Tesis de Licenciatura

OPTIMIZACION DE FIXTURES DEPORTIVOS: ESTADO DEL ARTE Y UN ALGORITMO TABU SEARCH PARA EL TRAVELING TOURNAMENT PROBLEM

Andrés Cardemil
ac7ℓ@dc.uba.ar

Director: Dr. Guillermo A. Durán

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Diciembre 2002

Resumen

La planificación de scheduling deportivos es una tarea muy compleja que las ligas organizadoras de torneos enfrentan frecuentemente. En este trabajo repasamos las características más importantes de los problemas que se presentan al intentar diseñar fixtures deportivos, y por otro lado, proponemos un algoritmo Tabu Search para el denominado Traveling Tournament Problem(TTP), el cual fue propuesto por M. Trick, G. Nemhauser y K. Easton como base para analizar y comparar distintas técnicas de resolución de esta clase de problemas. Nuestro algoritmo obtiene muy buenos resultados, entre ellos, las mejores soluciones (al momento de la presentación de este trabajo) para dos de las instancias propuestas.

Abstract

Sport Scheduling is a complex task that League organizers must face frequently while preparing their tournaments. In this work we firstly describe the characteristics of the problems that generally arise on this issue. On the other hand, we propose an heuristic algorithm based on Tabu Search to solve the so called Traveling Tournament Problem (TTP), which was introduced by M. Trick, G. Nemhauser and K. Easton with the idea of using it as a benchmark problem for the study and comparison of different resolution techniques. We were able to find very good results. Among them, the best solutions (at the time of the presentation of this work) for two of the proposed instances.

Agradecimientos

A mi director, Guillermo Durán, por su gran predisposición, buena voluntad y excelente trato desde el primer minuto hasta el último.

A mis compañeros: a los que compartieron conmigo distintos momentos a lo largo de todos estos años, a los que colaboraron con esta tesis, y a los que me dieron una mano con LaTeX.

A mis padres, que sencillamente me ayudaron y me bancaron en todo. Sin ellos, hubiese sido muy difícil (sino imposible) llegar a este punto.

Índice general

Capítulo 1

Introducción

Las ligas deportivas suelen enfrentarse con muchos problemas y dificultades a la hora de diseñar sus fixtures. Esto se debe principalmente a que tienen que satisfacerse diversos tipos de requisitos como los que imponen los contratos televisivos, los equipos participantes, la disponibilidad de estadios, etc. Sumado a esto, es común que se intenten optimizar ciertos aspectos tales como minimizar las distancias que los equipos involucrados deben recorrer a lo largo de un torneo. La gran diversidad de requisitos y condiciones que pueden presentarse y que deben considerarse, hace que esta sea una tarea muy compleja.

Además, otro aspecto a tener en cuenta en este tipo de problemas, es la enorme cantidad de soluciones factibles que pueden generarse. Esto lo convierte en un problema de Optimización Combinatoria muy interesante. En general, cuando se deben satisfacer determinados requisitos o minimizar algún aspecto en particular el problema de armado de fixtures deportivos pertenece a la clase NP-completo (en [35] puede verse la demostración para un ejemplo concreto).

Por otro lado, en muchos casos, no sólo es importante conseguir un "buen fixture" que satisfaga todos los requerimientos, sino que es necesario que el sistema implementado, obtenga el resultado en forma "rápida" (minutos u algunas horas a lo sumo), de forma tal que los organizadores puedan interactuar con el mismo de alguna manera, permitiéndoles incorporar al modelo básico algunos ajustes y modificaciones con facilidad, especialmente en las ligas donde existen intereses en conflicto, y es común que estos cambien por negociaciones o acuerdos de último momento.

En lo que resta de este capítulo, repasamos las publicaciones existentes sobre la planificación de fixtures e introducimos el marco teórico en el cual

se desarrolla esta tesis. En el capítulo 2, describimos por un lado algunas características del armado de fixtures, y por otro, los requisitos y condiciones que más frecuentemente se deben satisfacer. En el capitulo 3 presentamos el *Traveling Tournament Problem* que fue propuesto por M. Trick (junto con G. Nemhauser y K. Easton) en [41] con el objetivo de poder comparar distintas técnicas. Luego, en el capitulo 4, describimos en detalle un algoritmo *Tabu Search* que desarrollamos para resolver dicho problema, con el que obtuvimos muy buenas soluciones. En el capitulo 5, presentamos nuestros resultados, y por último, se encontrarán nuestras conclusiones y algunas ideas para continuar trabajando en el tema.

1.1. Literatura Existente

Si bien todavía existen muchas ligas que resuelven el problema en forma "manual", en los últimos años ha habido un creciente interés por desarrollar y utilizar métodos matemáticos y computacionales para el armado de fixtures. Se han propuesto diversas técnicas, y muchas de ellas se han aplicado a problemas reales, tales como los descriptos en [5, 13, 25, 29, 39, 45, 46, 47].

Entre las técnicas utilizadas, podemos destacar los trabajos realizados en base a Programación Entera [16, 37, 39, 40], *Constraint Programming* [24, 25, 26, 27, 35], y métodos heurísticos [5, 29, 45, 46, 48]. En el ámbito local, podemos mencionar el trabajo de Dubuc [14] en el cual se utilizó Simulating Annealing para armar el fixture de la AFA que fue utilizado en el Torneo Apertura de fútbol 1994-1995, que básicamente debía satisfacer las condiciones impuestas por la televisión y otras relativas a la seguridad.

1.2. Marco Teórico

1.2.1. Definiciones básicas

A continuación introduciremos algunos conceptos sobre grafos.

Denotamos un grafo G por un par $(V(G), E(G))$, o simplemente (V, E) donde V representa un conjunto finito de vértices (o nodos), y E, un conjunto de pares no ordenados de vértices de G, llamados arcos o aristas. Sean $n = |V|$ y $m = |E|$.

Un vértice v es adyacente a otro vértice w en G si $(v, w) \in E$. Decimos que v y w son los extremos de la arista. Se dice también que v y w son

vecinos.

Se define el grado de un vértice v como la cantidad de vecinos que posee.

Dos grafos G y H son isomorfos si existe una biyección entre $V(G)$ y $V(H)$ que conserva las adyacencias. En este caso, notamos $G = H$.

Un grafo G es completo si cualquier par de vértices distintos de G son adyacentes. Denotaremos K_n al grafo completo con n vértices.

Se dice que un grafo G tiene un coloreo de arcos válido, si a cada arco $e \in E$ se le asigna un color, de forma tal que si dos arcos p y q tienen algún nodo en común, entonces p y q tienen asignados colores distintos.

Para más detalle o para encontrar otras definiciones sobre grafos no presentadas aquí puede verse [2] y [23].

1.2.2. Optimización Combinatoria

Se dice que un determinado problema es de optimización combinatoria si consiste en encontrar entre una cantidad finita de subconjuntos de un conjunto dado, aquél que cumpliendo con ciertas restricciones maximice o minimice una cierta función objetivo. Una forma de resolver esta clase de problemas consiste en evaluar la función objetivo para todas las soluciones válidas (enumeración o fuerza bruta). Lamentablemente, este método, si bien en teoría parece sencillo, es tan costoso de realizar (en términos de tiempo) que aún con las computadoras con mayor poder de cálculo tomaría años resolver problemas de tamaño "mediano".

Por ejemplo, para el problema del viajante de comercio (TSP), en el cual un comerciante debe visitar, minimizando la distancia total recorrida, N ciudades para después retornar al lugar donde empezó, existen $\frac{(N-1)!}{2}$ soluciones posibles. De esta forma, si una cierta computadora puede evaluar 3000 soluciones en un segundo, significa que puede resolver este problema para 10 ciudades en un minuto (porque para ese caso existen $\frac{(10-1)!}{2} = 181440$ soluciones). Esta misma computadora tardaría casi 2 horas en resolver un problema para 12 ciudades, 168 días para 15 ciudades y varios años o siglos si seguimos aumentando la cantidad de ciudades. Como se puede observar, esta técnica de resolución no es factible para este tipo de problemas. Se han planteado otras alternativas para resolver estos problemas en forma exacta, pero para la mayoría de ellos, todos los algoritmos desarrollados hasta el momento, tienen, en el peor caso, el mismo comportamiento que la enumeración completa.

1.2.3. Metaheurísticas

Las metaheurísticas proveen un marco general que permite crear nuevos algoritmos combinando diferentes conceptos tomados de heurísticas básicas, de la física, de la biología, y de otras disciplinas. Entre las metaheurísticas más conocidas, podemos mencionar a Tabu Search, Algoritmos Genéticos, Simulated Annealing y GRASP. Definimos a una metaheurística como un proceso iterativo que guía a una heurística subordinada combinando inteligentemente distintos conceptos para explorar el espacio de búsqueda a partir de soluciones iniciales (generadas en forma aleatoria o bien por un algoritmo diseñado a tal efecto). En cada paso del algoritmo, se generan nuevas soluciones en base a las obtenidas en las etapas previas, pudiendo utilizar para ello diversos criterios.

Tabu Search

Tabu Search (TS) [17, 18, 28] fue propuesta a fines de los años 80 por F. Glover. Se sustenta en el diseño de técnicas cuyos objetivos son cruzar regiones de optimalidad local. TS es un mecanismo general de imposición de restricciones que se usa como guía para la implementación de una heurística definida para un problema particular. Estas restricciones se basan principalmente en la exclusión o prohibición directa de alternativas de búsqueda (clasificadas tabú), las que se definen en función de lo realizado en las etapas anteriores de la misma.

TS se basa en la premisa de que la resolución de un problema debe incorporar memoria flexible y exploración sensible o responsable (*responsive exploration*). La característica de memoria flexible del TS, permite la implementación de procedimientos que sean capaces de realizar una búsqueda eficiente en el espacio de soluciones. Como las elecciones locales están guiadas por la información recolectada durante (las etapas previas de) la búsqueda, TS contrasta con otros diseños que no utilizan memoria y que se sustentan fuertemente en procesos semi-aleatorios que implementan alguna forma de muestreo. La memoria flexible también contrasta con los diseños de memoria rígidos típicos de las estrategias branch and bound.

La exploración sensible y responsable, proviene del hecho de considerar que una mala elección estratégica (durante el proceso de búsqueda) provee (en general) mayor información que una buena elección aleatoria. La exploración responsable integra los principios de una búsqueda inteligente, es decir, explotar las características de las soluciones buenas, al mismo tiempo que se exploran regiones prometedoras no visitadas. TS trata de encontrar

nuevas y mejores formas de aprovechar los mecanismos asociados a estas dos ideas, convirtiéndose en una metodología para resolución de problemas muy flexible y eficaz, como lo han demostrado numerosas aplicaciones a diversos problemas clásicos de optimización combinatoria y problemas reales.

Vecindario y Lista Tabú

Sea S el conjunto de soluciones del problema a tratar, y sea $f : S \to \mathbb{R}$ una función de costo sobre S que se intenta optimizar. Introducimos, entonces, el concepto de vecindario $N(s)$ para cada $s \in S$. $N(s)$ es el conjunto de soluciones alcanzables desde s por medio de una pequeña modificación (o movimiento) m. Formalmente, $N(s) = \{s' \in S \ / \ s' = s \oplus m, \ m \in M\}$ donde M contiene todas las posibles modificaciones y "$s' = s \oplus m$" significa que s' se obtiene aplicando la modificación m a s.

Pueden producirse movimientos de inserción o de permutación. En el caso de grafos, por ejemplo, un movimiento de inserción podría ser agregar un nuevo arco o un nuevo nodo, mientras que el hecho de intercambiar las posiciones de dos nodos, sería un movimiento de permutación. Obviamente, se pueden implementar estrategias que combinen ambas características, incorporando además (si se quiere) información propia del problema a tratar. En definitiva, podemos decir que TS permite gran flexibilidad en la definición de la vecindad.

Otro elemento fundamental de TS (donde aparece el concepto de memoria flexible) es lo que se denomina *Lista Tabú* (LT). La LT está compuesta por "movimientos prohibidos", pues mantiene los atributos de soluciones que fueron cambiados en los últimos movimientos realizados. La LT puede funcionar simplemente como una cola, o se puede definir un tiempo de permanencia en la lista distinto para cada uno de los elementos que la componen. La idea de la LT, es restringir el vecindario de soluciones que se genera en cada iteración, limitándolo a aquellas soluciones que se obtienen sin realizar ningún movimiento considerado tabú, para evitar quedar atrapado en óptimos locales.

Criterio de aspiración

Para evitar "perderse" soluciones buenas, es frecuente que en ciertas circunstancias, se permita violar las restricciones impuestas por la lista tabú. El ejemplo más claro, es el de permitir realizar un movimiento tabú si la

solución que se obtiene con éste, es mejor que todas las encontradas hasta ese momento.

Memoria a largo plazo

En algunas aplicaciones, también se incorpora el concepto de memoria a largo plazo (en contraposición a la memoria de corto plazo recién descripta) y sus estrategias asociadas. Esto en general, consiste en almacenar soluciones denominadas elite (óptimo local de alta calidad), que fueron encontradas en distintas etapas del proceso de búsqueda, para que puedan ser aprovechadas más adelante, como por ejemplo, en las etapas de intensificación que se describen a continuación. Otra alternativa interesante, consiste en tratar de identificar que atributos suelen encontrarse en soluciones de alta calidad, para que por ejemplo, se trate de que el manejo de la lista fuerce a que en general las soluciones a analizar los conserven.

