	@MON	@PSM	
13	RIV	@LOM		@BOL		MON		@CIU	@UPC	U3F	GIG
14	@RIV	@BOL	@UPC	@LOM	U3F	GIG		CIU	@OBR	LIB	MON
15	MON	BOL	@OBR	@CIU	LIB	@GIG	@LOM				PSM
16		LIB	@GIG	@U3F		UPC	@RIV	PSM	MON	@BOL	
17	UPC	LOM	@BOL	@RIV		LIB	@U3F	MON	PSM	@CIU	@GIG
18	PSM	OBR			@GIG	LOM	@CIU	UPC		@RIV	@BOL
19	U3F	@LIB	GIG				RIV	@PSM	@CIU	@LOM	UPC
20	@MON	@PSM	RIV	CIU			GIG	@LIB	UPC	@U3F	
21	@OBR		@MON	LIB	CIU	@LOM					BOL
22	@UPC		MON	@LIB		@RIV		BOL		CIU	
23				LOM		@U3F			BOL		@MON
24	BOL	PSM		OBR	@MON	@CIU	@GIG	@U3F	RIV	LOM	@UPC
25				UPC	LOM	PSM	@BOL	U3F	@RIV	@MON	@OBR
26		@OBR	@PSM		GIG		LIB		@LOM		U3F
27	@LOM	@UPC		@OBR	MON					GIG	CIU
28								LOM			@RIV
29	@BOL		U3F	PSM	@UPC	CIU	@MON	@LOM	@LIB	OBR	RIV

Cuadro 4.2: Fase 1 de la resolución de la liga 2017/2018

4.3. Liga 2019/2020 (9 equipos)

Para esta temporada (que se vio finalmente interrumpida por la aparición del *covid19*) se contaba con 9 equipos para armar el torneo, un mínimo nunca antes alcanzado para ACLAV. La no paridad no es en sí un gran problema, ya muchos años se habían manejado torneos de 11 equipos y las mismas técnicas se pueden aplicar aquí sin problemas.

El problema más grande con el que se encontraron fue otro: el torneo era demasiado corto. Comparado con el torneo del año anterior, habrían en este campeonato **18** partidos menos en total. Y si nos fuéramos 2 años atrás, esa cantidad aumentaría a **38**.

Pero estos números están fijos, están perfectamente definidos por el *double round robin*. Dado un n la cantidad total de partidos a jugar es $n.(n-1)$, mientras se

mantenga ese sistema, el número no se puede cambiar. Lo que se decidió hacer entonces fue agregar al *double round robin* 18 partidos más, 4 por equipo. Así se rompe la estructura del torneo, se vuelve no balanceado respecto a la cantidad de encuentros entre equipos, pero se logra la misma cantidad de partidos que el torneo anterior.

¿Cómo afecta esto al *solver*?

Por suerte, hay pocas cosas que deben ser cambiadas. En realidad, el momento en que pedimos que el torneo sea un *double round robin* es en la restricción de que cada partido (con localía) se juegue una y solo una vez:

$$\sum_{k \in Fechas} x_{ijk} = 1 \qquad \forall i, j \in Equipos,\ i \neq j$$

Esto podría cambiarse muy fácilmente, pidiendo que el encuentro entre los equipos i y j, con i de local se repita C_{ij} veces, donde C_{ij}, con $i, j \in Equipos\,,\ i \neq j$ es una constante definida para todo par ordenado de equipos distintos.

Quedaría entonces:

$$\sum_{k \in Fechas} x_{ijk} = C_{ij} \qquad \forall i, j \in Equipos,\ i \neq j$$

Siempre podríamos poner $C_{ij} = 1\ \forall i, j \in Equipos,\ i \neq j$ y volver al caso de *double round robin*.

