UN ALGORITMO TABU SEARCH PARA EL TRAVELING TOURNAMENT PROBLEM

Andrés Cardemil[1]
Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Buenos Aires, Argentina
acardemil@dc.uba.ar

Guillermo Durán[2]
Departamento de Ingeniería Industrial
Facultad de Ciencias Físicas y Matemáticas
Universidad de Chile
Santiago, Chile
gduran@dii.uchile.cl

Resumen

La planificación de fixtures deportivos es una tarea muy compleja que las ligas organizadoras de torneos enfrentan frecuentemente. En este trabajo repasamos algunas de las características más importantes de los problemas que se presentan al intentar diseñar fixtures deportivos y proponemos un algoritmo Tabu Search para el denominado Traveling Tournament Problem (TTP), con el que obtuvimos muy buenos resultados.

1. Introducción

Las ligas deportivas suelen enfrentarse con muchas 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

[1] Partially supported by UBACyT Grant X184.

[2] Partially supported by FONDECyT Grant 1030498 and Millennium Science Nucleus "Complex Engineering Systems", Chile and "International Scientific Cooperation Program CONICyT/SETCIP"', Chile-Argentina.

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, las implicancias no sólo deportivas, sino también económicas que el diseño de fixtures ofrece, han permitido que en varios países se observe con gran interés este tipo de problemas.

Desde un punto de vista más teórico, la enorme cantidad de soluciones factibles que pueden generarse convierte a estos problemas en casos dignos de estudio en el campo de la Optimización Combinatoria. Incluso, en muchos casos, cuando se deben satisfacer determinados requisitos o minimizar algún aspecto en particular, el problema de confección de mixtures deportivos pertenece a la clase de problemas NP-completos. Puede verse [18] para más información sobre esto último.

En la sección 2 describimos algunas características del armado de fixtures deportivos así como los requisitos y condiciones que más frecuentemente se deben satisfacer. En la sección 3 presentamos el *Traveling Tournament Problem* (TTP) que fue propuesto por K. Easton, G. Nemhauser y M. Trick en [8] con el objetivo de poder comparar distintas técnicas de resolución de este tipo de problemas. Luego, en la sección 4, describimos brevemente la metaheurística Tabu Search, la cual utilizamos para desarrollar el algoritmo que presentamos en la sección 5 para resolver el TTP, con el que se han obtenido muy buenos resultados. A continuación, en la sección 6, se detallan los resultados obtenidos con dicho algoritmo, y en la última sección se describen algunas conclusiones y posible trabajo futuro.

2. Diseño de Fixtures Deportivos

En una competencia deportiva, n equipos deben jugar entre sí (una 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 *Double Round Robin* (DRR), o sea, con 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. Diremos que el HAP[3] 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 jugarán de local, el tercero de visitante, etc. A grandes rasgos, podemos clasificar los mixtures en dos grandes grupos. Por un lado, tenemos aquellos en los que todos los partidos se distribuyen de forma tal de minimizar la

[3] HAP: Home Away Pattern, siguiendo la notación empleada en [5,23] y otros.

cantidad de fechas o rondas requeridas (torneos clásicos de fútbol). Por otro lado, están aquellos en los que los partidos pueden distribuirse libremente a lo largo de todas las rondas que se deseen, al estilo de las ligas deportivas NBA o NHL en EE.UU. 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 n sea impar.

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 descritos en [3,7,13,21,23]. Entre las técnicas utilizadas, podemos destacar los trabajos realizados en base a Programación Entera [9,23,24], *Constraint Programming* [12,13,14,18] y métodos heurísticos [3,21,25]. También existen varias publicaciones sobre el *Traveling Tournament Problem*, las cuales se mencionan en la sección 3.