Intensificación y Diversificación

Otros elementos muy importante del TS son las estrategias de intensificación y diversificación, las cuales están vinculadas con el uso de memoria a largo plazo.

Las estrategias de intensificación se refieren básicamente al hecho de explotar regiones del espacio de soluciones consideradas buenas. Están basadas en la modificación de las reglas de elección para estimular combinaciones de movimientos y atributos de una buena solución ya encontrada. Además, pueden incluir desde un regreso a regiones consideradas buenas para realizar la misma búsqueda TS pero con una lista tabú de menor tamaño, hasta la implementación de algoritmos de optimización local particulares al problema.

Por otro lado, la diversificación, se refiere al hecho de permitir al algoritmo analizar distintas regiones del espacio de búsqueda, evitando volver siempre a las mismas zonas. En su caso mas extremo, esto puede implicar reiniciar la búsqueda desde una nueva solución generada al azar. Las estrategias de diversificación son particularmente de ayuda cuando soluciones mejores pueden encontrarse sólo si se cruzan barreras en la topología del espacio de búsqueda.

Esquema básico del Tabú Search

A continuación, describimos el esquema general de TS (para el caso de minimización), que se puede observar en la figura 1.1.

generar la solución inicial s
$mejorSol = s$
Mientras NO se cumpla la condición de parada, **Hacer**
  generar el conjunto V^* de soluciones $s_i = s \oplus m_i$
    tal que o $m_i \notin LT$ o bien s_i satisface el criterio de aspiración.
  elegir la mejor solución $s' \in V^*$ (respecto de la función f a optimizar)
  actualizar la LT con m_i
  $s = s'$
  Si $f(s') < f(mejorSol)$, **entonces** $mejorSol = s'$
  Cada "cierto tiempo", intensificar y/o diversificar
Fin Mientras

Figura 1.1: Esquema básico de Tabú Search para un problema de minimización

TS comienza la búsqueda desde una solución inicial cualquiera (incluida en S), generada en forma aleatoria o en forma determinística. Luego se va moviendo iterativamente desde una solución a otra vecina de acuerdo a la definición de vecindad que se utilice. Sumado a esto, en ciertas etapas de la búsqueda se realizan los procesos de intensificación y diversificación. El algoritmo termina cuando se cumple el criterio de parada establecido, que suele involucrar una cantidad máxima de iteraciones totales, o una cantidad máxima de iteraciones sin mejoras.

Capítulo 2

Características del armado de Fixtures

2.1. Descripción general

En una competencia deportiva, n equipos deben jugar entre sí (1 o más veces) en un cierto período de tiempo siguiendo algún esquema. Uno de los esquemas más populares es el denominado *Round Robin* donde todo equipo e juega contra todos los demás una cierta cantidad r de veces a lo largo de todo el torneo. Si r es 1, el esquema se denomina *Single Round Robin* (SRR). Si r es 2, se denomina *Doble Round Robin* (partido y revancha). Por lo general, un partido se juega en el estadio de uno de los dos equipos. Se dice que un equipo juega de local cuando lo hace en su propio estadio, y que juega de visitante en caso contrario.

A grandes rasgos, podemos clasificar los fixtures en dos grandes tipos:

Fixtures Restringidos Temporalmente: Son aquéllos en los que todos los partidos se distribuyen de forma tal de minimizar la cantidad de fechas o rondas requeridas. Ejemplos de fixtures de este tipo pueden verse en [25, 27, 39] y otros.

Fixtures Relajados Temporalmente: Son aquéllos en los que los partidos pueden distribuirse libremente a lo largo de todas las rondas que se deseen. Ejemplos típicos de esta clase de fixtures son los de las grandes ligas de basketball (NBA) en EEUU, o la de Hockey (NHL) como se describe en [5] y [16].

Por ejemplo, un torneo de 4 equipos que siga el esquema *Round Robin* necesitará al menos 3 rondas. Entonces, si se requiere armar un fixture temporalmente restringido este deberá tener exactamente 3 rondas (ver figura 2.1). En general, un torneo simple con n equipos requerirá por lo menos $n-1$ rondas si n es par y por lo menos n en el caso de que sea impar. Para simplificar las cosas, a lo largo de todo el trabajo asumiremos que n (la cantidad de equipos), es par, ya que para contemplar los casos en donde n es impar, sólo tendremos que agregar un equipo "ficticio" o "comodín". Entonces, cuando el fixture resultante indique que un cierto equipo e debe jugar contra ese equipo "ficticio" en la ronda r, diremos que e queda libre en la ronda r.

2.2. Requisitos y condiciones habituales

Como dijimos anteriormente, existe una gran diversidad de requisitos, condiciones y objetivos que los equipos, medios televisivos y otros actores interesados en el diseño de los fixtures suelen imponer a los organizadores.

Algunos de estos requisitos se refieren a los patrones de localías (HAP [1]). El HAP de un cierto equipo e es la secuencia de partidos que juega de local y de visitante. Por ejemplo la secuencia `LLVLVV` indica que los dos primeros partidos se jugará de local, el tercero de visitante, etc.

A continuación describiremos algunos de los requisitos mas frecuentes que los organizadores deben considerar:

- Para evitar favorecer o perjudicar a algún equipo, es necesario que todos jueguen la misma cantidad de partidos de local y de visitante.
- Ningún equipo puede jugar más de una cierta cantidad de partidos seguidos de local o de visitante. Además, para no dar ventajas, si dos equipos juegan entre sí mas de una vez, se suele exigir que lo hagan alternándose la localía.
- Dos equipos a y b deben tener HAP complementarios. Es decir que cuando A juega de local, B debe hacerlo de visitante y viceversa. Esto puede deberse a que A y B comparten el mismo estadio, o que por razones de seguridad y/o de transporte sea recomendable que jueguen en zonas alejadas.

[1] HAP: Home Away Pattern, siguiendo la notación empleada en [8, 39] y otros

- Un cierto equipo E debe jugar obligatoriamente de visitante (o de local) en una ronda o fecha determinada. Esto puede deberse, por ejemplo, a que su estadio ha sido reservado para otro evento en esa fecha.
- Dos equipos A y B no pueden enfrentarse en determinadas rondas. Esto es común que suceda entre dos rivales clásicos, y que por ejemplo, no se quiere que jueguen entre sí en las primeras rondas del torneo.
- Dos equipos A y B deben enfrentarse en una determinada ronda. Esto también puede darse entre dos rivales clásicos, y que por ejemplo, se quiere que jueguen en la última ronda del torneo.
- En los casos en que se jueguen dos vueltas (partido y revancha), es posible que se pida que el fixture sea **espejado**, es decir que la segunda vuelta sea igual a la primera pero invirtiendo las localías de todos los partidos. También es posible que no se requiera esto, sino que se solicite que exista al menos una cierta cantidad de rondas entre el partido y revancha de cada par de equipos. Si bien esto último brinda mayor flexibilidad, también incrementa el espacio de soluciones.
- Minimizar las distancias que los equipos recorren a lo largo del torneo. Esto puede ser con el objetivo de minimizar los costos y tiempos de los viajes, así como contribuir al mejor descanso de los jugadores.

2.3. Representación: Varias caras del mismo problema

Como primer paso para empezar a analizar el problema de armado de fixtures, estudiaremos las diversas formas en que estos se pueden representar. Incluiremos modelos matemáticos para presentar formalmente el problema. Para esto, tomaremos el caso general de un torneo *Single Round Robin* de n equipos que está temporalmente restringido. Es decir que los únicos requisitos a cumplir son los determinados por dicho esquema y el hecho de que debe realizarse en la menor cantidad de rondas posibles, o sea $n-1$ (n par). Queremos destacar, que para el caso general existen varios algoritmos polinomiales que generan fixtures válidos. Incluso, De Werra en [8, 9, 10] propone algunos métodos para construir fixtures que permiten satisfacer algunas restricciones simples sobre los HAP. Pero al agregar cierto tipo de condiciones y objetivos a optimizar, el problema suele resultar mucho más complicado. En [35], por ejemplo, se demuestra que cuando existen varias restricciones del tipo "Los

equipos e_1 y e_2 no pueden jugar en determinada ronda" o bien "un cierto equipo e_1 no puede jugar de local en una determinada ronda" el problema es NP-completo.

2.3.1. Tablas o Matrices

La representación mas común es simplemente ver el fixture como una gran tabla o matriz de $(n-1) \times n$ en donde se resumen todos los partidos que conforman un torneo de n equipos. Asumiremos que los equipos se representan con los números del 1 al n, (y eventualmente con las n primeras letras, esto es $1 \equiv A$, $2 \equiv B$, etc.). Entonces, diremos que el valor de la celda M_{re} de una cierta matriz M que representa un fixture indica el equipo rival con el que juega el equipo e en la ronda r. Además, para completar la representación asumiremos que un valor negativo en la celda M_{re} indica que el equipo e juega de visitante en la ronda r, mientras que un valor positivo significa que juega de local. La tabla de la figura 2.1, nos indica por ejemplo, que el equipo A deberá jugar contra B en la primera ronda de local, y contra D en la tercera de visitante. (los valores negativos se representan con el simbolo "@").

Rda	A	B	C	D
1	B	@A	D	@C
2	C	D	@A	@B
3	@D	C	@B	A

Figura 2.1: Representación de un Fixture de 1 vuelta simple. (@A debe leerse: 'A es local')

2.3.2. Cuadrados Latinos

Un *Cuadrado Latino* [3, 44] es una matriz de $n \times n$ donde en cada posición se puede colocar cualquier valor entero entre 1 y n con la condición de que ninguno aparezca más de una vez en la misma fila o columna. Siguiendo los pasos que se describen a continuación [3, 29] podemos obtener un fixture SRR: *i*) Construir un cuadrado latino de $(n-1) \times (n-1)$ en el cual no se repita ningún número en la diagonal principal, *ii*) Construir una nueva fila y una nueva columna copiando los valores de la diagonal y *iii*) Dejar en blanco los valores de la diagonal (se puede asumir que se los completa con el valor n para tener un cuadrado latino de $n \times n$).

③	1	2
1	②	3
2	3	①

aplicando los pasos descriptos, obtenemos:
⟶

	A	B	C	D
A	-	1	2	③
B	1	-	3	②
C	2	3	-	①
D	③	②	①	-

Figura 2.2: A partir de un Cuadrado Latino de 3×3 con valores distintos en la diagonal, obtenemos una representación de un fixture SRR para 4 equipos.

El resultado, es una matriz de $n \times n$ cuyas celdas representan el número de ronda en el que se deben enfrentar los distintos equipos. Por ejemplo, en la matriz resultante de la figura 2.2 (que representa el mismo fixture que la figura 2.1) el valor que se encuentra en la celda $[2, 3]$ o (B, C) indica que los equipos 2(B) y 3(C) jugarán entre si en la ronda 3.

Por las propiedades matemáticas que poseen los cuadrados latinos, estos han sido utilizados en varios contextos para analizar distintos aspectos de los fixtures. Por ejemplo, en [33] se usó el concepto de "rectángulos latinos" para demostrar la existencia de *fixtures prematuros* (ver 2.5). Nosotros, por otro lado, por razones de performance, al implementar el algoritmo que presentamos en la sección 4, utilizamos una estructura muy similar para almacenar parte de un fixture en memoria (con esto, podemos conocer en orden 1 en que ronda se enfrentan dos equipos determinados), según se detalla en la sección 4.11.

2.3.3. Grafos

También podemos utilizar grafos como forma de representación. Aquí explicaremos como hacer corresponder un coloreo de arcos y una 1-factorización de un grafo completo K_n con un fixture de n equipos.

Coloreo de arcos

Un grafo completo de n nodos $G = (V, E)$ y un coloreo válido de sus arcos también representan un fixture. Dado que G es un grafo completo y n es par, es fácil ver que el coloreo de sus arcos necesita $n-1$ colores. Supongamos que se utilizan los colores 1, 2..., $n-1$. Entonces para armar un fixture a partir de dicho coloreo, lo que se hace es lo siguiente: Para determinar los partidos

de la ronda i (recordemos que un fixture SRR tiene $n-1$ rondas), se toman todos los arcos pintados con el color i. Cada uno de los arcos representa un partido. Por ejemplo, el arco que une los nodos u y v indica que el equipo representado por el nodo u juega contra el equipo representado por v. En la figura 2.3, se puede ver un coloreo del grafo K_4 que representa el mismo fixture que las figuras anteriores.

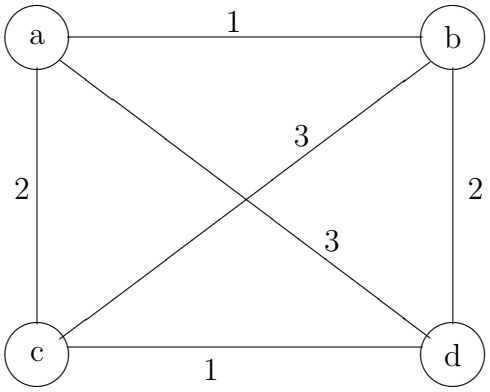


Figura 2.3: Un coloreo de arcos para K_4. Los arcos que tienen asociado el color i representan los partidos de la ronda i.