Otra opción posible sería dejar que el *solver* elija qué partidos repetir. Esto se puede hacer convirtiendo al lado derecho de la ecuación en variable en lugar de constante. Una posible solución sería la siguiente:

$$\sum_{k \in Fechas} x_{ijk} = 1 + c_{ij} \qquad \forall i, j \in Equipos,\ i \neq j$$

con

$$c_{ij} = \begin{cases} 1 & \textit{si se repite el partido } \mathbf{i} \textit{ contra } \mathbf{j} \textit{ con } \mathbf{i} \textit{ de local} \\ 0 & \textit{si no} \end{cases}$$

Definido para cada i, para cada j con $i \neq j$. (Observar que esto evita que se repita un partido más de una vez).

Después habría que agregar las restricciones extras que decidan cuántos partidos repetir. Una posible solución sería

$$\sum_{j \in Equipos\,,\, i \neq j} c_{ij} = 2 \qquad \forall i \in Equipos$$

Esto es, que cada equipo repita 2 partidos de local. Esto implícitamente hará que todos los equipos jueguen 4 partidos extra, aunque podría ser un restricción demasiado fuerte.

Otro acercamiento podría ser el siguiente:

$$\sum_{j \in Equipos\,,\, i \neq j} c_{ij} + c_{ji} = 4 \qquad \forall i \in Equipos$$

Esto significa precisamente que cada equipo repite 4 partidos en total, y no nos importa la localía. Esto es, un equipo podría repetir sus 4 partidos como local. Recordemos que existen ya restricciones limitando la cantidad consecutivas de local que puede jugar un equipo, y los viajes presentados por cada equipo contienen implícitamente la información de cuántos partidos consecutivos de visitante están dispuestos a jugar, por lo que esto no representa en sí un problema.

Hablando de los viajes, ¿es necesario cambiar algo? ¿los viajes contemplan que se repitan partidos? ¿se permite o no que se repita consecutivamente un partido?

La respuesta es que, de nuevo, los viajes ofrecidos por los equipos van a contener toda esa información. Si no quieren repetir un partido en un viaje pueden no agregar viajes con destinos repetidos. Si permiten repetir pero no inmediatamente después, también pueden hacerlo. Toda esa información vendrá del lado de los viajes y no es necesario agregarla explícitamente al *solver*.

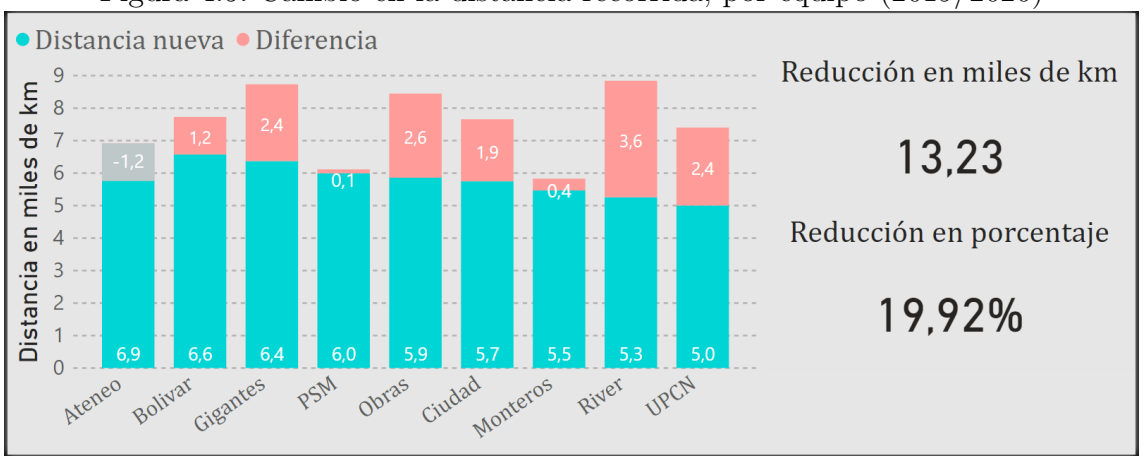
Esto aplica en ambos casos, en el de C fija y c variable.