2.1 Requisitos y condiciones habituales

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 los requisitos más frecuentes que los organizadores deben considerar son que todos los equipos jueguen la misma cantidad de partidos de local y de visitante, que ningún equipo puede jugar más de una cierta cantidad de partidos seguidos de local o de visitante, que dos equipos a y b deben tener HAP complementarios (es decir, que cuando a juega de local, b debe hacerlo de visitante y viceversa), que un cierto equipo e debe jugar obligatoriamente de visitante (o de local) en una ronda o fecha determinada, que dos equipos a y b no pueden enfrentarse en determinadas rondas, que se minimicen las distancias que los equipos recorren a lo largo del torneo, etc. 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.

3. Traveling Tournament Problem

En [8], K. Easton, G. Nemhauser y M. Trick propusieron un nuevo problema, al que denominaron *Traveling Tournament Problem* (TTP) el cual abstrae algunos aspectos claves de la confección de fixtures combinando condiciones en el HAP y el objetivo de minimizar distancias.

Dada una matriz de distancias *DIST* de tamaño $n \times n$, el TTP consiste en obtener un fixture *Double Round Robin* de $2 \times (n-1)$ rondas para n equipos con las siguientes características:

- Objetivo: Minimizar la distancia total recorrida por los equipos a lo largo de todo el torneo

- Ningún equipo puede jugar más de tres 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)

Ronda	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 1. Fixture válido para el TTP de *4* equipos (@A debe leerse: "A es local")

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. Este problema está muy lejos de ser sencillo, e incluso, para instancias pequeñas (*n=8,10*), todavía no se ha logrado obtener la solución óptima. Para más detalles sobre el TTP, o para ver las instancias y sus resultados conocidos, puede verse un sitio web en [22]. Allí figuran algunos de los resultados que se obtuvieron a lo largo del desarrollo de este trabajo, los cuales hacia fines del 2002 (publicadas en su momento en [2]) superaban en algunas instancias a todas las soluciones previamente conocidas, y que luego fueron mejoradas por distintos investigadores.

Se han desarrollado y utilizado diversas técnicas para intentar resolver el problema. Easton, Nemhauser y Trick [8] intentaron resolver el problema usando *Constraint Programming* y Programación Entera. Benoist, Laburthe y Rottembourgh [17] presentaron un enfoque mixto que combina *Constraint Programming* y Relajación Lagrangiana. Crauwels y Oudheusden [4] probaron utilizando la metaheurística conocida como Colonia de Hormigas. Según el sitio web del *TTP* y lo publicado por Leong en [15], Xingwen Zhang obtuvo algunos resultados interesantes aplicando una combinación de *Constraint Programming*, *Simulated Annealing* y técnicas de *Hill-Climbing*. Anagnostopoulos y otros [1] obtuvieron excelentes resultados utilizando *Simulated Annealing*. Shen y Hantao Zhang en [19] proponen una especie de nueva metaheurística que aplicaron al TTP consiguiendo también muy buenos resultados. Existe también una variante

espejada del TTP, la cual hasta el momento sólo ha sido estudiada por Ribeiro y Urrutia [16], quienes también obtuvieron muy buenas soluciones, mejorando incluso algunas de la versión no espejada.

4. Tabu Search

Tabu Search (TS) [10,11] es una metaheurística que fue propuesta a fines de los años 80 por F. Glover. 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, cuyo objetivo es cruzar regiones de optimalidad local. 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. Es una metodología para resolución de problemas muy flexible y eficaz, que ha sido utilizada con muy buenos resultados en diversos problemas clásicos de Optimización Combinatoria y problemas reales.

4.1 Vecindario y Lista Tabú

Sea S el conjunto de soluciones del problema a tratar y sea $f : S \rightarrow R$ una función de costo sobre S que se intenta optimizar. Introducimos, entonces, el concepto de vecindario $N(s)$ para cada $s \in S$. Definimos a $N(s)$ como 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.

Otro elemento fundamental de TS 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. Para evitar "perderse" soluciones buenas, se introduce el concepto de criterio de aspiración, cuyo objetivo es permitir violar las restricciones impuestas por la LT. 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.

4.2 Intensificación y Diversificación

Otros elementos muy importantes del TS son las estrategias de intensificación y diversificación. 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. 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 más 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.