Factorización de grafos

Equivalentemente, el problema de armar un fixture SRR con n equipos se puede relacionar también con el problema de construir una 1-factorización de un grafo completo $G = (V, E)$ de n nodos. Esto es, particionar el conjunto de arcos E en $n-1$ particiones, denominadas *1-factores* , de $\frac{n}{2}$ arcos cada una, de forma tal que en cada *1-factor* los arcos no sean adyacentes (tomados de a dos), o dicho de otra forma, que cada vértice de V aparezca exactamente una vez.

Veamos ahora como una 1-factorización de un grafo completo G con n nodos, (cuyos 1-factores se numeran del 1 al $n-1$), se corresponde con un fixture SRR de n equipos:

- Cada nodo representa un equipo.
- Un arco (u, v) de un determinado 1-factor r representa un partido entre u y v en la ronda r.
- Cada 1-factor representa una ronda.

- Las propiedades de la factorización garantizan que cada equipo juegue contra todos los demás exactamente una vez. Esta propiedad es utilizada en algunos métodos constructivos para el armado de fixtures. Puede verse [43] para más detalles.

2.3.4. Programación Entera

En base a un modelo de Programación Entera, también podemos representar un fixture *Single Round Robin* de n equipos como se describe a continuación[36, 37]:

Sea X_{ijr} una variable binaria que toma el valor 1 sólo si el equipo i juega contra el equipo j en la ronda r (podemos además asumir que i juega de local y j de visitante) y 0 en otro caso. El problema, entonces consiste en encontrar una matriz $X[x_{ijr}]$ de ceros y unos que cumpla las siguientes condiciones:

$$\sum_{i=1}^{n}(X_{ijr} + X_{jir}) = 1 \quad \forall j, r \tag{2.1}$$

$$\sum_{r=1}^{n-1}(X_{ijr} + X_{jir}) = 1 \quad \forall\, i \neq j \tag{2.2}$$

2.1 Cada equipo juega exactamente una vez por ronda (ya sea de local o visitante)
2.2 Cada equipo juega contra todos los demás exactamente una vez.

Una solución factible para este conjunto de restricciones constituye un fixture SSR. Para un problema dado, se puede querer obtener sólo una solución cualquiera, todas las posibles, o la mejor solución posible respecto de alguna función objetivo que se quiera optimizar.

2.3.5. Constraint Satisfaction Problem

Ahora modelaremos el armado de un fixture con el esquema *Single Round Robin* como un *Constraint Satisfaction Problem*(CSP). Siguiendo lo expuesto en [22] y [27], diremos que un determinado CSP se define por una tupla (X, D, C) donde:

- $X = \{x_1, x_2, ..., x_k\}$ es un conjunto finito de variables.

- $D = \{D_1, D_2, ..., D_k\}$ asigna a cada variable $x \in X$ un dominio finito D_x de posibles valores.
- $C = \{c_1, c_2, ..., c_p\}$ es un conjunto finito de p restricciones. Cada elemento $c_i \in C$ expresa una restricción sobre los valores de las variables $x_1....x_k$ especificando que combinaciones de valores son compatibles.

Entonces, una vez definidos X, D *y* C el problema consiste en asignar valores de sus respectivos dominios a todas las variables x_i de forma tal de que se respeten las restricciones definidas en C. En este caso, también podemos querer simplemente encontrar sólo una solución, encontrar todas las posibles, o tal vez una solución óptima (o por lo menos una solución "buena") respecto de algún objetivo definido en términos de las variables.

Como el conjunto de todas las asignaciones posibles (no necesariamente "compatibles") está definido por el producto cartesiano de los dominios $D_1 \times D_2 \times ... \times D_k$, resolver un CSP implica determinar una asignación particular entre un número potencialmente enorme de posibles soluciones, lo que en general se traduce en que muchos de los problemas modelados de esta forma, son muy difíciles de resolver.

Utilizando la representación descripta en 2.3.1 podemos definir el problema de armado de fixtures SRR como un *Constraint Satisfaction Problem*. Allí vimos que (de manera informal) un fixture se puede representar por una matriz de $(n-1) \times n$ cuyas celdas M_{re} indican el equipo rival con el que juega el equipo e en la ronda r.

Definimos entonces, un problema *Single Round Robin* para n equipos, (con n par) de las siguiente manera:

- X contiene las variables M_{re} donde $r = 1 \ldots n-1$ y $e = 1 \ldots n$
- $D_{re} = \{-n, \ldots, -1,\ 1, \ldots,\ n\} \setminus \{e, -e\} \qquad (e = 1 \ldots n,\ r = 1 \ldots n-1)$
- C contiene las siguientes restricciones.

$$|M_{r,e}| \neq |M_{s,e}| \ \forall\, r \neq s \quad (e = 1...n; \quad r, s = 1...n-1) \tag{2.3}$$

$$[\ i \neq j \Rightarrow |M_{r,i}| \neq |M_{r,j}|\ ] \quad \forall r : 1 \leq r \leq n-1, \quad (i, j = 1...n) \tag{2.4}$$

$$[\ M_{r,i} = j \Leftrightarrow M_{r,j} = -i\ ] \quad \forall r : 1 \leq r \leq n-1, \quad (i, j = 1...n) \tag{2.5}$$

2.3 Todos los equipos juegan contra todos, uno distinto en cada ronda.
2.4 En cada ronda, todos juegan contra un equipo distinto
2.5 Si A juega de local contra B, entonces B juega de visitante contra A

2.4. Espacio de Búsqueda

Al intentar contar la cantidad exacta de soluciones factibles, nos dimos cuenta que esta era una tarea muy difícil. Descubrimos que hasta el momento, no se conoce ninguna fórmula general para saber cuantos fixtures SRR se pueden generar para n equipos. Lo que si se sabe es que dicho número crece exponencialmente en función de n.

Con el objetivo de contar las posibles soluciones para valores pequeños de n implementamos un programa con el que logramos obtener la siguiente secuencia: {1; 6; 720; 31,449,600} que representa la cantidad de fixtures distintos[2] que se pueden armar para $2, 4, 6$ y 8 equipos respectivamente (términos $0, 1, 2$ y 3). Buscando en [38], "la enciclopedia on-line de secuencias enteras", encontramos un término mas de la secuencia (444,733,651,353,600), la que se correspondía con el número de "matrices simétricas de $(2n + 1) \times (2n + 1)$ en las cuales cada una de sus filas es una permutación de $1..(2n + 1)$".

Recordando la representación de Fixtures como Cuadrados Latinos descripta en la sección 2.3.2 (pág. 16) podemos ver fácilmente que esto se corresponde con el número de fixtures distintos para $2n + 2$ equipos. Si bien, como dijimos anteriormente, no se conoce ninguna fórmula exacta para contar la cantidad de fixtures SSR, en [44] pueden verse algunas aproximaciones que tienen que ver con el número de cuadrados latinos.

En [12], Dinitz demuestra que la cantidad de 1-factorizaciones No-isomorfas del grafo K_{12} es 526,915,620 y se explica en detalle el método utilizado. Luego, deduce que la cantidad de 1-factorizaciones distintas para K_{12} es $252, 282, 619, 805, 368, 320$. A partir de esto, podemos calcular que la cantidad de fixtures distintos para $n = 12$ equipos es 11! veces ese número, ya que por cada 1-factorización, podemos permutar sus correspondientes 1-factores (que representan las 11 rondas de un torneo) de 11! formas distintas. En general, esto significa que a partir de una sola 1-factorización de K_n (n par), podemos obtener $(n - 1)!$ fixtures distintos, permutando el orden de sus rondas (1-factores).

En resumen, podemos observar fácilmente que el tamaño del espacio de soluciones crece de manera altamente exponencial a medida que crece el valor de n, siendo incluso mucho mayor a $(n!)^2$ para $n \geq 10$. De hecho, el número total de cuadrados latinos de orden n se puede calcular de la siguiente manera[44]: $N(n) = n! \times (n-1)! \times L(n)$ donde $L(n)$ (que se calcula

[2]Consideramos que dos fixtures f_1 y f_2 para los equipos $1..n$ son distintos, si $\exists r_1, e_1$ tal que $M^{f_1}_{r_1,e_1} \neq M^{f_2}_{r_1,e_1}$. De esta forma, estamos considerando que dos fixtures que tengan invertidas el orden de alguna de sus rondas, también son fixtures distintos.

numéricamente), representa la cantidad de cuadrados latinos normalizados (aquéllos que su primera fila y columna están dados por $1, 2, \ldots, n$). Por ejemplo, $L(12) = 1{,}62 \times 10^{44}$, $L(14) = 2{,}33 \times 10^{70}$.

Si bien la fórmula presentada contempla todos los posibles cuadrados latinos, y a nosotros sólo nos importan los que no tienen valores repetidos en la diagonal (ver 2.3.2), igualmente estos números nos dan una idea de la complejidad del problema de implementar algún tipo de búsqueda sobre este enorme espacio de soluciones. Y hasta ahora sólo analizamos el caso de un problema de una sola vuelta (SRR) donde no nos importaba quien jugaba de local o visitante. Si tenemos en cuenta estas variantes, evidentemente el espacio de soluciones se hace todavía mucho mas grande.

2.5. Fixtures Prematuros

Otro problema que en apariencia parece más sencillo, pero que está muy lejos de serlo, es el de determinar si es posible obtener un fixture válido partiendo con algunos partidos ya establecidos (por ejemplo, ya se han definido las cuatro primeras rondas). A la generalización de este problema se lo conoce con el nombre de *Quasigroup Completion Problem* (QCP) [1, 19, 20, 32]. Los cuadrados latinos están asociados a elementos algebraicos conocidos como *quasi-groups*. De esta forma, recurriendo a la representación vista en la sección 2.3.2 (pág. 16) podemos ver este problema, como el de completar un cuadrado latino que solo está lleno parcialmente (en forma correcta). Cuando no es posible completar un cuadrado latino (fixture) con algunos valores (partidos) pre-establecidos, diremos que tenemos un *fixture prematuro*. Se ha demostrado que este problema es NP-completo [4, 33]. También se ha observado, que existe una fase de transición[19, 20] (relacionada con la cantidad de celdas llenas inicialmente), en la cual se produce un cambio entre las instancias que pueden ser resueltas (es decir completar el cuadrado latino) y las que no pueden resolverse (es decir, que no se pueden completar correctamente). De hecho, las instancias del problema que requieren más tiempo para ser resueltos (de una manera u otra, es decir poder completar el cuadrado latino o determinar que no es posible) se encuentran en dicha fase de transición, donde existen suficientes restricciones (celdas llenas) que limitan el número de alternativas, pero no demasiadas para ver rápidamente que no tienen solución. Dicho de otra forma, las instancias en las cuales hay sólo unas pocas celdas pre-establecidas se puede completar correctamente y en poco tiempo. Cuando la mayor parte de las celdas están llenas, el problema también se puede resolver (de una manera u otra) en poco tiempo. Pero, por

otro lado, las instancias que se encuentran en la fase de transición requieren mucho mas esfuerzo, y además, las instancias que se encuentran en el borde de dicha fase, que suelen ser solubles, se convierten en no solubles con sólo llenar unas pocas celdas adicionales.

Gomes ha creado una página web [21] donde se pueden probar y analizar la resolución de distintas instancias del QCP. En esta demostración, se utiliza una matriz de 10×10, que en lugar de números, debe completarse con 10 colores distintos.

Capítulo 3

Traveling Tournament Problem

3.1. Descripción y Motivación

En [41], M. Trick, G. Nemhauser y K. Easton propusieron una clase de problemas que denominaron *Traveling Tournament Problem*(TTP) los cuales abstraen algunos aspectos claves del armado de fixtures combinando condiciones en el HAP y el objetivo de minimizar distancias.

Dados un entero U (entre 1 y n) y una matriz de distancias $DIST$ de $n \times n$, el TTP consiste en obtener un fixture *Doble Round Robin* para n equipos (NO necesariamente espejado) temporalmente restringido (es decir con $2 \times (n-1)$ rondas) con las siguientes características:

- Objetivo: Minimizar la distancia recorrida por los equipos a lo largo de todo el torneo
- Ningún equipo puede jugar más de U partidos seguidos de visitante ni de local.
- Ningún par de equipos puede enfrentarse en dos rondas consecutivas.
- Se asume que todos los equipos empiezan en su lugar de origen y deben retornar al mismo al final del torneo. Cuando un equipo juega varios partidos seguidos de visitante, viaja del estadio de un rival a otro (sin pasar por su lugar de origen)

Definir un valor de $U = 1$ implica que todos los equipos deben jugar alternadamente de local y de visitante. Esto significa que para $n > 2$ no existe solución, pues como se muestra en [8] y [9], no se puede crear un

Rda	A	B	C	D
1	B	@A	D	@C
2	C	D	@A	@B
3	@D	C	@B	A
4	@B	A	@D	C
5	D	@C	B	@A
6	@C	@D	A	B

Figura 3.1: Fixture válido para el TTP de 4 equipos. (@A debe leerse: 'A es local')

fixture con esas características (al menos $n-2$ equipos tienen que repetir su condición de local o visitante una vez). Por otro lado, si tomamos $U = n$, el problema se parece al del viajante de comercio (TSP), porque en este caso, el recorrido óptimo que podría realizar cada equipo es similar al recorrido óptimo del TSP ya que un equipo podría jugar todos sus partidos de visitante seguidos y regresar a su estadio (evidentemente no es factible que todos los equipos realicen su recorrido de esta forma, pero seguramente tendrán muchos "tramos" en común con su mejor tour TSP asociado). Para otros valores de U el problema sigue resultando muy complejo. Incluso para instancias pequeñas (ej: $n = 10$) todavía no se ha logrado obtener la solución óptima.