¿Con cuál nos quedamos? Aunque es obvio que la versión variable siempre va a dar una mejor solución que la fija, por una cuestión deportiva quizás sea mejor fijar la cantidad. Esto puede hacerse mediante un sorteo, y librar la decisión a la suerte. Hay que tener en cuenta que los encuentros entre equipos van a tener asimetrías y esto puede beneficiar o perjudicar a los equipos en el torneo.

Además, por una cuestión de comparación de resultados, creí que era más justo comparar contra los partidos que efectivamente se jugaron en lugar de poder decidir sobre ellos.

Por lo que, volviendo al caso de C fijo, bastará con procesar la información del campeonato pasado y fijar los valores que correspondan.

Figura 4.5: Cambio en la distancia recorrida, por equipo (2019/2020)



Otra vez observamos que hubo un equipo (**Ateneo**) que empeoró su situación respecto del *fixture* original. De todas maneras, aquí volvimos a tener una alta reducción global, de casi **20 %**.

Aquí la mayor reducción fue para **River**, que sorprendentemente era el equipo que más viajaba. En este nuevo esquema, reduce en más de un **40 %** la cantidad de kilómetros viajados.

En el cuadro 4.3 se ve la solución de la primera fase del problema. **PSM** será el equipo que tendrá la "desventaja" de jugar su último partido antes que el resto.

Fecha	CIU	GIG	MON	OBR	BOL	PSM	RIV	UPC	ATE
0				@BOL	OBR	@ATE			PSM
1	OBR	UPC	PSM	@CIU		@MON		@GIG	
2	@BOL	ATE		@RIV	CIU		OBR		@GIG
3				@PSM	ATE	OBR			@BOL
4		@OBR		GIG		ATE			@PSM
5	RIV	@UPC	@ATE	BOL	@OBR		@CIU	GIG	MON
6	@RIV	@ATE			@UPC		CIU	BOL	GIG
7		@MON	GIG					@ATE	UPC
8			UPC			@RIV	PSM	@MON	
9	PSM	OBR		@GIG		@CIU			
10					PSM	@BOL		ATE	@UPC
11	BOL	MON	@GIG		@CIU		@UPC	RIV	
12	@PSM		@UPC	RIV		CIU	@OBR	MON	
13		@PSM	@OBR	MON		GIG	@ATE		RIV
14	GIG	@CIU	RIV		UPC		@MON	@BOL	
15	UPC	@RIV	OBR	@MON			GIG	@CIU	
16		PSM	BOL	ATE	@MON	@GIG	UPC	@RIV	@OBR
17	@MON		CIU		@ATE	UPC		@PSM	BOL
18	@ATE	RIV			@PSM	BOL	@GIG		CIU
19	@OBR		@PSM	CIU	RIV	MON	@BOL		
20	@UPC		@RIV	@ATE			MON	CIU	OBR
21	MON	@BOL	@CIU	PSM	GIG	@OBR			
22	ATE		@BOL		MON	@UPC		PSM	@CIU
23		BOL		UPC	@GIG		ATE	@OBR	@RIV
24						RIV	@PSM		
25									
26	@GIG	CIU	ATE	@UPC	@RIV		BOL	OBR	@MON

Cuadro 4.3: Fase 1 de la resolución de la liga 2019/2020

4.4. Análisis de tiempos de corrida

Para que todos los casos de prueba fueran comparables entre sí se corrieron en una máquina virtual con sistema operativo *Linux*, con 8 *vCPUs* y 64 *GB* de RAM.

Los algoritmos se corrieron en *Python* versión 3.7.3, llamando internamente a *CPLEX*, versión 12.10. El código desarrollado para resolver el problema puede encontrarse en el repositorio de GitHub `www.github.com/joacodp/voley`.

Para cada uno se puso como límite de tiempo 72 horas, con el gap predeterminado (10^{-6}). La corrida para la temporada 2017/2018, por ejemplo, había sido con límite de 168 horas (una semana) debido al tamaño del problema. Pero para que la comparación sea justa aquí se corrió con 72 horas, igual que el resto. En el cuadro 4.4 podemos analizar y comparar los resultados de las corridas para las tres temporadas, sumado a unas corridas de referencia.