4.3 Esquema básico de Tabú Search

A continuación, describimos el esquema general de TS (para el caso de minimización), que se puede observar en la figura 2.

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.

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 2. Esquema básico de Tabú Search para un problema de minimización.

5. Nuestro Algoritmo para el TTP

Presentamos aquí un algoritmo *Tabu Search* desarrollado para resolver el *Traveling Tournament Problem*. Se detallarán los distintos elementos que lo componen.

5.1 Espacio de Soluciones

Notaremos como $\boldsymbol{F}$ al conjunto de soluciones factibles, que representaremos mediante matrices. Dado que el TTP es un torneo DRR, tenemos que la cantidad R de rondas del torneo es $2\ x\ (n\text{-}1)$ y que la celda M_{re} toma valores enteros con su módulo entre 1 y n. Asumiremos que un valor positivo en la celda M_{re} indica que el equipo e juega de local en la ronda r contra el equipo M_{re}, mientras que un valor negativo (representado por @ en las figuras) indica que dicho equipo juega de visitante en la ronda r con el equipo $-M_{re}$. Entonces, diremos que una solución factible para el TTP con n equipos es cualquier matriz M de dimensión $R\ x\ n$ que satisfaga las siguientes restricciones:

→ Restricciones generales para un DRR (no espejado):

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

$$[\ (i \neq j) \Rightarrow |M_{r,i}| \neq |M_{r,j}|\ ]\ \ \ (i, j = 1...n) \tag{2}$$

$$[M_{i,r} = j \Leftrightarrow M_{j,r} = -i]\ \ \ \forall r : 1 \leq r \leq R,\ \ (i, j = 1...n) \tag{3}$$

→ Restricciones particulares para TTP:

$$|M_{r,e}|\ \neq\ |M_{r+1,e}|\ \ (e = 1...n;\ \ r = 1...R-1) \tag{4}$$

$$[M_{r,e} > 0 \wedge M_{r+3,e} > 0] \Rightarrow [M_{r+1,e} < 0 \vee M_{r+2,e} < 0]\ \ (e = 1...n;\ \ r = 1...R-3) \tag{5}$$

$$[M_{r,e} < 0 \wedge M_{r+3,e} < 0] \Rightarrow [M_{r+1,e} > 0 \vee M_{r+2,e} > 0]\ \ (e = 1...n;\ \ r = 1...R-3) \tag{6}$$

(1) Todo equipo juega dos veces (partido y revancha) contra todos los demás.

(2) En cada ronda, todos juegan contra un equipo distinto.

(3) Si A juega de local contra B, entonces B juega de visitante contra A.

(4) Ningún par de equipos juega entre sí en dos rondas seguidas.

(5) Ningún equipo juega más de *3* partidos seguidos de local.

(6) Ningún equipo juega más de *3* partidos seguidos de visitante.

5.2 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. Ésta es una variante del propuesto en [20] para generar fixtures de una sola vuelta. Por otro lado, el segundo algoritmo que se basa en el propuesto por Dinitz y Stinson en [6] genera mixtures aleatorios sin seguir ningún patrón en especial. 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 descrito en la sección 5.7) 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 un fixture correcto.

5.3 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 $DIST_{ij}$ 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 $M_{R+1,k} > 0$ *(*$\forall k$*:* $1,\ldots,n$*)*. Luego, definimos a T_{kr} 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 utilizada 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 $f:\boldsymbol{F}\rightarrow\boldsymbol{N}$ 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 según el fixture *x*. El valor de C_e ($e=1...n$) se calcula de esta forma:

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

5.4 Vecindarios

A diferencia del esquema clásico de TS, en lugar de definir un solo 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, aunque algunos se utilizan durante mayor tiempo que otros (especialmente el *IPR*[4], que denominamos *vecindario principal*). Aplicamos esta idea no sólo con el objetivo de disminuir 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 cuáles 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 $\boldsymbol{F}\subset\boldsymbol{S}$. Pero para todos 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 cuando no estamos usando diversificación, sólo se estarán explorando regiones factibles. Pero en las etapas de diversificación, el algoritmo sí 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.

