Inteligencia Artificial

Algoritmo basado en hormigas para resolver el Relaxed Travel Tournament Problem

Roberto Fuentes Zenteno

4 de septiembre de 2019

Resumen

Traveling Tournament Problem (TTP) es un problema que busca ordenar de la manera más óptima el enfrentamiento entre diversos equipos, creando así una programación deportiva óptima, es decir, construir un horario con la distancia mínima de desplazamiento para cada equipo. En dichos horarios, uno o más equipos pueden tener una exención (*bye*) en cualquier intervalo de tiempo. En este trabajo se analizará una variante del TTP donde los *byes* son posibles, el cual es conocido como *Relaxed Traveling Tournament Problem* (RTTP). Incluso pequeños ejemplos de este problema parecen ser difíciles de resolver, por lo que este problema será resuelto usando la metaheurística conocida como optimización de colonias de hormigas (*Ant Colony Optimization*, ACO) , la cual es una metaheurística que ha demostrado su calidad , versatilidad y rapidez en varios problemas de optimización combinatoria.

Keywords: *Relaxed Traveling Tournament Problem*, *Ant Colony Optimizatión*, programación deportiva.

1. Introducción

Antes de comenzar hablando de RTTP, analizaremos de que trata TTP y su estado del arte, ya que RRTP es una variante del *Traveling Tournament Problem*.
El problema surge por el trabajo realizado para la Liga Mayor de Béisbol (MLB) en Estados Unidos. Crear una programación horaria razonable de la MLB es una tarea difícil y costosa, ya que treinta equipos juegan 162 partidos que se extiende desde principios de abril hasta finales de septiembre. Mientras que la creación de un horario implica "hacer malabarismos" con cientos de peticiones y requisitos, los temas clave para un horario giran en torno dos factores importantes: La distancia de viaje y el "flujo": patrón de los partidos en casa y fuera de casa en el horario (conocido como patrón *home/away*). Si bien esta es la liga de béisbol mas importante a nivel nacional, este tipo de programación horaria perfectamente podría ser aplicado a toreos o ligas menores, teniendo en cuenta que podría llegar a modificarse las restricciones o el modelamiento debido a las reglas que poseen estos partidos.

Figura 1: Logo de la MLB y los distintos equipos que participan.

Mientras que los equipos desean limitar la cantidad total que viajan, otros también están preocupados por cuestiones más tradicionales con respecto a sus patrones *home/away*. Por ejemplo, en el baloncesto universitario, algunas ligas trabajan en un horario de viernes a domingo en el que los equipos viajan directamente de su partido del viernes a su partido del domingo. Esto ha sido explorado por Campbell y Chen [20], donde el objetivo era minimizar la distancia recorrida por equipos el fin de semana. Russel y Leung[18] tenían un objetivo similar en su trabajo de programar el horario para partidos de béisbol en las ligas menores. En ambos casos, el límite del número de partidos consecutivos fuera de casa era fijo, lo que dio lugar a límites interesantes basados en variantes del problema. Otras referencias a problemas de programación de deportes se pueden encontrar en Nemhauser y Trich [15].
Se propone entonces el problema llamado *Traveling Tournament Problem* que resume los temas clave para crear un horario que combine los problemas de viaje y los problemas de "flujo" (patrones de viaje ida y vuelta), el cual puede ser aplicado para armar horarios a distintos deportes (Tenis, Fútbol, Baloncesto, entre otros). Este problema resulta difícil de resolver debido a la combinación de los viajes entre equipos, donde casos con tan sólo ocho equipos son intratables en relación con otros estados del arte de distintos autores. Esto hace que el problema sea atractivo como punto de referencia: es fácil de declarar y los datos requeridos son mínimos.