En estas nuevas tenemos como viajes posibles a realizar cualquier viaje de longitud menor o igual a n (con n igual a 3 y 4), lo que en lenguaje de **TTP** dimos a llamar U.

Esto quiere decir que un equipo de Buenos Aires tendrá en esos casos la opción de jugar de visitante en Catamarca, luego en Neuquén y por último en Tucumán. Aún cuando el *solver* podría optar este viaje para de algun modo bajar la cantidad **global** de kilómetros recorridos, parece altamente improbable que suceda. De todas formas, poner el límite en qué viajes parecen probables de ser elegidos y qué otros no es una tarea poco práctica para llevar a cabo. Lo mejor será dar todas las opciones posibles al *solver* para que decida.

Pero, ¿en qué perdemos al agregar opciones que, ciertamente, pueden potencialmente mejorar la solución? Se pierde drásticamente en tiempos de corrida, o calidad de solución, depende cuál de las dos fijemos. En este caso, al estar todas las corridas limitadas en el tiempo, se ve que quedamos en un gap más alto y por consiguiente peores mejoras respecto al resultado del campeonato real, en algunos casos empeorando incluso.

Los resultados nos muestran que elegir bien qué viajes dar como *input* al *solver* es crucial. A mismo tiempo de corrida pudimos obtener mejoras considerables en solución. Y aunque el tiempo no parece ser una variable limitante para este problema (el algoritmo se correría una vez por año para cada campeonato), no hay seguridad de cuánto le podría tomar a algunas corridas llegar a resultados satisfactorios.

El caso de 2018/2019 con viajes de longitud hasta 4, por ejemplo, quedó con un gap de 100 % y no hay forma de saber si le costaría semanas o años en hallar el óptimo.

Temporada	Tipo	Tiempo empleado	gap	Mejora respecto al real
2017/2018	viajes lógicos	72 horas	13,37 %	13,5 %
2017/2018	longitud ≤ 3	72 horas	50,02 %	-60,01 %
2017/2018	longitud ≤ 4	72 horas	100 %	-46,23 %
2018/2019	viajes lógicos	72 horas	7,07 %	24,87 %
2018/2019	longitud ≤ 3	72 horas	16,87 %	14,21 %
2018/2019	longitud ≤ 4	72 horas	100 %	-23,56 %
2019/2020	viajes lógicos	72 horas	12,00 %	19,92 %
2019/2020	longitud ≤ 3	72 horas	16,32 %	8,32 %
2019/2020	longitud ≤ 4	72 horas	45,99 %	-21,36 %

Cuadro 4.4: Cuadro comparativo de 9 corridas distintas para las últimas 3 temporadas

Para intentar mitigar el problema se armó un último conjunto de corridas. La idea era probar lo mismo que antes: para cada año los casos con viajes de tamaño menor o igual a 3 y 4.

La diferencia es que ahora, para ayudar al *solver*, ponemos como punto de inicio para el *branch & cut* (conocido como *warm-start*) la mejor solución factible disponible.

Esto es, para las temporadas 2018/2019 y 2019/2020 las corridas de 72 horas con viajes lógicos y para 2017/2018 la corrida de 168 horas.

Ya que el *warm-start* es una solución entera factible, la solución encontrada final será igual o mejor que la mejor que teníamos hasta el momento. Desgraciadamente, sea porque el problema es demasiado grande o porque la solución ya es suficientemente buena, el *solver* mejora apenas las soluciones ofrecidas o no las mejora en absoluto.