5.4.1. 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 r_1], [e_1 vs e_4 en r_2], [e_2 vs e_4 en r_1] y [e_2 vs e_3 en r_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:

- $y.rival(e_1, r_1) = y.rival(e_2, r_2) = e_4 = x.rival(e_1, r_2) = |M^x_{r2,e1}|$
- $y.rival(e_1, r_2) = y.rival(e_2, r_1) = e_3 = x.rival(e_1, r_1) = |M^x_{r1,e1}|$
- el resto de los partidos se mantiene igual

Aunque se puede observar fácilmente que al aplicar un movimiento *IPR* sobre un fixture cualquiera, el resultante siempre será un fixture que cumpla el esquema DRR; es importante destacar que esto no siempre es 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 4, 5 y 6 en la sección 5.1.

Es importante mencionar que al realizar un movimiento *IPR* se consideran todas las variantes posibles respecto de la localía de los cuatro partidos que se alteran originalmente (2^4), seleccionando entre las que resulten factibles aquella con la que se obtiene el mejor resultado. También cabe destacar que 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	.	.	.	.	.	.

↔

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 3. 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.

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

5.4.3 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. Es decir, que simplemente se permutan dos columnas de la matriz M^x asociada al fixture *x*. Está 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.

5.4.4 Intercambio de localías - IL

La vecindad *IL(x)* de un fixture x^{es}, 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 fixture, e_1 juega contra e_2 de visitante en la ronda r_1 y de local en r_2. Está claro que el fixture resultante luego de una permutación de este tipo sólo puede llegar a violar las restricciones 5 ó 6 pues la "estructura" de la matriz se mantiene igual.

5.5 Manejo de la Lista Tabú

La Lista Tabú registra cuáles son los movimientos considerados tabú. Según cual sea la vecindad que se está 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 está 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).

5.6 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 todos los posibles movimientos, el algoritmo inmediatamente cambia la dirección de la búsqueda seleccionando otra vecindad.

5.7 Optimizaciones locales

Sumado al proceso de búsqueda tabú que realiza la exploración del espacio de soluciones utilizando como guía (alternadamente) las distintas vecindades anteriormente descritas, el algoritmo cada cierta cantidad de iteraciones (que se definen por parámetro) intenta efectuar algunas pequeñas optimizaciones locales. Para esto, se evalúa el costo de todos los fixtures válidos para el *TTP* que se obtienen mediante las operaciones que se describen a continuación, y si se obtienen mejoras se aplica la operación sobre la solución actual y se repite el proceso hasta que no se obtengan más mejoras. Estas operaciones se van aplicando en el mismo orden en el que aquí se mencionan, a saber:

- Permutación de *2* y *3* rondas: similar a lo definido para la vecindad *IR*.
- Permutación de *2* y *3* equipos: similar a lo definido para la vecindad *IE*.
- Permutación de localías: (entre los dos partidos que juegan entre sí un par de equipos) similar a lo definido para la vecindad *IL*.
- Inversión de *Tours*: se invierte la localía de todos los partidos (tour) de cada uno de los equipos (por separado).
- Modificación de los HAP: Dado un fixture *f*, se intentan realizar algunas modificaciones seudoaleatorias 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.

5.8 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.

5.8.1 Disminución del tamaño de la Lista Tabú

Esta estrategia 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 varía aleatoriamente en el rango *(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, reiniciando la misma a partir de alguna de las mejores soluciones obtenidas hasta ese momento.

5.8.2 Optimización del recorrido de algunos equipos

La idea de esta estrategia 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…n,* **Hacer**

Si { e $\notin$ T } **entonces**

Para cada ronda *r:=1…R*, **Hacer**

Si *g* define que *e* juega de visitante en la ronda *r*, **entonces**

definir ese mismo partido en el fixture parcial.