En definitiva, podemos ver que el TTP modela importantes aspectos prácticos del problema de armado de fixtures. Sus características que incluyen una mezcla de factibilidad y optimalidad, sumado al hecho de que no existe una larga historia en la resolución de este tipo de problemas, hacen que el TTP sirva como base para un análisis profundo y pueda ser utilizado como *benchmark* para comparar y estudiar diversas técnicas que intenten resolver el problema.

Para más detalles sobre el TTP, o para ver las instancias y sus resultados conocidos, puede verse su sitio web en [42]. Allí figuran algunos de los resultados de esta tesis como mejor solución (al momento de presentación de este informe) en dos de las instancias propuestas (NL12 y NL14, ver figura 5.1). Además, pueden observarse varios de los fixtures que obtuvimos durante la etapa de desarrollo, que hasta agosto del 2002 superaban a todas las soluciones previamente conocidas, que luego fueron mejoradas por Zhang, Shen y nuestros propios resultados.

3.2. Trabajos anteriores

En [41] se señala que se aplicaron técnicas de Programación entera y de *Constraint Programming* para estudiar el problema. En [34] se propone un enfoque mixto que incluye técnicas de relajación Lagrangiana. A su vez, mientras escribíamos este informe, tomamos conocimiento (a través del sitio web del TTP donde se van informando las nuevas soluciones obtenidas) de que Zhang y otros investigadores estaban también trabajando en el tema, obteniendo excelentes resultados. Por otro lado, en [31] puede verse una demo (hecha en Java) que resuelve las instancias más sencillas del TTP.

En estos trabajos, se han analizado principalmente instancias donde el valor de U se definía en 3, y como consecuencia, los únicos resultados publicados hasta el momento corresponden a dichas instancias. Debido a esto, hemos decidido concentrarnos en analizar este caso en particular, es decir, que a partir de ahora, cuando nos referimos al TTP, estaremos asumiendo que el valor de U es 3.

Recientemente, en PATAT'02 han sido presentados dos nuevos trabajos. En [15], se describe un enfoque que combina *Constraint Programming* y Programación entera. En [6] se desarrolla una estrategia heurística basada en Colonia de Hormigas.

3.3. Formulación matemática

Representaremos el problema como un CSP extendiendo la formulación descripta en la sección 2.3.5 (pág. 19) para un torneo SRR. Además de las restricciones propias del TTP que deben incorporarse al modelo, otra diferencia importante es que el TTP es un torneo DRR y por lo tanto la cantidad R de rondas del torneo es $2 \times (n-1)$. Dado que ahora es muy importante saber quien juega de local y de visitante, recordemos que un valor positivo en la celda M_{re} indica que el equipo e juega de local en la ronda r, mientras que un valor negativo (representado por la @ en las figuras) indica que dicho equipo juega de visitante en la ronda r. A continuación, modelamos el TTP como un CSP mediante la tupla (X, D, C):

- X contiene las variables M_{re} $\quad (e = 1...n,\ r = 1...R)$
- $D_{re} = \{-n, \ldots, -1,\ 1, \ldots,\ n\} \setminus \{e, -e\}$ $\quad (e = 1 \ldots n,\ r = 1 \ldots R)$
- C contiene las siguientes restricciones: $\quad (e = 1...n,\ r = 1...R)$

↬ Restricciones generales para un DRR (no espejado):

$$M_{r,e} \neq M_{s,e} \; \forall\, r \neq s \quad (e = 1...n; \quad r, s = 1...R) \tag{3.1}$$

$$[\ (i \neq j) \Rightarrow |M_{r,i}| \neq |M_{r,j}|\ ] \quad (i, j = 1...n) \tag{3.2}$$

$$[M_{i,r} = j \Leftrightarrow M_{j,r} = -i] \quad \forall r : 1 \leq r \leq R, \quad (i, j = 1...n) \tag{3.3}$$

↬ Restricciones particulares para TTP (fijando $U = 3$):

$$|M_{r,e}| \;\neq\; |M_{r+1,e}| \tag{3.4}$$

$$[M_{r,e} > 0 \wedge M_{r+3,e} > 0] \Rightarrow [M_{r+1,e} < 0 \vee M_{r+2,e} < 0] \tag{3.5}$$

$$[M_{r,e} < 0 \wedge M_{r+3,e} < 0] \Rightarrow [M_{r+1,e} > 0 \vee M_{r+2,e} > 0] \tag{3.6}$$

3.1 Todo equipo juega dos veces (partido y revancha) contra todos los demás. Notar que a diferencia de lo definido en 2.3.5, aquí no se utilizan los valores absolutos
3.2 En cada ronda, todos juegan contra un equipo distinto
3.3 Si A juega de local contra B, entonces B juega de visitante contra A
3.4 Ningún par de equipos juegan entre si en dos rondas seguidas
3.5 Ningún equipo juega más de 3 partidos seguidos de local
3.6 Ningún equipo juega más de 3 partidos seguidos de visitante

Capítulo 4

Nuestro Algoritmo

Aquí presentamos un algoritmo Tabu Search (ver sección 1.2.3) que desarrollamos para resolver el *Traveling Tournament Problem*. Se detallarán los distintos elementos que lo componen.

4.1. Espacio de Soluciones

De acuerdo a lo descripto en la sección 3.3 (pág. 26), diremos que una solución válida para el TTP es cualquier matriz M de dimensión $R \times n$ que satisfaga las restricciones allí expuestas. Notaremos al conjunto de soluciones factibles (fijando $U = 3$) como F.

4.2. Solución inicial

Implementamos dos algoritmos para generar la solución inicial. Ambos generan fixtures en forma aleatoria, pero el primero de ellos lo hace siempre siguiendo un mismo "esquema" o "patrón" establecido, ya que es un algoritmo determinístico. Este, es una variante del propuesto en [36] para generar fixtures balanceados de una sola vuelta. El fixture que se obtiene al utilizar nuestra variante de este algoritmo, depende de la numeración previa de los equipos por lo que permutando dicha numeración, podemos generar en principio cerca de $(n-1)!$ fixtures distintos (este número no es exacto, porque se pueden obtener fixtures equivalentes a partir de permutaciones distintas). Además, al momento de generar los partidos de la segunda vuelta, los equipos se vuelven a permutar, de forma tal que no se empiece con fixtures espejados,

porque en la práctica, observamos que de esta manera es mas frecuente que se obtengan mejores resultados. Entonces, la cantidad de fixtures distintos que se pueden generar (con este algoritmo) es en realidad bastante mayor, cercana a $(n-1)!^2$.

Por otro lado, el segundo algoritmo que se basa en el propuesto por Dinitz en [11] sí genera fixtures aleatorios sin seguir ningún patrón en especial. La idea principal de este algoritmo es ir eligiendo partidos al azar y agregarlos al fixture, y en caso de que ese partido ya se hubiese definido en otra ronda, por ejemplo, éste se elimina, asegurándose de esta forma, que en cada paso del algoritmo la cantidad de partidos definidos siempre aumente o se mantenga constante.

Ambos algoritmos generan fixtures válidos para torneos DRR pero no siempre son válidos para el TTP. Por lo tanto, al resultado que se obtiene con cualquiera de estos algoritmos, se le aplica una serie de mejoras (basadas en lo descripto en 4.6) y correcciones en los HAP de los equipos que violen las restricciones establecidas por el TTP. En caso de que no se pueda corregir rápidamente, se repite el proceso hasta obtener uno correcto.

4.3. Función de Evaluación

Utilizaremos como función de evaluación la suma de los recorridos de cada equipo a lo largo del torneo, ya que justamente el objetivo del *Traveling Tournament Problem* es minimizar dicha suma, respetando las restricciones expuestas.

Primero, definimos que la matriz $DIST$ de dimensión $n \times n$ es la matriz de distancias, y por lo tanto el valor denotado por $DIST_{i,j}$ representa la distancia entre el estadio del equipo i y el estadio del equipo j.

Por otro lado, recordando que el TTP especifica que todos los equipos empiezan (ronda 0) y terminan (ronda $R+1$) el torneo en su propio estadio, asumiremos que $M_{0,k} > 0$ y que $M_{R+1,k} > 0$ $(\forall k : 1...n)$. Luego, definimos a $T_{k,r}$ como la distancia que debe recorrer el equipo k del estadio donde juega en la ronda $r-1$ al estadio donde juega en la ronda r, de la siguiente manera:

$$T_{k,r} = \begin{cases} DIST_{|M_{r-1,k}|,\,|M_{r,k}|} & \text{si } (M_{r-1,k} < 0) \wedge (M_{r,k} < 0) \quad (i) \\ DIST_{|M_{r-1,k}|,\,k} & \text{si } (M_{r-1,k} < 0) \wedge (M_{r,k} > 0) \quad (ii) \\ DIST_{k,\,|M_{r,k}|} & \text{si } (M_{r-1,k} > 0) \wedge (M_{r,k} < 0) \quad (iii) \\ 0 & \text{si } (M_{r-1,k} > 0) \wedge (M_{r,k} > 0) \quad (iv) \end{cases}$$

(i) El equipo k juega de visitante en las rondas r y $r-1$. Entonces la distancia buscada es la que separa los estadios de los dos rivales de k en las rondas r y $r-1$
(ii) El equipo k juega de local en la ronda r y de visitante en $r-1$. Luego, la distancia buscada es la que separa al estadio de k con el estadio de su rival en la ronda $r-1$.
(iii) El equipo k juega de local en la ronda $r-1$ y de visitante en r. Luego, la distancia buscada es la que separa al estadio de k con el estadio de su rival en la ronda r.
(iv) El equipo k juega de local en las rondas r y $r-1$. Por lo tanto, permanece en su estadio y la distancia que recorre entre esas rondas es nula.

Finalmente, definimos a la función de evaluación $f : F \rightarrow \mathbb{N}$ de la siguiente manera:

$$f(x) = \sum_{e=1}^{n} C_e$$

donde C_e indica la distancia total recorrida por el equipo e a lo largo de todo el torneo. El valor de los C_e $(e = 1...n)$ se calcula de esta forma:

$$C_e = \sum_{r=1}^{R+1} T_{e,r}$$

4.4. Vecindarios

A diferencia del esquema clásico de TS, en lugar de definir un sólo vecindario que dirige la búsqueda en el espacio de soluciones, nosotros definimos varios, los cuales se van alternando en distintas etapas del proceso de exploración. Luego de varios experimentos, observamos que en general siempre se obtenían mejores resultados cuando se utilizaba uno de ellos en particular, al que denominamos principal. De todas maneras, decidimos mantener la idea de usar varias vecindades para guiar la búsqueda, pero utilizando el resto de las vecindades con menor frecuencia y duración que el principal. Con esto, no sólo disminuimos el riesgo de quedarnos en mínimos locales, sino que a la vez estamos haciendo una especie de diversificación indirecta de la búsqueda. En las próximas secciones detallaremos cuales son los vecindarios que utilizamos. Asumiremos que S es el conjunto de soluciones factibles de torneos DRR sin considerar las restricciones particulares del TTP. De esta forma, tenemos que $F \subset S$. Luego veremos como se incorporan las restricciones adicionales que plantea el TTP al proceso de búsqueda. Para todas los

vecindarios que se describirán a continuación, se considerarán vecinos sólo a aquellos fixtures válidos para el TTP. En consecuencia, durante el proceso normal de búsqueda, sólo se estarán explorando regiones factibles. Pero en las etapas de diversificación, el algoritmo si se moverá intencionalmente por regiones no factibles, con el objetivo de llegar a otras regiones factibles que de otra manera podrían no ser visitadas nunca. Por último, antes de detallar cada uno de los vecindarios, queremos destacar que en todos ellos se cumple la propiedad de simetría, es decir que "x es vecino de z si y solo si z es vecino de x"; propiedad muy deseable cuando se define una vecindad para realizar una búsqueda Tabú.

4.4.1. Intercambio parcial de rondas (IPR) - Vecindario Principal

La vecindad $IPR(x)$ de un fixture $x \in S$ consiste en todos los fixtures que se pueden obtener aplicando a x un movimiento IPR. Un movimiento IPR es una permutación de cuatro partidos entre dos rondas (r_1, r_2) que involucra a sólo cuatro equipos (e_1, e_2, e_3, e_4). Para que un movimiento IPR (definido por r_1, r_2, e_1, e_2, e_3 y e_4) pueda ser aplicado a un fixture x, este debe satisfacer las siguientes propiedades:

- $x.rival(e_1, r_1) = x.rival(e_2, r_2) = e_3$
- $x.rival(e_1, r_2) = x.rival(e_2, r_1) = e_4$

donde $x.rival(e, r)$ indica el equipo contra el que juega e en la ronda r en el fixture x, es decir $|M^x_{r,e}|$.