Temporada	Tipo	Tiempo empleado	gap	Mejora respecto al real
2017/2018	longitud ≤ 3	72 horas	11,83 %	15,95 %
2017/2018	longitud ≤ 4	72 horas	100 %	15,81 %
2018/2019	longitud ≤ 3	72 horas	8,72 %	24,87 %
2018/2019	longitud ≤ 4	72 horas	100 %	24,87 %
2019/2020	longitud ≤ 3	72 horas	11,36 %	19,96 %
2019/2020	longitud ≤ 4	72 horas	19,04 %	19,92 %

Cuadro 4.5: Corridas con punto inicial predefinido

Notar que la "mejora respecto al real" fue igual o casi igual a la de las mejores soluciones (fila correspondiente a los viajes lógicos, cuadro 4.4), excepto la de la temporada 2017/2018 en donde se usó la corrida de más horas.

Capítulo 5

Conclusiones

Hemos recorrido el universo de los problemas lineales, estudiando su comportamiento y proponiendo el método *simplex* para resolverlos.

También hemos visto los problemas enteros y mixtos y analizado estrategias para resolver estos.

Luego hemos pasado a estudiar el problema mixto que nos compete: el "*Traveling Tournament Problem*" o **TTP**. El **TTP** consiste en construir un *fixture* de tipo *double round robin*, esto es, todos juegan contra todos de local y visitante, minimizando distancias recorridas.

Vimos que este problema mezclaba características de optimización con características de factibilidad, más propias del *constraint programming*. También vimos lo difícil del problema y distintas técnicas para intentar simplificarlo.

Vimos, por ejemplo, que podíamos relajar el tiempo y permitir que hubiera "fechas libres" para facilitar la factibilidad. O limitar cuáles viajes eran posibles y cuáles no. O prefijar valores y sobre ese resultado armar un *fixture*.

Todas esas técnicas las usamos después en nuestro problema particular del vóley. Este campeonato es un *double round robin* como queremos y una cantidad lo suficientemente baja de equipos para poder modelarlo con un modelo de programación mixta.

Nuestro plan fue mejorar el modelo anterior que usaba un sistema de parejas para armar un *fixture* con distancias bajas. Además, se jugaban en específicas fechas dobles cada fin de semana. Vimos que este acercamiento lograba buenos resultados con relativo poco esfuerzo computacional pero que podía ser mejorado.

Luego recorrimos los detalles del nuevo modelo que proponía:

- Romper el sistema de parejas, desacoplando los equipos y permitiéndoles hacer viajes impensados para el modelo anterior.
- Tomar como viajes posibles los admitidos por los equipos, ayudando a la factibilidad y a la posterior aceptación del *fixture* óptimo.
- Permitir que los partidos se desarrollaran *cualquier día de la semana*, no solo jueves y sábados.

Con este nuevo modelo matemático resolvimos el problema del vóley generando un *fixture* para las últimas 3 temporadas.

En los 3 casos se experimentó una reducción en la cantidad de kilómetros recorridos, en un **15, 81 %** para la temporada 2017/2018, en un **24, 87 %** para la temporada 2018/2019 y un **19, 92 %** para la inconclusa temporada 2019/2020.

Vimos que, en la temporada 2018/2019, las distancias se redujeron para **todos** los equipos, más allá de haberse reducido globalmente.

Además logramos seguir todas las restricciones especiales del campeonato: no jugar los días de campeonatos internacionales, jugar día por medio estando de viaje, tener descansos antes y después de cada viaje, respetar un máximo de días sin jugar y tener una última fecha para todos los equipos, entre otras.

5.1. Desventajas

Aún cuando se mejore la distancia recorrida, se cumplan todos los pedidos de los equipos, se utilicen viajes deseados por los mismos equipos, no todo es mejora en la nueva solución. Existe un punto que puede resultar una gran desventaja para la propuesta de este nuevo campeonato: la posibilidad de jugar cualquier día de la semana.

Aunque a priori puede parecer mejor o peor según la persona, hay algo seguro. El público no cuenta ya con la seguridad de que su equipo juega siempre, por ejemplo, los días jueves.

Este nuevo esquema requiere de seguidores un poco más adeptos, que sigan el *fixture* para saber en qué fecha va a jugar su equipo el próximo partido. Esto podría eventualmente llevar a una reducción en el público, afectando económicamente al club.