2. Definición del Problema

2.1. Traveling Tournament Problem

2.1.1. Definición

En TTP, n equipos juegan unos contra otros durante un período de tiempo con un determinado esquema. Un ejemplo es el *double round robin*: Cada equipo juega dos veces contra todos los demás equipos: un partido en casa (*home*) cuando el equipo i, donde $i \in \{1, 2, \ldots, n\}$ usa sus propias instalaciones y un partido fuera de casa (*away*) cuando el partido tiene lugar en la

instalación del oponente. Cuando el número de equipos n es par, y cada equipo juega a lo sumo un partido por fecha, el número mínimo de fechas en las que se distribuyen los $n \cdot (n-1)$ partidos es igual a $M = 2n - 2$. En un torneo normal, se trata de minimizar el número de descansos, es decir, que un equipo juegue dos partidos consecutivos en la misma ubicación, donde el término ubicación denota a *home* o *away*. Para aumentar la tensión en el torneo y que así más gente quiera ver los partidos, se debe evitar la situación de que dos equipos jueguen sus dos partidos entre sí en dos fechas consecutivas, lo cual llamaremos repetidor (*repeater*).
TTP se define como un torneo *double round robin* para n equipos, con n par y sin *repeater*. Se permiten descansos ya que en un país grande (por ejemplo, Estados Unidos), donde las distancias entre las ciudades de origen pueden ser muy grandes, es ventajoso organizar un recorrido para varios partidos consecutivos fuera de casa.
Un ejemplo de una planificación para 4 equipos es la siguiente:

	Fechas					
Equipos	1	2	3	4	5	6
A	B	C	D	-B	-C	-D
B	-A	-D	C	A	D	-C
C	D	-A	-B	-D	A	B
D	-C	B	-A	C	-B	A

Cuadro 1: Programación para $n = 4$

Además, una planificación sin repetidores (*No repeaters*) significa que en fechas continuas no deben darse partidos del tipo i con $-j$ y luego j con $-i$, como se muestra en siguiente tabla :

	Fechas	
Equipos	1	2
A	-B	B
B	A	-A

Cuadro 2: Ejemplo de una programación deportiva usando NoR-TTP (*No repeaters TTP*)

2.1.2. Objetivo

El objetivo es minimizar la suma de las duraciones de los recorridos por los equipos, donde asumimos que todos los equipos comienzan y terminan en su ciudad natal. TTP busca encontrar un calendario compacto, es decir, la cantidad de espacios de tiempo es igual a la cantidad de juegos que realiza cada equipo. Esto obliga a cada equipo a jugar en una determinada ranura de tiempo.

2.1.3. Parámetros

Los parámetros de TTP son los siguientes:

- n: Este parámetro representa el número de equipos que participa en el torneo. Se asume n par.
- D: Este parámetro es una matriz $n \times n$, la cual representa una matriz de distancia entera. Asumimos que todos los equipos comienzan y terminan en su ciudad natal , por lo que la matriz de distancia D es simétrica. Para representar un elemento de esta matriz usaremos la notación $d_{i,j}$, donde $1 \leq i, j \leq n$, donde $d_{i,j}$ denota la distancia entre los equipos i y j.

- U y L: Los parámetros enteros L y U controlan la cantidad de partidos consecutivos que un equipo tiene que jugar tanto de local como de visita. Generalmente estos toman los valores $L = 1$ y $U = 3$, de forma que cada planificación pueda tener a lo mas tres juegos consecutivos en el mismo lugar de juego (Visita/Local). Por otra parte para $L = 1$ y $U = n$ se produce el caso que el equipo se mantiene en viaje durante todo el torneo, lo que es equivalente al problema del TSP (*Traveling Salesman Problem*), en cambio si tenemos un valor muy pequeño para U, el equipo deberá retornar muy seguido a su origen, lo que origina que cada partido de visita está prácticamente intercalado con los de local.

2.2. Otras variantes

Otras variantes al problema TTP son:

- En espejo (*mirrored*): Se genera un *Single Round Robin* en las primeras $(n-1)$ fechas y las siguientes $(n-1)$ fechas corresponden al mismo *Single Round Robin* anterior, pero con los lugares de los encuentros invertidos. Por ejemplo, si el equipo i juega de local en la fecha 1 con el equipo j, en la fecha n jugará de visita con el equipo j. El símbolo $-$ denota un partido de vuelta. Un ejemplo se adjunta en la siguiente tabla:

	Fechas					
Equipos	1	2	3	4	5	6
A	C	B	D	-C	-B	-D
B	D	-A	-C	-D	A	C
C	-A	D	B	A	-D	B
D	-B	-C	-A	B	C	A

Cuadro 3: Ejemplo de una programación deportiva usando MTTP (*Mirrored TTP*)

- *Non-round robin scheduling* (Non-RR TTP): Este problema es una variante del TTP propuesto por Douglas Moody. En esta variante, los equipos no juegan un torneo *double round robin*, sino que hay un valor de "Combate" (*Matchups*) entre los equipos i y j, que representa el número de veces que i debe visitar j. El TTP original es un Non-RR TTP con un valor de emparejamiento de $1 \forall i, i \neq j$.