Fin Si

Fin Para

Fin Si

Fin Para

Con esto, se obtiene un fixture parcial en donde sólo falta definir los partidos en que los equipos seleccionados juegan de visitantes. 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 sólo varía entre *4* y *5* (dependiendo de la cantidad de equipos), ya que una mayor selección provocaría, en general, que el algoritmo tardase demasiado. La idea es utilizar este algoritmo con varios subconjuntos de equipos, eligiendo preferentemente aquellos 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* ó *3* equipos con menor costo entre todos los demás, con la idea de que estos pueden llegar a "flexibilizar" su recorrido haciéndolo más 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, se puede ver un ejemplo de la aplicación de este algoritmo.

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 descrito, obtenemos: →

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. 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.

5.9 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 **no factibles**, y luego utilizar los algoritmos de optimización local descritos en la sección 5.7 para tratar de volver a obtener un fixture correcto. De esta forma, como se muestra en la figura 5, podemos llegar a otras regiones del espacio de búsqueda, atravesando regiones de soluciones no factibles, a las que de otra forma no podríamos llegar.

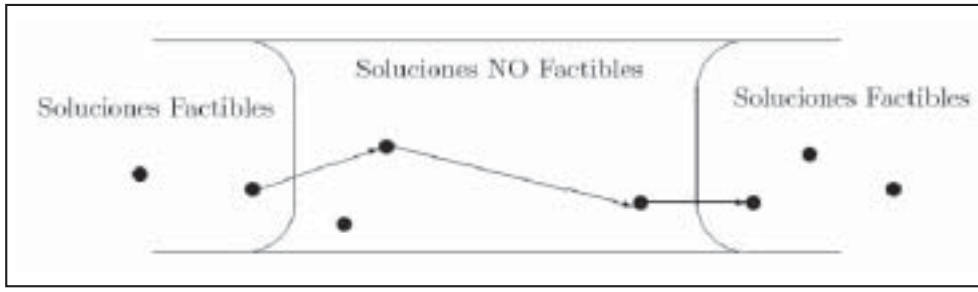

Figura 5. Proceso de diversificación: pasaje de una solución factible a otra, a través de una región de soluciones no factibles.

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.

5.10 Criterio de Parada

Entre los parámetros de entrada que el algoritmo recibe, se incluye 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.

5.11 Esquema del algoritmo implementado

En la figura 6, a modo de resumen, presentamos el esquema general del algoritmo propuesto, que denominamos *TSparaTTP*. Este algoritmo implementa una búsqueda Tabú con las características descritas en las secciones anteriores. A partir de una matriz *DIST* de distancias de tamaño *n x n* genera el mejor fixture que pudo encontrar en su búsqueda para esa matriz *DIST*. Además, el algoritmo puede recibir varios parámetros que son los que influyen en las decisiones que este realiza durante su ejecución:

- **Max-iteraciones:** Cantidad máxima de iteraciones que el algoritmo realiza.
- **Max-iteraciones-sin-mejora:** Cantidad máxima de iteraciones seguidas sin mejora.
- **Max-Tam-ListaTabu:** Tamaño máximo de la Lista Tabú.
- **Cantidad-iteraciones-Intensificación:** Cantidad de iteraciones durante las cuales se intensifica (reduciendo el tamaño de la Lista Tabú).
- **Tam-Bloque:** Cada cuántas iteraciones se debe cambiar (aleatoriamente) el tamaño de la Lista Tabú.
- **Frecuencia-Opt-Local:** Cada cuántas 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 algoritmo termina:

$$IE \rightarrow IPR \rightarrow IR \rightarrow IPR \rightarrow IL \rightarrow IPR \rightarrow IL$$

6. Resultados

El algoritmo presentado fue escrito y compilado en C++. Todas las pruebas realizadas, se hicieron en una computadora Intel Pentium III-800Mhz con 256MB de memoria RAM.

Para evaluar el desempeño del algoritmo se utilizaron todas las instancias que se encuentran disponibles en la página web del TTP [22], las cuales se dividen en dos grupos.

6.1 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 (NL), posee *16* equipos ubicados en ambas costas de Estados Unidos y en Canadá. 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* y *6* equipos.