Pero, más importante, podría afectar a los jugadores. Y aquí el hecho es mucho más simple e inevitable: puede haber una resistencia al cambio. Cambiar la forma en la que se hacen las cosas siempre es difícil y más cuando se vienen haciendo hace muchos años.

Sin embargo, es común también que una vez hecho el cambio se mire hacia atrás y se convenza que el cambio fue bueno.

Esto fue lo que pasó, por ejemplo, en el caso del básquet de Argentina [5]. Allí se tardó mucho en adoptar el nuevo formato, pero hoy en día nadie querría volver atrás.

Anhelo que en unos años podamos decir lo mismo del vóley.

5.2. Próximos pasos

Quedan muchos puntos para seguir estudiando el problema, tanto en el plano teórico como práctico.

Desde el punto de vista más teórico se puede seguir estudiando los límites del problema, qué variantes de n, o de cantidad de días se pueden probar y que el

problema siga siendo resoluble. Además se puede intentar conseguir desigualdades válidas que ayuden a los tiempos de ejecución. Esto nos permitirá ir agrandando el problema para encontrar mejores óptimos o imaginar torneos más y más grandes.

De hecho, una ventaja de esto es poder ir llevando la solución a distintos torneos y distintos deportes sin dejar de usar, en esencia, la misma estrategia.

Los otros puntos, más prácticos, vienen de pensar en agregados que se le pueden hacer a este mismo problema del vóley.

En nuestro caso nos preocupamos básicamente por reducir la distancia de los viajes globalmente en el primer paso y dar prioridad a los días preferidos por los equipos en el segundo. Podría estudiarse distintos tipos de función objetivo para adaptar el *fixture* a otras necesidades de los equipos.

Estas podrían ser

- Minimizar la distancia del equipo que más viaja.
- Maximizar la distancia del equipo que menos viaja.
- Minimizar la diferencia entre el equipo que más viaja y el equipo que menos viaja.
- Minimizar la máxima cantidad de días de descanso entre partido y partido dada una longitud prefijada para el campeonato.

Además, podrían agregarse restricciones para todo lo que es campeonatos extras de los equipos. No es igual el descanso si el partido se juega en San Pablo que si se juega en Buenos Aires. Además, a un equipo que no sea de Buenos Aires puede convenirle jugar un partido en Buenos Aires antes o después de tener un viaje en avión a Brasil. Estas distintas disposiciones específicas de los equipos en distintos días del torneo podrían definirse explícitamente en las restricciones.

Por último y quizás lo más importante, está el tema de la potencial infactibilidad en la segunda parte de nuestra solución. Nuestro problema está separado en dos partes, la segunda alimentándose de la solución de la primera.

Por más que haya un intento (la relajación en el tiempo, por ejemplo) nada asegura que el óptimo del primer problema va a llevar a un resultado factible en el segundo. Está claro que en este caso hay que cambiar la solución del primero y volver a intentar pero, ¿cómo?

Es probable que el primer modelo tenga varios óptimos y al quedarnos con uno solo rechazamos a todo el resto, quizás mejores candidatos para el segundo modelo.

Un camino interesante para estudiar es el de poder lograr soluciones equivalentes para barajar en el segundo modelo, sin que sea excesivamente complicado computacionalmente.

Incluso se podría intentar modificar levemente la solución que tenemos para ajustarla al segundo modelo, sin que empeore mucho su valor en la función objetivo. Podría ocurrir que algunas vecindades simples del punto en el que estamos parado

corrijan las infactibilidades y nos eviten correr el primer modelo (el más costoso) desde cero.

Por suerte seguirá habiendo torneos desafiantes que pongan al límite nuestra capacidad de crear *fixtures* óptimos y que nos permitan seguir estudiando el tema de manera teórica.