- *Relaxed*: Esta es la variante que se resolverá en este trabajo. Propuesto por Renjun Bao and Michael Trick [2], esta variante nos indica que la planificación no es compacta. Esto quiere decir que los equipos tendrán exenciones (*byes*) en sus planificaciones. *Bye* se refiere a una situación en la cual uno o varios participantes no compiten temporalmente en una o varias rondas de un torneo. El número de *byes* es controlado por un parámetro K, el cual representa el número de *byes* por cada equipo en la planificación. $K = 0$ corresponde al problema TTP original. La longitud del programa de RTTP es $2(n-1)+K$. Mientras mayor sea el valor de K, mayor será la flexibilidad para encontrar soluciones con menor distancia recorrida.

  Dicho de otra forma, tenemos:

  - $|S_{i,j} \in 0, \cdots, N|$, con $||S_{i,j}|| \neq i$.
  - $S_{i,j} > 0$ cuando un partido se juega en casa.
  - $S_{i,j} < 0$ cuando un partido se juega de visita.
  - $S_{i,j} = 0$ cuando hay una exención, es decir, un día libre para el equipo i.

3. Estado del Arte

3.1. Ant Colony System

El modelo usado para resolver este problema es el algoritmo *Ant Colony System* (ACO) presentado por Marco Dorigo [10] en el año 1997. Este criterio busca repartir en diferentes sectores del espacio de búsqueda hormigas que exploten el área donde fueron asignadas. Esto quiere decir que el algoritmo tendrá distintos puntos partidas para comenzar a buscar soluciones, guardando siempre la mejor solución. El algoritmo utiliza un método probabilístico para la elección de valores en cada nodo, denominado *random-proportional method*:

$$P_k(r,s) = \frac{[\tau(r,s)]^\alpha \cdot [\eta(r,s)]^\beta}{\sum_{u \in J_k(r)} [\tau(r,s)]^\alpha \cdot [\eta(r,s)]^\beta} \tag{1}$$

Donde τ es la matri¡de feromona, η es la matriz de visibilidad, J_k el conjunto de valores permitidos por una hormiga k en un nodo r, α y β son parámetros que entregan la importancia relativa entre la feromona (exploración) y la visibilidad (explotación). Luego, el valor a escoger en un nodo se selecciona mediante la siguiente regla:

$$j = \begin{cases} argmax_{u \in J_k(r)} [\tau(r,u)]^\alpha \cdot [\eta(r,u)]^\beta & \text{si } q \leq q_0 \\ P_k(r,s) & \text{si } q > q_0 \end{cases}$$

Donde q_0 es un parámetro y q es un valor escogido al azar por el algoritmo. Además se utiliza la actualización local y global de las feromonas del algoritmo para seleccionar las mejores soluciones:

- **Regla de actualización global de feromonas**: En ACS solo la mejor hormiga puede depositar feromonas, modificando de forma global la matriz de feromonas (es decir, la hormiga que construya el recorrido mas corto). Esto tiene por fin que la búsqueda sea dirigida a mejores soluciones. La actualización global se realiza después de que todas las hormigas han completado sus recorridos. La regla se define matemáticamente como:

  $$\tau_{i,j}(t) = (1-\rho)\tau_{i,j}(t) + \rho \Delta\tau_{i,j}(t)$$

  Donde $0 < \rho < 1$ es un parámetro, y $\Delta\tau_{i,j}(t) = \frac{1}{L^+}$, Donde L^+ es el largo de valores posibles para un nodo.