Si se cumplen las condiciones expuestas, entonces tenemos que en x están definidos los siguientes cuatro partidos: [e_1 vs e_3 en la ronda 1], [e_1 vs e_4 en la ronda 2], [e_2 vs e_4 en la ronda 1] y [e_2 vs e_3 en la ronda 2]. El movimiento consiste en intercambiar las rondas de estos partidos manteniendo la estructura de todos los demás. Esto significa que de estos cuatro partidos, los que se jugaban en la ronda r_1 pasan a jugar en la ronda r_2, mientras que los que se jugaban en r_2 lo harán en r_1. En definitiva, el resultado de aplicar un movimiento IPR a un fixture x (que cumple las propiedades pedidas), es un nuevo fixture y que tiene las siguientes características:

- $y.rival(e_1, r_1) = y.rival(e_2, r_2) = e_4 = x.rival(e_1, r_2) = |M^x_{r_2,e_1}|$
- $y.rival(e_1, r_2) = y.rival(e_2, r_1) = e_3 = x.rival(e_1, r_1) = |M^x_{r_1,e_1}|$

- *el resto de los partidos se mantienen igual.*

Aunque se puede observar fácilmente que al aplicar un movimiento IPR sobre un fixture cualquiera, el fixture resultante siempre será un fixture que cumpla el esquema DRR, es importante destacar que esto no es siempre cierto para que el fixture sea considerado válido para el TTP. En particular, al aplicar un movimiento IPR podremos obtener un vecino que viola cualquiera de las restricciones definidas por 3.4, 3.5 y 3.6.

Bajo esta definición, estamos manteniendo las localías de los partidos. Es decir que si el partido entre e_1 y e_3 que se cambia de ronda indicaba que e_1 era local, en la nueva ronda lo seguirá siendo. Pero otra posibilidad, es que también se cambien las localías, en cuyo caso, se estaría afectando también la localía del partido revancha. Para permitir que el movimiento IPR sea más productivo, decidimos incorporar esta idea. Esto es, cuando se ve que se puede realizar un cierto movimiento IPR se consideran todas las variantes posibles respecto de la localía de los cuatro partidos que se alteran originalmente (2^4), seleccionando aquella con la que se obtiene el mejor resultado. En la práctica, esta variante produjo mejoras importantes, y además permite realizar movimientos que de otra forma podrían no considerarse, porque se podrían violar los límites que el TTP establece para la cantidad de partidos seguidos que se juegan de local o visitante. A pesar de esto, (aunque poco frecuente) es posible, especialmente para las instancias con pocos equipos, que no se pueda realizar ningún movimiento IPR válido. Cuando esto sucede, automáticamente el algoritmo deja de utilizar esta vecindad como guía en el proceso de búsqueda, y continúa usando alguna de las que se describen más adelante.

Rda	A	B	C	D	E	F
1	E	@F	@D	C	@A	B
2	B	@A	E	@F	@C	D
3	F	@E	D	@C	B	@A
4	@C	D	A	@B	F	@E
5	.	.	.	.	.	.

$\leftrightarrow$

$\leftrightarrow$

Rda	A	B	C	D	E	F
1	F	@E	@D	C	B	@A
2	B	@A	E	@F	@C	D
3	E	@F	D	@C	@A	B
4	@C	D	A	@B	F	@E
5	.	.	.	.	.	.

Figura 4.1: Dos fixtures vecinos según IPR. El movimiento que lleva de uno a otro está dado por los equipos A, B, E, F y por las rondas 1 y 3. El resto de las rondas (que no se ven en la figura) son iguales en ambos fixtures.

Para analizar todos los vecinos de un fixture f según IPR, utilizamos el algoritmo de la figura 4.2. Veamos ahora cual es la complejidad del mismo. El ciclo exterior requiere $O(n^2)$ (porque hay $\binom{n}{2}$ pares de equipos). El interior requiere $O(R)$. Por otro lado, lo que más tiempo requiere en el cuerpo del ciclo, es calcular el costo de los vecinos que se obtienen, que es de $O(n^2)$. Pero en realidad, lo que nos interesa es comparar la diferencia entre todos ellos, y por lo tanto, nos alcanza con calcular la diferencia entre el costo del fixture que produce cada movimiento con el costo del fixture original. Esto puede hacerse en $O(4n)$ pues basta con restar el costo (C_e) de los 4 equipos del fixture original involucrados en el movimiento y sumar esos mismos costos del vecino. En principio, tenemos que "teóricamente" en el peor caso, la complejidad del algoritmo es de $O(n^2 \times R \times 4n) \equiv O(n^4)$. Pero en la práctica, suele ser (un grado) menor, porque el costo sólo se evalúa cuando se encuentran vecinos, y de acuerdo a las pruebas realizadas, la cantidad de vecinos siempre fue inferior a n^2.

Recordando que este procedimiento se repite en cada iteración de la búsqueda Tabú, la cual se va moviendo de un vecino a otro, analicemos lo siguiente: En la iteración i se paso de un fixture x a un vecino z. Luego, en la iteración siguiente, se deberán calcular todos los vecinos de z para continuar la búsqueda. Pero en realidad, si tenemos en cuenta que entre x (para el que previamente habíamos calculado y almacenado todos sus movimientos IPR) y z sólo hay algunos partidos de diferencia, en lugar de calcular nuevamente todos los vecinos de z, podemos tomar los previamente calculados para x descartando los que involucran a las rondas y equipos afectados por el movimiento que lleva de uno a otro, y buscar solamente los "nuevos" vecinos que pueden haber aparecido. En resumen, podemos decir que x y z tienen la mayoría de sus vecinos en común, por lo que para conocer todos los vecinos de z a partir de los de x solo hace falta calcular los que son propios de z (y no de x). Implementando esta idea en el algoritmo de búsqueda, logramos reducir los tiempos que demora el algoritmo en analizar el vecindario IPR, ya que con esto, nos basta con re-calcular el costo de los vecinos de x encontrados en la vuelta anterior (lo que requiere O(4n) para cada uno, y como dijimos el párrafo anterior, en la práctica se observó que la cantidad de vecinos se mantiene menor a n^2) y para buscar los nuevos vecinos, en lugar de utilizar el algoritmo descripto, se usa una variante en la cual desaparece el ciclo que iteraba por todas las rondas, porque nos alcanza con sólo analizar las dos rondas del último movimiento realizado, ya que si existen vecinos nuevos, estos deben involucrar por lo menos alguna de las rondas que se modificaron para pasar de x a z.

Figura 4.2: Algoritmo para calcular (entre todos los posibles) el mejor vecino IPR de f

Función ObtenerMejorVecino (Fixture f)

$mejorCostoVec = \infty$
$mejorMov = nil$
Para todo par de equipos e_1, e_2 **Hacer**
  Para cada ronda $r = 1 \dots R$ **Hacer**
    e_3 = f.rival(e_1, r)
    e_4 = f.rival(e_2, r)
    Si los equipos e_1, e_2, e_3, e_4 y el par de rondas (r, f.ronda(e_1, e_4))
    o bien el par (r, f.ronda(e_4, e_1)) forman un movimiento IPR m válido
     (es decir que cumple las propiedades descriptas en 4.4.1) **entonces**
      Construir el vecino vec aplicando m a f
      $costoVec$ = calcularCostoTTP(vec, $Dist$)
      Si ($costoVec < mejorCostoVec$ y m no es Tabú)
      o vec cumple el criterio de aspiración **entonces**
        $mejorCostoVec = costoVec$
        $mejorMov = m$
      Fin Si
    Fin Si
  Fin para
Fin para
Devolver $mejorMov$

Por otro lado, otra variante implementada, con el objetivo de darle mayor velocidad al algoritmo, consiste en elegir el primer vecino que disminuye el costo actual, en lugar del "mejor" entre todos los vecinos. De esta forma, la elección del movimiento a realizar es bastante más rápida. Para esto, es importante que los "pares de equipos" que se van eligiendo en las distintas iteraciones se hagan en distinto orden, para evitar que se utilicen siempre los mismos equipos.

4.4.2. Intercambio de rondas - IR

La vecindad $IR(x)$ de un fixture $x \in S$ consiste en todos los fixtures que se pueden obtener a partir de x, permutando un par de rondas r_1 y r_2. Es

decir, que simplemente se permutan dos filas de la matriz M^x asociada al fixture x. Esta claro que el fixture resultante luego de una permutación de este tipo no viola ninguna de las tres primeras condiciones sobre M^x pero si pueden violarse cualquiera de las tres últimas.

Para analizar todos los vecinos $IR(x)$ se deben seleccionar todos los subconjuntos de dos rondas (es decir, $\binom{R}{2}$ casos), lo que requiere $O(\frac{R^2}{2})$. Luego, por cada selección hay que permutar las rondas del fixture original y evaluar la diferencia de costo que se obtiene, lo que realizamos en $O(2n + 4n)$ pues sólo hace falta re-calcular los valores $T_{i,r}$ que involucran a las rondas $r_1, r_1 + 1, r_2$ y $r_2 + 1$. Por lo tanto, la complejidad de elegir un vecino IR es de $O(nR^2) \equiv O(n^3)$.

En este caso (así como en el resto de las vecindades que se describen a continuación) también es posible elegir directamente el primer vecino que mejora el costo actual, en lugar de analizar todos los posibles. De esta forma, aunque en el peor caso teórico la complejidad es la misma, en la práctica se puede lograr que el proceso de búsqueda sea más rápido.

4.4.3. Intercambio de equipos - IE

La vecindad $IE(x)$ de un fixture $x \in S$ consiste en todos los fixtures que se pueden obtener a partir de x, permutando un par de equipos e_1 y e_2. Es decir, que simplemente se permutan dos columnas de la matriz M^x asociada al fixture x. Esta permutación es equivalente a renombrar los equipos, por lo que esta vez, ninguna permutación de este estilo puede generar (a partir de fixtures válidos) fixtures inválidos.

Para analizar todos los vecinos $IE(x)$ se deben seleccionar todos los subconjuntos de dos equipos (es decir, $\binom{n}{2}$ casos), lo que requiere $O(\frac{n^2}{2})$. Luego, por cada selección hay que permutar los equipos del fixture original y evaluar la diferencia de costo que se obtiene, lo que realizamos en orden lineal, porque sólo evaluamos los tramos que se modifican. Por lo tanto, la complejidad de elegir un vecino IE es de $O(n^2 n) \equiv O(n^3)$.

4.4.4. Intercambio de localías - IL

La vecindad $IL(x)$ de un fixture $x \in S$, consiste en todos los fixtures que se pueden obtener a partir de x, cambiando la localía de los dos partidos entre un mismo par de equipos e_1 y e_2. Es decir, que si en x, e_1 jugaba de local contra e_2 en la ronda r_1, y de visitante en la ronda r_2, las únicas diferencias

entre x y el vecino que se obtiene permutando las localías de los equipos e_1 y e_2 es simplemente que en este último, e_1 juega contra e_2 de visitante en la ronda r_1 y de local en r_2. Esto implica cambiar el signo de los valores de las celdas $M[r_1, e_1]$, $M[r_2, e_1]$, $M[r_1, e_2]$ y $M[r_2, e_2]$. Está claro que el fixture resultante luego de una permutación de este tipo sólo puede llegar a violar las restricciones 3.5 o 3.6 pues la "estructura" de las matriz se mantiene igual.

Para analizar todos los vecinos $IL(x)$ se deben seleccionar todos los equipos tomados de a dos, lo que requiere $O(\frac{n^2}{2})$. Luego, por cada selección hay que invertir las localías de los partidos que el par de equipos juegan entre sí y evaluar la diferencia de costo que se obtiene, lo que realizamos en $O(1)$ pues sólo es necesario calcular una cantidad fija (8) de coeficientes $T_{e,r}$. Por lo tanto, la complejidad de elegir un vecino IL es de $O(n^2)$.

4.5. Manejo de la Lista Tabú

La Lista Tabú registra cuales son los movimientos considerados tabú. Según cual sea la vecindad que se esta usando para dirigir la búsqueda, en la Lista Tabú se guardan los últimos t movimientos realizados. Durante el proceso normal de búsqueda, el valor t se modifica cada cierta cantidad de iteraciones que se definen como parámetro, eligiendo su valor en forma aleatoria en un rango determinado (que también se establece en los parámetros de entrada). Cada vez que se realiza un movimiento, éste se incorpora a la lista, y a su vez (si la lista esta completa), se elimina de ésta el movimiento que mayor tiempo ha permanecido como tabú. En resumen, podemos decir que la Lista Tabú tiene un comportamiento FIFO (el primer elemento que llega es el primero en irse).

4.5.1. Criterio de Aspiración

El criterio de aspiración utilizado, consiste en permitir realizar un movimiento que está considerado tabú, cuando la aplicación del mismo genera una solución cuyo costo es inferior al mejor costo obtenido hasta ese momento. Además, es importante destacar, que si en algún momento la lista tabú llegara a prohibir a todos los posibles movimientos, el algoritmo inmediatamente cambia la dirección de la búsqueda seleccionando otra vecindad.

4.6. Optimizaciones locales

Sumado al proceso de búsqueda tabú que realiza la exploración del espacio de soluciones utilizando como guía (alternadamente) las distintas vecindades anteriormente descriptas, el algoritmo cada cierta cantidad de iteraciones (que se definen por parámetro) intenta efectuar algunas pequeñas optimizaciones locales, a saber:

- Permutación de 2 y 3 rondas: Se evalúa el costo de todos los fixtures que se pueden obtener permutando 2 (o 3) rondas entre sí. Si el que obtiene menor costo es mejor que el fixture original, entonces se aplica la permutación correspondiente, logrando una mejora local. Para evitar influir en la estrategia tabú, cuando la búsqueda que se está realizando está siendo dirigida por la vecindad IR, esta optimización no se realiza.

- Permutación de 2 y 3 equipos: Se evalúa el costo de todos los fixtures que se pueden obtener permutando (o renombrando) 2 (o 3) equipos. Si el que obtiene menor costo es mejor que el fixture original, entonces se aplica la permutación correspondiente, logrando una mejora local. En este caso, cuando la búsqueda que se está realizando está siendo dirigida por la vecindad IE, esta optimización no se realiza.