6.2 Instancias Circulares

Muchos de los análisis sobre el TTP giran alrededor de los aspectos relacionados con el famoso 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 sí 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,..., n-1*, tenemos que en este grafo existen arcos de i a $i+1$ ($i=0...n-1$) y de $n-1$ a 0, todos con longitud unitaria. 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.

```
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 Tabu, 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
         vecindadActual = Elegir nueva vecindad para dirigir la busqueda

      Si hayQueDiversificar*, entonces
         intensifico a partir de mejorFixture
         diversifico
         vecindadActual = Elegir nueva 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 6. Esquema del Algoritmo *TSparaTTP.*

6.3 Resultados

Ejecutamos el algoritmo utilizando distintos valores para los parámetros de entrada. A continuación detallamos los valores utilizados:

- Max-iteraciones: Se utilizaron valores en el rango de 100.000 a 5.000.000.
- Max-iteraciones-sin-mejora: Se definió como: $\frac{Max\ -\ Iteraciones}{2}$
- Max-Tam-ListaTabú: Su valor estaba en el rango [n-3, n+3].
- Cantidad-iteraciones-Intensificación: Entre 3.000 y 7.000.
- Tam-Bloque: Entre 2.000 y 50.000 iteraciones.
- Frecuencia-Opt-Local: Cada $2n$ iteraciones, aproximadamente.

En las figuras 7 y 8 se pueden observar los (mejores) resultados que obtuvimos para cada una de las instancias evaluadas. Por cada una de ellas, se informa el costo de la mejor solución conocida hasta el momento en que desarrollamos este trabajo (noviembre 2002), el costo de la mejor solución obtenida por nuestro algoritmo, el costo de la mejor solución conocida al momento de la presentación de este trabajo (mayo 2004), las diferencias entre nuestras soluciones y las otras dos informadas y por último el tiempo (en segundos) que nuestro algoritmo demoró para obtener cada uno de los resultados que se describen.

Como se puede ver, nuestro algoritmo obtiene buenos resultados alcanzando siempre el óptimo para las instancias *NL4, NL6, NL8, CIRC4* y *CIRC6*. Además, podemos ver que el algoritmo es robusto, ya que para todas las instancias utilizadas, siempre obtuvimos resultados con una diferencia menor al *10,5%* con respecto a los mejores resultados conocidos, utilizando tiempos de procesamiento relativamente bajos. Teniendo en cuenta que, mientras realizábamos nuestros experimentos, los resultados obtenidos los comparábamos con los publicados en ese momento (noviembre 2002), es importante destacar, que en ese entonces nuestro algoritmo había conseguido las mejores soluciones para las instancias NL12 y NL14, que, como ya mencionamos, posteriormente fueron mejoradas por otros investigadores.

Problema	Mej. Sol Previa (Nov-02)	Mej. Sol TSparaTTP (Nov-02)	Mej. Sol Actual (May-04)	Dif % Nov-02	Dif % Actual	Tpo.(segs) Mejor Sol. TS para TTP
NL4	8276	**8276**	8276 *	0.0	0.0	11
NL6	23916	**23916**	23916 *	0.0	0.0	52
NL8	39721	**39721**	39721	0.0	0.0	642
NL10	61608	62561	59583	1.5	5.0	3831
NL12	119012	**118955**	112298	**-0.1**	5.9	6057
NL14	207075	**205894**	190368	**-0.5**	8.1	11332
NL16	284235	293013	267194	3.0	9.6	20830

Figura 7. Resultados obtenidos para las instancias NL.

** Indica las soluciones óptimas. En negrita aparecen las instancias para las cuales logramos la mejor solución en Nov.-02.*

Problema	Mej. Sol Previa (Nov-02)	Mej. Sol TSparaTTP (Nov-02)	Mej. Sol Actual (May-04)	Dif % Nov-02	Dif % Actual	Tpo. Mejor Sol. (segs) TS para TTP
CIRC4	20	**20**	20 *	0.0	0.0	15
CIRC6	64	**64**	64 *	0.0	0.0	31
CIRC8	132	134	132	1.5	1.5	390
CIRC10	266	268	254	0.7	5.5	1956
CIRC12	448	458	420	2.2	9.0	1635
CIRC14	712	730	682	2.5	7.0	9277
CIRC16	984	1074	976	9.1	10.0	23974
CIRC18	1442	1550	1398	7.5	10.5	35236
CIRC20	1990	2086	1898	1.8	9.9	48412

Figura 8. Resultados obtenidos para las instancias Circulares.