Siempre y cuando sigan existiendo los deportes, siempre y cuando siga habiendo vóley, siempre y cuando haya alguien con ganas de pegarle a una pelota de *volea* para pasarla del otro lado de la red, solo por diversión.

Bibliografía

[1] K Easton, G Nemhauser & M Trick (2001). "The traveling tournament problem: Description and benchmarks", Lecture Notes in Computer Science. 2239. 580-584. 10.1007/3-540-45578-7_43.

[2] D Applegate, R E Bixby, v Chvátal, & W Cook (2006). "The traveling salesman problem: A computational study". Princeton: Princeton University Press.

[3] A Schaerf (1999). "Scheduling Sport Tournaments using Constraint Logic Programming". Constraints. 4. 43-65. 10.1023/A:1009845710839.

[4] F Bonomo, A Cardemil, G Durán, J Marenco & D Saban (2012). "An application of the traveling tournament problem: The Argentine volleyball league", Interfaces 42 (3), 245-259.

[5] G Durán, S Durán, J Marenco, F Mascialino, P A Rey (2017). "Scheduling Argentina's Professional Basketball Leagues: A Variation on the Relaxed Travelling Tournament Problem". European Journal of Operational Research. 275 (3), 1126-1138.

[6] C C Ribeiro, S Urrutia (2006). "Maximizing breaks and bounding solutions to the mirrored traveling tournament problem". Discrete Applied Mathematics. 154. 1932-1938. 10.1016/j.dam.2006.03.030.

[7] R Hoshino & K Kawarabayashi (2012). "The Linear Distance Traveling Tournament Problem". Proceedings of the National Conference on Artificial Intelligence. 3.

[8] D C Uthus, P J Riddle, H W Guesgen (2012). "Solving the traveling tournament problem with iterative-deepening A*". Journal of Scheduling - SCHEDULING. 15. 1-14. 10.1007/s10951-011-0237-x.

[9] D Uthus, P Riddle & H Guesgen (2009). "DFS* and the Traveling Tournament Problem". 5547. 279-293. 10.1007/978-3-642-01929-6_21.

[10] C Ribeiro & S Urrutia (2007). "Heuristics for the Mirrored Traveling Tournament Problem". European Journal of Operational Research. 179. 775-787. 10.1016/j.ejor.2005.03.061.

[11] R Bao (2006). “Time Relaxed Round Robin Tournament and the NBA Scheduling Problem”. Master’s thesis, Cleveland State University.

[12] R Melo, S Urrutia & C Ribeiro (2009). “The traveling tournament problem with predefined venues”. J. Scheduling. 12. 607-622. 10.1007/s10951-008-0097-1.

[13] R Hoshino & K Kawarabayashi (2011). “The Multi-Round Balanced Traveling Tournament Problem”. ICAPS 2011 - Proceedings of the 21st International Conference on Automated Planning and Scheduling.

[14] G Durán, M Guajardo, J Miranda, D Sauré, S Souyris, A Weintraub, & R Wolf (2007). “Scheduling the Chilean soccer league by integer programming”. Interfaces. 37. 539-552.

[15] G Durán, M Guajardo, & J M D Sauré (2017). “Scheduling the south american qualifiers to the 2018 fifa world cup by integer programming”. European Journal of Operational Research. 262 (3). 1109-1115.

[16] C Fleurent & J Ferland (1993). “Allocating games for the NHL using integer programming”. Operations Research. 41. 649-654. 10.1287/opre.41.4.649.

[17] G L Nemhauser & M A Trick (1998). “Scheduling A Major College Basketball Conference”. Operations Research. 46. 10.1287/opre.46.1.1.

[18] R Gomory (1958). “Outline of an Algorithm for Integer Solutions to Linear Programs”. Bulletin of the American Mathematical Society. 64. 275-278. 10.1090/S0002-9904-1958-10224-4.

[19] E Balas, S Ceria, G Cornuéjols & N R Natraj. (1996). “Gomory cuts revisited”. Operations Research Letters. 19. 1-9. 10.1016/0167-6377(96)00007-7.