- Permutación de localías entre los dos partidos que juegan entre sí un par de equipos. Se evalúa el costo de todos los fixtures que se pueden obtener permutando la localía de los partidos que juegan entre sí todos los pares de equipos. Si el que obtiene menor costo es mejor que el fixture original, entonces se aplica la permutación correspondiente, logrando una mejora local. Cuando la búsqueda que se está realizando está siendo dirigida por la vecindad IL, esta optimización no se realiza.

- Inversión de Tours: Se evalúan todos los fixtures que se pueden obtener invirtiendo la localía de todos los partidos (tour) de cada uno de los equipos (por separado). Si el que obtiene menor costo es mejor que el fixture original, entonces se invierte el tour del equipo que corresponda, logrando una mejora local.

- Modificación de los HAP: Dado un fixture f, se intenta realizar algunas modificaciones pseudo-aleatorias a las localías de los partidos, para intentar maximizar (respetando las restricciones) la cantidad de partidos seguidos que los equipos juegan de visitante. La idea es tratar de aplicar "buenos patrones" de localías a algunos equipos elegidos al azar, para

tratar de disminuir las distancias que éstos deben recorrer sin cambiar el orden (definido por f) en que enfrentan a los rivales.

4.7. Intensificación

Como dijimos anteriormente, el objetivo principal de la intensificación es explotar regiones del espacio de soluciones consideradas "buenas". Nuestro algoritmo incluye dos estrategias de intensificación que se describen a continuación.

Disminución del tamaño de la Lista Tabú

Consiste simplemente en realizar la búsqueda en forma normal durante un número acotado de iteraciones, pero utilizando un valor mas pequeño para el tamaño de la lista tabú, el cual varia aleatoriamente en el rango de [4-8] (definido empíricamente) cada vez que se inicia un proceso de intensificación. Esta estrategia se utiliza cada vez que se cambia la vecindad que guía la búsqueda, re-iniciando la misma a partir de alguna de las mejores soluciones obtenidas hasta el momento.

Optimización del recorrido de algunos equipos

La idea de esta estrategia que se relaciona con el problema de QCP presentada en la sección 2.5 (pág. 22), se basa en que para el TTP, la parte más importante en la definición del tour de un equipo en particular (la secuencia de sus partidos a lo largo del torneo) es el orden en que juega sus partidos de visitante, ya que estos son los que definen el costo de ese equipo.

A partir de un fixture válido g y un subconjunto T de equipos seleccionados, se construye un fixture parcial f de la siguiente manera:

Para cada equipo $e := 1 \dots n$ **Hacer**
  Si $e \notin T$ **entonces**
    Para cada ronda $r := 1 \dots R$ **Hacer**
      Si g define que e juega de visitante en la ronda r **entonces**
        definir ese mismo partido en f
      Fin Si
    Fin para
  Fin Si
Fin para

Con esto, se obtiene un fixture parcial en donde sólo faltan definir los partidos en que los equipos seleccionados juegan de visitante. Como la construcción se hace a partir de un fixture válido, sabemos que existe al menos una forma de completarlo correctamente.

Luego, a partir del fixture parcial que se obtiene, se construyen todos los fixtures posibles respetando esta configuración inicial. La cantidad de equipos que se selecciona solo varía entre 4 y 5 (dependiendo de la cantidad de equipos) ya que una mayor selección provocaría, en general, que el algoritmo tarde demasiado, como consecuencia de lo descripto en 2.5. La idea es utilizar este algoritmo con varios subconjuntos de equipos, eligiendo preferentemente aquéllos cuyos costos asociados con sus recorridos (C_e) son los que tienen mayor valor (respecto de otros equipos), pues estos son los que más posibilidad tienen de mejorar. Pero también se trata de elegir los 2 o 3 equipos con menor costo entre todos los demás, con la idea de que estos pueden llegar a "flexibilizar" su recorrido haciéndolo mas costoso, pero a la vez podrían disminuir el costo de otros, y entonces, en suma, el costo total del fixture podría llegar a ser menor. Variantes de esta idea podrían utilizarse si por ejemplo, además de minimizar el costo total, se quisiera que los equipos tengan costos "balanceados".

En la figura 4.3, se puede ver un ejemplo de la aplicación de este algoritmo. A partir de un fixture válido para 4 equipos, se construye un fixture parcial en el cual están definidos todos los partidos que los equipos A y C juegan de visitante, y faltan definir todos los partidos en que los equipos B y D juegan de visitante.

Rda	A	B	C	D
1	B	@A	D	@C
2	C	D	@A	@B
3	@D	C	@B	A
4	@B	A	@D	C
5	D	@C	B	@A
6	@C	@D	A	B

seleccionando los equipos B y D y aplicando el algoritmo descripto, obtenemos: $\longrightarrow$

Rda	A	B	C	D
1				
2	C		@A	
3	@D	C	@B	A
4	@B	A	@D	C
5				
6	@C		A	

Figura 4.3: A partir de un fixture válido, y un subconjunto de equipos seleccionados, obtenemos un fixture parcial, donde faltan definir todos los partidos en que B y D juegan de visitante. Luego, se buscará (entre todos los posibles) el mejor fixture que incluya los partidos ya establecidos.

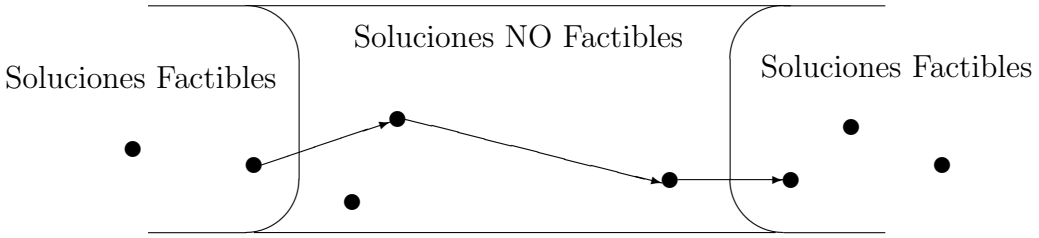
Este tipo de intensificación se realiza como paso previo a la diversificación,

que se describe en la siguiente sección.

4.8. Diversificación

La etapa de diversificación, que se realiza después de cierta cantidad de iteraciones sin mejora, consiste en realizar varios movimientos IPR consecutivos, que conducen a vecinos **inválidos**, y luego utilizar los algoritmos de optimización local descriptos en 4.6 para tratar de volver a obtener un fixture correcto. De esta forma, como se muestra en la figura 4.4 podemos llegar a otras regiones del espacio de búsqueda, atravesando regiones de soluciones no factibles, a las que de otra forma podríamos no llegar.

Figura 4.4: Proceso de diversificación: Pasaje de una solución válida a otra, a través de una región de soluciones no factibles.



Además, como se mencionó anteriormente, consideramos que el hecho de variar con cierta frecuencia la vecindad que guía la búsqueda (aunque algunas vecindades se utilicen con mucha menor frecuencia y duración que otras) provoca que el proceso de búsqueda realice indirectamente pequeñas diversificaciones y disminuya la posibilidad de quedarse atrapado en mínimos locales.

4.9. Criterio de Parada

Entre los parámetros de entrada que el algoritmo recibe, se incluyen la cantidad máxima de iteraciones que este puede realizar y la cantidad admisible de iteraciones sin mejora. Cuando se supera cualquiera de estas dos cotas, el programa termina.

4.10. Esquema del algoritmo implementado

En la figura 4.5, a modo de resumen, presentamos el esquema general del algoritmo propuesto, que denominamos $TSparaTTP$. Este algoritmo, implementa una búsqueda Tabú con las características descriptas en las secciones anteriores. A partir de una matriz D de distancias de $n \times n$ genera el mejor fixture que pudo encontrar en su búsqueda para esa matriz D. Además, el algoritmo puede recibir varios parámetros que son los que influyen en las decisiones que éste realiza durante su ejecución:

Max-iteraciones: Cantidad máxima de iteraciones que el algoritmo realiza

Max-itera-sin-mejora : Cantidad máxima de iteraciones seguidas sin mejora

Max-Tam-ListaTabu : Tamaño máximo de la lista Tabú

Cant-itera-Int : Cantidad de iteraciones durante las cuales se intensifica (reduciendo el tamaño del lista tabú)

Tam-Bloque : Cada cuantas iteraciones se debe cambiar (aleatoriamente) el tamaño de la lista Tabú

Frecuencia-Opt-Local : Cada cuantas iteraciones se realizan las optimizaciones locales

Opcionalmente, también se le puede indicar al algoritmo la forma en que se deben alternar las distintas vecindades a lo largo del proceso de búsqueda. Nosotros evaluamos distintas variantes, y finalmente utilizamos la siguiente secuencia, que se va repitiendo hasta que el algortimo termina:

$$IE \hookrightarrow IPR \hookrightarrow IR \hookrightarrow IPR \hookrightarrow IL \hookrightarrow IPR \hookrightarrow IL$$

4.11. Estructura de datos utilizada

Dado un determinado fixture, los datos que se necesitaban conocer con mayor frecuencia son los siguientes:

- Ronda en la que dos equipos se enfrentan (partido y revancha)
- Rival de un equipo en una determinada ronda
- Si un equipo juega de local o visitante en una determinada ronda

Para poder responder a estas cuestiones eficientemente, decidimos almacenar todos los datos de un fixture en dos matrices, según se describen a continuación.

Matriz de Rondas: Una matriz de $n \times n$, donde el valor de la celda $[fil, col]$ indica en que ronda se enfrenta el equipo *col* (de local) vs el equipo fil (de visitante).(Los valores de la digonal se ignoran). Es similar a la representación vista en la sección 2.3.2 (pág. 16)

Matriz de Recorrido: Una matriz de $2(n-1) \times n$, donde el valor de la celda $[r, e]$ indica el rival del equipo e en la ronda r. Un valor positivo indica que e juega de local, y uno negativo indica que e juega de visitante. Esto implica que esta matriz (MR) debe satisfacer la siguiente propiedad: $MR[r, |a|\ ] = b \iff MR[r, |b|\ ] = (-1)a$.

De esta forma, logramos responder cualquiera de los puntos mencionados en orden $O(1)$.

4.12. Complejidad del algoritmo propuesto

En base al esquema presentado en la figura 4.5, podemos ver que en cada iteración el algoritmo busca el vecino al cual moverse, y además en "algunas" rondas puede realizar los procesos de intensificación, diversificación o de optimización local. De todas estas tareas, la que mayor tiempo necesita, de acuerdo a lo presentado en las secciones anteriores, requiere O(n^4). Por lo tanto, podemos determinar que la complejidad (en el peor caso) del algoritmo final es de O(Qn^4) donde Q representa la cantidad máxima de iteraciones permitidas. Pero en la práctica, por todo lo expuesto en la sección 4.4.1, el tiempo requerido suele ser bastante menor.

```
Algoritmo TSparaTTP

   Entrada: n, Dist  (#equipos, Matriz de distancias)
   Salida:  Mejor Fixture obtenido

   Generar fixture f al azar
   mejorFixture = f
   costo_actual = CalcularCostoTTP(f, Dist)
   mejorCosto = costo_actual
   c = 0
   vecindadActual = IPR

   Mientras NO se cumpla el criterio de Parada, hacer

      c++
      segun vecindadActual, elegir el movimiento que lleve al mejor vecino de f
                            que no sea Tab, o que cumpla el criterio de aspiracion
      agregar m a la lista tabu
      f = el vecino de f que se obtiene aplicando el movimiento m

      Si (c mod Frecuencia_Opt_Local) = 0, entonces
         hacer Optimizaciones Locales sobre f

      costo_actual = CalcularCostoTTP(vec, Dist)

      Si costo_actual < mejorCosto, entonces
         mejorFixture = f
         mejorCosto = costo

      Si (c mod Tam_Bloque) = 0, entonces
         cambio el tamao de la lista tabu, eligiendolo al azar en el rango permitido

      Si hayQueCambiarDeVecindad*, entonces
         intensifico a partir de mejorFixture
         elijo otra vecindad para dirigir la busqueda

      Si hayQueDiversificar*, entonces
         intensifico a partir de mejorFixture
         diversifico
         elijo otra vecindad para dirigir la busqueda

   Fin Mientras

   Devolver mejorFixture
```

* indica que las decisiones dependen de los parámetros de entrada. A la vecindad IPR se la utiliza con mayor frecuencia y duración que el resto.

Figura 4.5: Esquema del algoritmo TSparaTTP

Capítulo 5

Resultados

El algoritmo presentado fue escrito y compilado en C++. Todas las pruebas realizadas, se hicieron en una computadora Intel Pentium III-800Mhz con el sistema operativo Windows 2000.

Para evaluar el desempeño del algoritmo se utilizaron todas las instancias que se encuentran disponibles en la página web del TTP [42], las cuales se dividen en dos grupos.

Instancias NL

Estas instancias, se corresponden con distancias reales entre los estadios de distintos equipos (ciudades) de Estados Unidos que pertenecen a la liga MLB de Baseball (Major League Baseball), en cuyas características se basó la definición del TTP. La MLB está dividida en dos partes. Una de ellas, la National League, posee 16 equipos ubicados en ambas costas de Estados Unidos y en Canada. Estas instancias se generaron tomando subconjuntos de estos 16 equipos. De esta manera, (definiendo un orden de los equipos), la instancia NLx está formada por las distancias entre los primeros x equipos. Hasta el momento, sólo se conoce el valor óptimo para $4, 6$ y 8 equipos.