- **Regla de actualización local de feromonas**: Luego de cada transición una hormiga modifica localmente la matriz con la siguiente definición matemática:

  $$\tau_{i,j}(t) = (1-\rho)\tau_{i,j}(t) + \rho\tau_0$$

  Donde τ_0 es un parámetro del algoritmo que indica la cantidad inicial de feromonas en la matriz.

3.2. Traveling Tournament Problem

El problema original (TTP) fue presentado por K. Easton, G. Nemhauser y M. Trick [11] en el 2001 en su paper *The traveling tournament problem: Description and benchmarks*. Esta investigación fue inspirada por el trabajo realizado para la Liga Mayor de Béisbol (MLB) en Norteamérica, considerando la distancia de viaje y el "flujo": patrón de los partidos en casa y fuera de casa en el horario. Este paper menciona un resumen básico de la estructura de TTP,

sus variables y el objetivo final, el cual es minimizar el viaje de todos los equipos, entregando una planificación horaria óptima. Además, se mencionan algunos métodos para solucionar este problema como *Integer Programming* y *Constraint Programming*, mencionando también instancias base para testear un algoritmo. Se hace énfasis en que es un problema difícil de resolver incluso para instancias pequeñas (para instancias con $n = 8$ hacia arriba no existia solución).
Por otra parte en el año 2003 los mismos autores publicaron el paper *Solving the Travelling Tournament Problem: A Combined Integer Programming and Constraint Programming Approach* [12], donde muestran los resultados combinando la programación lineal junto con la programación con restricciones. Los autores pudieron resolver las instancias con 4 y 6 equipos, sin embargo para 8 equipos en adelante no pudieron seguir resolviendo el problema.
Posterior a esto, en el año 2003 H. Crauwels y D. Van Oudheusden buscaron solucionar TTP con ACO (Ant Colony Optimization) en su paper *Ant Colony Optimization and Local Improvement* [6]. TTP fue resuelto por búsqueda local por A. Anagnostopoulos, L. Michel, P. Van Hentenryck, y Y. Vergados en su paper *A simulated annealing approach to the traveling tournament problem* [1] usando la técnica *simulated annealing*, y por los autores L. Di Gaspero y A. Schaerf en su paper *A composite-neighborhood tabu search approach to the traveling tournament problem* [13] utilizando *Tabu Search*, obteniendo resultados muy influyentes e interesantes para los estados del arte posteriores.
Luego en el 2007 P.-C. Chen, G. Kendall y G. V. Berghe propusieron resolver TTP usando ACO como una hiperheuristica en su paper *An ant based hyper-heuristic for the travelling tournament problem* [5], obteniendo resultados atractivos para las demás investigaciones.
En el año 2009 David C. Uthus, Patricia J. Riddle y Hans W. Guesgen publicaron su paper denominado *An Ant Colony Optimization Approach to the Traveling Tournament Problem* [4], el cual fue usado en este documento como referencia. Este paper busca optimizar el problema con una mezcla entre *forward cheking*, *backjumping* y ACO, denominando a la técnica como AFC. La técnica se presenta a continuación:

Algoritmo 1: AFC

```
X Input: X, D, C
i ← 1;
SelectVariable;
D'_i ← D_i for 1 ≤ i ≤ n //Initialize domains ;
J_i ← for 1 ≤ i ≤ n // Initialize conflict sets while 1 ≤ i ≤ n do
    x_i ← SelectValueACO ;
    if x_i = null ∧ #backjumps = limit then
        RestartAnt ;
    end
    else if x_i = null then
        i' ← i;
        i latest index in J_i;
        if SafeBackjump then
            J_i ← J_i ∪ J_i' −{x_i}
        end
        Undo changes to D' and J from i' to i
    end
    else
        i ← i + 1;
        SelectVariable ;
    end
end
Return X
```

La función *SelectValueACO* escoge un valor aplicando el *proportional-random method* de ACO

como se explica a continuación. Luego, se evaluea si este valor presenta inconsistencias con el conjunto de restricciones que tiene una casilla. Si este conjunto restringe todas las variables permitidas entonces se retorna *null*. Caso contrario devuelve el valor seleccionado, colocando ese valor dentro de la planificación. El número de *backjumps* se definirá más adelante, y la sección *Restart Ants* junto a *SafeBackjump* se explicarán con mayor detalle posteriormente.