** Indica las soluciones óptimas. En negrita aparecen las instancias para las cuales logramos la mejor solución en Nov.-02.*

7. Conclusiones

En este trabajo hemos descrito las características que hacen al problema de planificación de fixture deportivo y hemos planteado las principales dificultades que las ligas organizadoras de torneos tienen que enfrentar frecuentemente. Luego describimos el *Traveling Tournament Problem*, cuyas características lo convierten en un problema interesante para analizar y resolver. Finalmente, presentamos un algoritmo Tabú Search para el TTP, con el que se obtienen muy buenos resultados utilizando relativamente poco tiempo de procesamiento.

Creemos que lo expuesto en este trabajo contribuye para incentivar la investigación en el área de planificación de fixtures deportivos. A continuación daremos algunas ideas que pueden tomarse como base para trabajos futuros:

- Agregar más restricciones al TTP y ver cómo estas afectan al desempeño del algoritmo. En particular, por ejemplo, pedir que la cantidad máxima

de partidos seguidos de local o visitante sea *2* ó *4* (en lugar de *3*), o pedir que las distancias recorridas por cada equipo sean parejas tratando de mantener el costo total lo más bajo posible.

- Aplicar las ideas implementadas en el algoritmo *TSparaTTP* a otros problemas de fixtures deportivos.
- Estudiar la complejidad computacional teórica del TTP. A pesar de que se presume su NP-completitud, este resultado aún, no ha sido probado en la literatura. Incluso, es un problema abierto conocer la complejidad de generar el fixture óptimo para un solo equipo, manteniendo las restricciones del TTP.

Referencias

[1] Anagnostopoulos, A.; Michel, L.; Hentenryck, P. & Vergados, Y. "A simulated annealing approach to the traveling tournament problem", Proceedings CPAIOR'03, 2003.

[2] Cardemil, A., "Optimización de Fixtures Deportivos: Estado del arte y un algoritmo Tabu Search para el Traveling Tournament Problem", "Tesis de Licenciatura, Departamento de Computación, FCEyN Universidad de Buenos Aires, 2002".

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

[4] 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.

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

[6] 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.

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

[8] Easton, K., Nemhauser, G.& Trick, M. A. "The Traveling Tournament Problem description and benchmarks", T. Walsh editor, Principles and Practice of Constraint Programming, Vol 2239 of Lecture Notes in Computer Science, 580-584. Springer, 2001.

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

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

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

[12] 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.

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

[14] Henz, M.; Müller T.; Thiel, S. & Van Brandenburg, M. "Global Constraints for Round Robin Tournament Scheduling", European Journal of Operations Research (*EJORS),* 2004, 153 (1), 92-101.

[15] Leong, G. "Constraint programming for the travelling tournament problem", 2003, disponible en: www.comp.nus.edu.sg/~henz/students.

[16] Ribeiro, C. & Urrutia, S. "Heuristics for the Mirrored Traveling Tournament Problem", aceptado en PATAT-04, (2004).

[17] 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.

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

[19] Shen, H. & Zhang, H. "Greedy Big Steps as a Meta-Heuristic for Combinatorial Search", 2003, disponible en: http://goedel.cs.uiowa.edu/AR-group/readings/aaai_ttp.pdf

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

[21] 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.

[22] Trick, M. "Challenge Traveling Tournament Instances", sitio web del TTP, en http://mat.gsia.cmu.edu/TOURN

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

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

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