Instancias Circulares

Muchos de los análisis sobre el TTP giran alrededor de los aspectos relacionados con el problema del viajante de comercio (TSP). Sin embargo, no está claro que el TTP sea fácil de resolver a pesar de que la solución para el TSP si lo sea.

Para estudiar estos aspectos, se armó la clase de instancias circulares, que se definen de la siguiente manera: Una instancia circular de n nodos o equipos (denotada $CIRCn$) tiene distancias generadas por el grafo circular de n nodos con distancias equivalentes a una unidad. Si numeramos los nodos como $0, 1, \ldots, n-1$, tenemos que en este grafo existen arcos de i a $i+1$ $(i = 0 \ldots n-1)$ y de $n-1$ a 0, todos con longitud 1. De esta forma, tenemos que la distancia de i a j $(i > j)$ es el mínimo entre $i-j$ y $j-i+n$. Para este caso, hasta el momento, tampoco nadie ha logrado obtener el valor óptimo para instancias con más de 8 equipos.

Resultados

Ejecutamos el algoritmo utilizando distintos valores para los parámetros de entrada, los cuales, en general, los aumentábamos (en forma proporcional) a medida que se incrementaba la cantidad de equipos (n). Como resumen, a continuación detallamos los rangos utilizados:

- Max-iteraciones: Se utilizaron valores en el rango de 100,000 a 1,000,000.
- Max-itera-sin-mejora: Se definió como $\frac{Max-iteraciones}{2}$.
- Max-Tam-ListaTabu: Su valor estaba en el rango $[n-3, n+3]$
- Cant-itera-Int: Entre 2,000 y 4,000
- Tam-Bloque: Entre 2,000 y 20,000 iteraciones
- Frecuencia-Opt-Local: Cada $2n$ iteraciones aproximadamente.

Como se describió en 4.2, utilizamos dos algoritmos para generar la solución inicial. Aunque en líneas generales, se obtuvieron resultados similares con ambas alternativas, en algunos casos se observó una leve mejoría al utilizar la variante que genera soluciones totalmente aleatorias, sin seguir ningún esquema establecido, por lo que decidimos utilizar esta variante para los experimentos que se detallan a continuación.

En las figuras 5.1 y 5.2 se pueden observar los resultados que obtuvimos para cada una de las instancias evaluadas. Por cada una de ellas, se informa lo siguiente:

- Problema: Nombre de la instancia
- Cota inf.: Cota inferior
- Mejor Sol. Previa: La mejor solución conocida previamente
- Resultado: Mejor solución obtenida por nuestro algoritmo
- Dif %: Diferencia entre nuestra mejor solución y la mejor conocida previamente

Como se puede ver, para todas las instancias NL, nuestro algoritmo obtiene muy buenos resultados alcanzando siempre una diferencia menor al 3 % con respecto a las mejores conocidas hasta el momento, e incluso, hemos podido igualar(NL4, NL6 y NL8) y mejorar algunas de ellas (NL12 y NL14).

En el caso de las instancias Circulares, también obtuvimos buenos resultados, aunque en este caso están en general un poco más alejados de los mejores resultados publicados previamente, especialmente para las instancias con más de 14 equipos. Dado que estas instancias tienen características "especiales" por la forma en que se construyeron, creemos que nuestro algoritmo podría obtener mejores resultados si se tratara de "aprovechar" estas características en algún paso de la búsqueda.

Problema	Cota Inferior	Mej. Sol. Previa	Mej. Sol. TSparaTTP	Dif %
NL4	8276	8276	**8276**	0.0
NL6	23916	23916	**23916**	0.0
NL8	39479	39721	**39721**	0.0
NL10	57500	61608	62561	1.5
NL12	107483	119012	**118955**	-
NL14	182797	207075	**205894**	-
NL16	248852	284235	293013	3.0

Figura 5.1: Resultados obtenidos para las instancias NL.

Problema	Cota Inferior	Mej. Sol. Previa	Mej. Sol. TSparaTTP	Dif %
CIRC4	20	20	**20**	0.0
CIRC6	64	64	**64**	0.0
CIRC8	128	132	134	1.5
CIRC10	220	266	268	0.7
CIRC12	384	448	458	2.2
CIRC14	588	712	730	2.5
CIRC16	832	984	1074	9.1
CIRC18	1188	1442	1550	7.5
CIRC20	-	1990	2086	4.8

Figura 5.2: Resultados obtenidos para las instancias Circulares. Los valores correspondientes a CIRC18 y CIRC20 se obtuvieron empezando la búsqueda a partir de buenas soluciones previamente conocidas.

Si bien es de esperar que a medida que aumentemos la cantidad de iteraciones que permitimos ejecutar a nuestro algoritmo, se puedan lograr cada vez mejores resultados, también es importante que el algoritmo obtenga

soluciones "aceptables" en poco tiempo, para cuando se requiera tener cierta interacción en el diseño del fixture, como por ejemplo en los casos donde los requerimientos cambian a último momento.(aunque no es el caso del TTP)

Para analizar este aspecto en nuestro algoritmo, volvimos a ejecutarlo para todas las instancias evaluadas, pero definiendo el parámetro de iteraciones máximas en 90,000 lo que en la práctica equivale a un tiempo de ejecución de unos pocos minutos hasta a lo sumo 35 para el peor caso. Los resultados obtenidos se muestran en la tablas de las figuras 5.3 y 5.4. Allí se puede observar que, en comparación a los resultados anteriores, y teniendo en cuenta las restricciones de tiempo, las soluciones obtenidas (en promedio) son más que aceptables.

Problema	Cota Inferior	Mej. Sol. Previa	Resultado Promedio	Tpo. Prom (minutos)	Dif %
NL4	8276	8276	**8276**	1	0.0
NL6	23916	23916	**23916**	1	0.0
NL8	39479	39721	40621	3	2.2
NL10	57500	61608	65619	5	6.5
NL12	107483	119012	124774	9	4.8
NL14	182797	207075	215279	14	3.9
NL16	248852	284235	306675	19	7.9

Figura 5.3: Resultados obtenidos para las instancias NL permitiendo como máximo 90,000 iteraciones.

Problema	Cota Inferior	Mej. Sol. Previa	Resultado Promedio	Tpo. Prom (minutos)	Dif %
CIRC4	20	20	**20**	1	0.0
CIRC6	64	64	**64**	1	0.0
CIRC8	128	132	139	3	5.3
CIRC10	220	266	280	5	5.2
CIRC12	384	448	480	8	7.1
CIRC14	588	712	756	13	6.1
CIRC16	832	984	1132	18	15.0
CIRC18	1188	1442	1613	25	11.8
CIRC20	-	1990	2224	35	11.7

Figura 5.4: Resultados obtenidos para las instancias Circulares permitiendo como máximo 90,000 iteraciones.

Capítulo 6

Conclusiones y Trabajo Futuro

En este trabajo hemos descripto las características que hacen al problema de scheduling deportivo y hemos planteado las principales dificultades que las ligas organizadoras de torneos tienen que enfrentar frecuentemente. También presentamos una breve descripción del problema de los fixtures prematuros y de la complejidad que ocasiona para la búsqueda el enorme espacio de soluciones que se debe explorar. Luego describimos el *Traveling Tournament Problem*, cuyas características lo convierten en un problema interesante para analizar y resolver. Y es por eso que en estos últimos meses varios investigadores han estado tratando de obtener mejores resultados. De hecho, mientras nosotros estábamos desarrollando esta tesis, nos enteramos (en más de una ocasión), que se habían obtenido mejores resultados para varias de las instancias evaluadas. Finalmente, presentamos un algoritmo Tabú Search para el TTP, con el que obtuvimos muy buenos resultados, entre ellos dos de las mejores soluciones (hasta el momento) para las instancias publicadas en [42].

Creemos que lo expuesto en esta tesis abre puertas para continuar la investigación en el area de planificación de scheduling deportivos. A continuación daremos algunas ideas que pueden tomarse como base para trabajos futuros:

- Intentar que los equipos tengan "recorridos" parejos o balanceados, tratando de mantener el costo total lo más bajo posible
- Agregar más restricciones al TTP y ver como estas afectan al desempeño del algoritmo. En particular, se puede probar para otros valores de U (cant. máxima permitidas de partidos seguidos de local o visitante), o bien pedir que se generen fixtures espejados.

- Combinar las ideas implementadas con Algoritmos Genéticos (Alg. Meméticos, ver [30])
- Incorporar al algoritmo otro tipo de Vecindades parecidas a IPR.
- Aplicar las ideas implementadas en el algoritmo TSparaTTP a otros problemas de scheduling deportivos.

Problemas Abiertos

Además, quedan abiertos los siguientes problemas:

- Encontrar una fórmula general para calcular cuántos fixtures distintos se pueden construir para un esquema dado. Ni siquiera se conoce para el caso más simple: un torneo Single Round Robin sin restricciones adicionales, tal como se presentó en la sección 2.3.1
- Decidir si el TTP pertenece o no a la clase de problemas NP-Completos. Si bien, en comparación con algunos problemas NP-completos conocidos, este no pareciera ser más sencillo, hasta el momento, nadie ha podido comprobar si el TTP pertenece a dicha clase o no. En la literatura encontramos dos trabajos donde se analizan problemas con algunas características en común al TTP. En [5], Costa realiza una transformación del problema de armar un fixture para la NHL (si bien este no sigue el esquema Round Robin, se deben minimizar las distancias que los equipos deben recorrer) a una variante del Open Shop Scheduling Problem (OSSP), pero no queda claro que complejidad tiene esa variante del OSSP. Por otro lado, en [7], Czech describe el "Delivery Problem"(DP)[1] que es muy similar al problema de determinar el tour óptimo para un sólo equipo en el TTP, y presenta una reducción del DP al "Set Partitioning Problem", que sí es NP-completo, pero esto, evidentemente, aunque nos puede dar algunas pistas para analizar el problema, tampoco nos garantiza nada sobre la NP-completitud del TTP ni del DP, (pues para eso, la reducción debería hacerse en el sentido inverso).

[1]Se tiene un depósito central y un conjunto de clientes al que hay que visitar. No se pueden visitar mas de k de clientes seguidos (ej. $k = 3$). El objetivo es minimizar las distancias recorridas.

Apéndice A

Fixtures obtenidos

Aquí se encuentran algunos de los fixtures obtenidos por nuestro algoritmo. Estos se corresponden con las mejores soluciones conocidas (al momento de la presentación de este trabajo), para algunas de las instancias propuestas en [42].

```
==============================
INSTANCIA NL8: -->  39721  <--
==============================
```

ATL	NYM	PHI	MON	FLA	PIT	CIN	CHI
FLA	@PHI	NYM	PIT	@ATL	@MON	CHI	@CIN
CIN	PIT	MON	@PHI	CHI	@NYM	@ATL	@FLA
CHI	@MON	PIT	NYM	CIN	@PHI	@FLA	@ATL
@MON	PHI	@NYM	ATL	@CHI	CIN	@PIT	FLA
@NYM	ATL	CHI	@PIT	@CIN	MON	FLA	@PHI
@PHI	CHI	ATL	@CIN	@PIT	FLA	MON	@NYM
NYM	@ATL	@FLA	@CHI	PHI	@CIN	PIT	MON
PHI	@FLA	@ATL	CIN	NYM	@CHI	@MON	PIT
@CHI	CIN	@PIT	FLA	@MON	PHI	@NYM	ATL
@PIT	FLA	CIN	CHI	@NYM	ATL	@PHI	@MON
@CIN	MON	FLA	@NYM	@PHI	CHI	ATL	@PIT
PIT	@CHI	@CIN	@FLA	MON	@ATL	PHI	NYM
MON	@CIN	@CHI	@ATL	PIT	@FLA	NYM	PHI
@FLA	@PIT	@MON	PHI	ATL	NYM	@CHI	CIN

```
================================
INSTANCIA NL12: -->  118955  <--
================================
```

ATL	NYM	PHI	MON	FLA	PIT	CIN	CHI	STL	MIL	HOU	COL
NYM	@ATL	@FLA	@HOU	PHI	@COL	MIL	@STL	CHI	@CIN	MON	PIT
PHI	@CIN	@ATL	@FLA	MON	@HOU	NYM	@COL	MIL	@STL	PIT	CHI
@CIN	@STL	@MON	PHI	PIT	@FLA	ATL	@HOU	NYM	@COL	CHI	MIL
@PIT	CIN	FLA	COL	@PHI	ATL	@NYM	STL	@CHI	HOU	@MIL	@MON
@FLA	COL	CIN	@PIT	ATL	MON	@PHI	HOU	@MIL	STL	@CHI	@NYM
MIL	@MON	COL	NYM	@CIN	CHI	FLA	@PIT	HOU	@ATL	@STL	@PHI
@HOU	@CHI	@MIL	PIT	@COL	@MON	STL	NYM	@CIN	PHI	ATL	FLA
@COL	@MIL	@CHI	STL	@HOU	CIN	@PIT	PHI	@MON	NYM	FLA	ATL
MON	HOU	@CIN	@ATL	COL	STL	PHI	@MIL	@PIT	CHI	@NYM	@FLA
COL	MIL	HOU	@STL	CHI	@CIN	PIT	@FLA	MON	@NYM	@PHI	@ATL
CHI	@PHI	NYM	@COL	MIL	@STL	HOU	@ATL	PIT	@FLA	@CIN	MON
@MON	FLA	@PIT	ATL	@NYM	PHI	@CHI	CIN	COL	@HOU	MIL	@STL
@NYM	ATL	STL	FLA	@MON	HOU	@MIL	COL	@PHI	CIN	@PIT	@CHI
@PHI	STL	ATL	HOU	@PIT	FLA	CHI	@CIN	@NYM	COL	@MON	@MIL
PIT	@FLA	@STL	@CIN	NYM	@ATL	MON	MIL	PHI	@CHI	@COL	HOU
FLA	@HOU	@COL	@MIL	@ATL	@CHI	@STL	PIT	CIN	MON	NYM	PHI
@STL	@COL	@HOU	@CHI	CIN	@MIL	@FLA	MON	ATL	PIT	PHI	NYM
HOU	CHI	MIL	CIN	STL	COL	@MON	@NYM	@FLA	@PHI	@ATL	@PIT
STL	@PIT	CHI	MIL	HOU	NYM	COL	@PHI	@ATL	@MON	@FLA	@CIN
CIN	PHI	@NYM	CHI	@STL	MIL	@ATL	@MON	FLA	@PIT	COL	@HOU
@CHI	MON	PIT	@NYM	@MIL	@PHI	@HOU	ATL	@COL	FLA	CIN	STL
@MIL	PIT	MON	@PHI	@CHI	@NYM	@COL	FLA	@HOU	ATL	STL	CIN

```
================================
INSTANCIA  NL14: -->  205894  <--
================================
```

ATL	NYM	PHI	MON	FLA	PIT	CIN	CHI	STL	MIL	HOU	COL	SF_	SD_
@NYM	ATL	@MIL	CIN	CHI	@COL	@MON	@FLA	@HOU	PHI	STL	PIT	@SD_	SF_
FLA	CIN	@STL	@MIL	@ATL	@SF_	@NYM	@HOU	PHI	MON	CHI	@SD_	PIT	COL
MON	@FLA	SF_	@ATL	NYM	@SD_	STL	MIL	@CIN	@CHI	COL	@HOU	@PHI	PIT
NYM	@ATL	STL	@FLA	MON	SF_	COL	SD_	@PHI	HOU	@MIL	@CIN	@PIT	@CHI
@CHI	SD_	@MON	PHI	@COL	STL	HOU	ATL	@PIT	SF_	@CIN	FLA	@MIL	@NYM
@MIL	PIT	CIN	SD_	@SF_	@NYM	@PHI	@COL	HOU	ATL	@STL	CHI	FLA	@MON
@STL	@PHI	NYM	PIT	@SD_	@MON	@HOU	@SF_	ATL	@COL	CIN	MIL	CHI	FLA
CIN	HOU	MON	@PHI	STL	COL	@ATL	@SD_	@FLA	@SF_	@NYM	@PIT	MIL	CHI
@FLA	STL	@PIT	HOU	ATL	PHI	SF_	COL	@NYM	@SD_	@MON	@CHI	@CIN	MIL
PIT	SF_	@CIN	STL	SD_	@ATL	PHI	HOU	@MON	COL	@CHI	@MIL	@NYM	@FLA
MIL	@COL	SD_	SF_	@HOU	@CIN	PIT	@STL	CHI	@ATL	FLA	NYM	@MON	@PHI
SD_	@SF_	PIT	@CHI	@STL	@PHI	@COL	MON	FLA	@HOU	MIL	CIN	NYM	@ATL
@COL	@SD_	MIL	@HOU	@PIT	FLA	@SF_	STL	@CHI	@PHI	MON	ATL	CIN	NYM
@SF_	PHI	@NYM	@PIT	HOU	MON	@SD_	@MIL	COL	CHI	@FLA	@STL	ATL	CIN
@SD_	CHI	HOU	COL	SF_	@STL	MIL	@NYM	PIT	@CIN	@PHI	@MON	@FLA	ATL
SF_	COL	CHI	@STL	@CIN	HOU	FLA	@PHI	MON	SD_	@PIT	@NYM	@ATL	@MIL
@PHI	@MIL	ATL	@CIN	@CHI	SD_	MON	FLA	@COL	NYM	SF_	STL	@HOU	@PIT
@MON	@CHI	@COL	ATL	@MIL	CIN	@PIT	NYM	@SF_	FLA	SD_	PHI	STL	@HOU
COL	@HOU	@SF_	MIL	CIN	@CHI	@FLA	PIT	@SD_	@MON	NYM	@ATL	PHI	STL
@PIT	MIL	@SD_	CHI	COL	ATL	@STL	@MON	CIN	@NYM	@SF_	@FLA	HOU	PHI
@CIN	MON	COL	@NYM	MIL	CHI	ATL	@PIT	SF_	@FLA	@SD_	@PHI	@STL	HOU
PHI	@PIT	@ATL	FLA	@MON	NYM	SD_	SF_	@MIL	STL	@COL	HOU	@CHI	@CIN
@HOU	@MON	FLA	NYM	@PHI	@MIL	@CHI	CIN	SD_	PIT	ATL	SF_	@COL	@STL
STL	FLA	@CHI	@SF_	@NYM	@HOU	@MIL	PHI	@ATL	CIN	PIT	SD_	MON	@COL
CHI	@CIN	@HOU	@SD_	PIT	@FLA	NYM	@ATL	MIL	@STL	PHI	@SF_	COL	MON
HOU	@STL	@FLA	@COL	PHI	MIL	CHI	@CIN	NYM	@PIT	@ATL	MON	SD_	@SF_

Bibliografía

[1] *Anderson L.*, "Completing partial latin squares", Mathematisk Fysiske Meddelelser 41 (1985), 23-69.

[2] *Berge C.*, "Graphs", North-Holland, 1985.

[3] *Bogomolny Alexander*, "Latin Squares", en http://www.cut-the-knot.com/arithmetic/latin_intro.shtml.

[4] *Colbourn C.J.*, "Embedding partial Steiner triple systems is NP-complete", Journal of Combinatorial Theory, series A35 (1983), 100-105.

[5] *Costa D.*, "An evolutionary tabu search Algorithm and the NHL scheduling Problem", INFOR Vol. 33, No.3 (1995), 161-178.

[6] *Crauwels H. & Oudheusden Van D.*, "A Generate-and-Test Heuristic Inspired by Ant Colony Optimization for the Traveling Tournament Problem", Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling, 2002.

[7] *Czech Z.J. & Marzec E.*, "Heuristic algorithms for solving the set partitioning problem", en Intelligent Information Systems Workshop, Zakopane, Poland, (1997).

[8] *De Werra*, "Geography, Games and Graphs", Discrete Applied Mathematics 2 (1980) 327-337.

[9] *De Werra*, "Scheduling in sports", in P. Hansen (Ed.), Studies on Graphs and Discrete Programming, North Holland (1981), 381-395.

[10] *De Werra*, "Some models of graphs for scheduling sports competitions", Discrete Applied Mathematics 21 (1988), 47-65.

[11] *Dinitz J. & Stinson D.*, "A hill-climbing algorithm for the construction of one-factorizations and Room squares", SIAM J. Algebraic and Disc. Methods, Vol. 8, No.3 (1987), 430-438.

[12] *Dinitz J., Garnick D. & McKay B.*, "There are 526,915,620 nonisomorphic one-factorizations of K12", Journal of Combinatorial Designs Vol 2, No.4 (1994), 273-285.

[13] *Dinitz J. & D. Froncek*, "Scheduling the XFL", Congressus Numerantium 147 (2000), 5-15.

[14] *Dubuc E.*, "Problema de los fixtures condicionados", Depto. de Matemática, FCEyN, UBA (1995), manuscrito.

[15] *Easton K., Nemhauser G. & Trick M.*, "Solving the Traveling Tournament Problem: A Combined Integer Programming and Constraint Programming Approach", Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling, 2002.

[16] *Fleurent C. & Ferland J.A.*, "Allocating games for the NHL using integer programming", Operations Research 41-4 (1993), 649-654.

[17] *Glover F*, "Tabu Search - Part 1", ORSA Journal on Computing 1,3 (1989), 190-206.

[18] *Glover F. & Laguna M.*, "Tabu Search", in Reeves, C.R. (Ed.), Modern Heuristics Techniques for Combinatorial Problems, Blackwell Scientific Publishing, Oxford, (1993).

[19] *Gomes C. & Shmoys D.*, "Completing Quasigroups or Latin Squares: A Structured Graph Coloring Problem", In Proceedings of the Comp. Symposium on Graph Coloring and Extensions, Ithaca, NY, 2002.

[20] *Gomes C. & Selman B.*, "Problem Structure in the Presence of Perturbations", In Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97), New Providence, RI.

[21] *Gomes C.*, "The Quasigroup Completion Problem",
en http://www.cs.cornell.edu/Info/People/gomes/QUASIdemo.html.

[22] *Hamiez Jean P. & Hao Jin-Kao*, "Solving the Sports League Scheduling Problem with Tabu Search", ECAI-2000, Workshop on Local Search for Planning and Scheduling, Berlin.

[23] *Harary F.*, "Graph Theory", Addison Wesley, 1969.

[24] *Henz M.*, "Constraint-based Round Robin Tournament Planning", in D. De Schreye, editor, Proceedings of the International Conference on Logic Programming, Las Cruces, New Mexico (1999), 545-557.

[25] *Henz M.*, "Scheduling a Major Basketball Conference-Revisited", Operations Research, 49(1) (2001).

[26] *Henz, Martin*, "Friar Tuck - A Constrained-Based Tournament Scheduling Tool", IEEE INTELLIGENT SYSTEMS, 2000 (Software disponible en http://www.comp.nus.edu.sg/∼henz/projects/FriarTuck/).

[27] *Henz M., Müller T., Thiel S. & Van Brandenburg M.*, "Global Constraints for Round Robin Tournament Scheduling", disponible en: http://www.comp.nus.edu.sg/∼henz/drafts/one-factor.ps.

[28] *Hertz A., Tailard E. & De Werra D.*, "A Tutorial on Tabu Search", Proc. of Giornate di Lavoro AIRO'95 (Enterprise Systems: Management of Technological and Organizational Changes), 13-24.

[29] *Leonard J.*, "Interactive game scheduling with genetic algotithms", Master Thesis, Department of Computer Sciences, Royal Melbourne Institute of Technology University, Australia (1998).

[30] *Moscato P.*, "Memetic Algorithms' Home Page", en http://www.densis.fee.unicamp.br/˜moscato/memetic_home.html.

[31] *Onno Waalewijn*, "Java applets for TSP, TTP & other related problems", en http://home.planet.nl/˜onno.waalewijn/.

[32] *Peterson I.*, "MathTrek: Completing Latin Squares", en http://www.maa.org/mathland/mathtrek_5_8_00.html.

[33] *Rosa, A. & W.D. Wallis*, "Premature sets of 1-factors or how not to schedule round robin tournaments", Discrete Applied Mathematics 4 (1992), 291-297.

[34] *Rottembourg B., Laburthe F. & Benoist T.*, "Lagrange Relaxation and Constraint Programming Collaborative schemes for Traveling Tournament Problems", CPAI-OR, Wye College, IK (2001), 15-26.

[35] *Schaerf, A.*, "Scheduling Sport Tournaments Using Constraint Logic Programming", Constraints 4 (1999), 43-65.

[36] *Schreuder, J.A.M.*, "Constructing timetables for sport competitions", Mathematical Programming Study 13 (1980), 58-67.

[37] *Schreuder J.A.M.*, "Construction Aspects with ILP models for scheduling KNVB Fixture Lists", manuscrito en preparación (2002).

[38] *Sloane, N.J.A.*, "Sequences A036981, A000438, A065594, A000474, A064120, A003191 & others.", in The On-Line Encyclopedia of Integer Sequences: http://www.research.att.com/~njas/sequences/.

[39] *Trick, M.A., Nemhauser, G.L.*, "Scheduling a Major College Basketball Conference", Operations Research 46(1) (1998), 1-8.

[40] *Trick, M.A*, "A Schedule then Break Aproach to Sports Scheduling", aceptado en PATAT 2000 (Konstanz).

[41] *Trick M.A., Easton K., Nemhauser G.*, "The Traveling Tournament Problem description and benchmarks",
disponible en: http://mat.gsia.cmu.edu/TOURN/ttp.ps.

[42] *Trick M.*, "Challenge Traveling Tournament Instances",
en http://mat.gsia.cmu.edu/TOURN.

[43] *Wallis W.D.*, "One-Factorizations", Kluwer Academic Publishers, Dordrecht (1997).

[44] *Weisstein W.*, "Information about Latin Squares",
en http://mathworld.wolfram.com/LatinSquare.html.

[45] *Terril, B.J., Willis, R.J.*, "Scheduling the Australian State Cricket Season Using Simulated Annealing", Journal of the Operational Research Society 45 / 3 (1994), 276-280.

[46] *Wright, M.*, "Timetabling county cricket fixtures using a form of tabu search", Journal of the Operational Research Society 45 (1994), 758-770.

[47] *Van Brandenburg M.*, "Intermural Tournament planning", Technical Report, National University of Singapore, Noviembre 2000.

[48] *Von Zuben Fernando & Concilio Ricardo*, "Evolutionary Design Of Schedules In Championships With Compact Genetic Codification and Local Search", Workshop on Memetic Algorithms, GECCO 2000.