Algoritmo 2: SelectValueACO

```
X Input: X, D, C
while D'_i ≠ ∅ do
    a ← Value chosen with ACO's proportional-random method ;
    D'_i ← D'_i \ a ;
    emptydomain ← false ;
    for i < k ≤ n do
        for b ∈ D'_k do
            if a not consistent with value b then
                J_k ← J_k ∪ i ;
                D'_k ← D'_k \ b
            end
        end
        if D'_k = ∅ then
            emptydomain ← true ;
        end
    end
    if emptydomain then
        Reset changes to D'_k and J_k, i < k ≤ n
    end
    else
        Return a
    end
end
Return null
```

El algoritmo 1 recibe un conjunto de variables X, sus dominios asociados D y sus restricciones respectivas C. Se selecciona por ACO una variable, y se evalua si la variable seleccionada es nula (debido a que al aplicar las restricciones no quedó ninguna variable en el conjunto X y por lo tanto no retornó nada) y el número de saltos llegó al limite. Esto quiere decir que el algoritmo trató por varios caminos dentro del árbol de posibilidades de solución, llegando en todas sus ramas a puntos que no tendrían solución. En este caso, se realiza un *reset* de las hormigas, comenzando nuevamente a buscar soluciones desde otro punto.
Si nuestra variable escogida es *null* pero aún no se alcanza el limite de *backjumps* quiere decir que puede existir otra rama que nos pueda llevar a una solución viable. Siendo así, volvemos donde se hizo la última asignación de una variable (a través de un *safeBackjumping*), borrando este valor y probando con otro número que nos lleve a un camino distinto, con el objetivo de que la variable *null* anterior se transforme en un conjunto de opciones para escoger y seguir avanzando por esa rama.
Finalmente si la variable escogida es un valor distinto de nulo, quiere decir que es una variable viable y se asigna en la casilla correspondiente, quitando ese valor del conjunto de dominio respectivo, avanzando a la siguiente casilla para asignar.
En resumen, el algoritmo irá por cada ronda y cada equipo, buscando un contrincante posible para poder asignar, devolviendo finalmente una planificación para la instancia entregada.
Algunos detalles importantes a considerar son los siguientes:

- *Unsafe Backjumping*: Un algoritmo *backjumping* se considera seguro si no salta lo suficientemente atrás para no encontrar ninguna solución [8]. Esto no es problema para ACO, ya

que sólo se está tratando de crear una solución factible. Una nueva idea que se consideró es el uso de *unsafe backjumping* para ayudar a reducir el número total de saltos hacia atrás. *Unsafe backjumping* está diseñado para problemas que tienen un gran espacio de solución, como es el TTP. Cuando se compara con el *safe backjumping*, el *unsafe backjumping* a veces retrocede lo suficiente para que algunas soluciones puedan pasar por alto. Esto sucede cuando el algoritmo salta primero del índice i al índice j, y luego hace un segundo salto hacia atrás del índice j a otro índice más atrás. Los *unsafe backjumping* pueden retroceder más de lo necesario, permitiendo que el algoritmo salga de una solución infactible más rápido. Esto permite continuar construyendo a partir de una solución factible con mayor rapidez, reduciendo el tiempo necesario para encontrar una solución factible (esto sólo es práctico si el espacio de solución factible es grande). El *unsafe backjumping* se realiza dentro del Algoritmo 2 omitiendo la línea 12 (*SafeBackjump*).

- *Ant Restarts*: Otra característica que los autores añadieron al algoritmo AFC es el reinicio de hormigas (*Ant Restarts*). Una vez que el algoritmo ha pasado por demasiados *backjumps*, el algoritmo reiniciará el proceso de construcción de la solución para una hormiga. Esto es útil para problemas donde es difícil propagar todas o algunas de las restricciones. Su objetivo es dar al algoritmo tiempo suficiente para construir una solución, pero limitando el número de *backjumps* en los casos en los que se producen demasiados saltos hacia atrás. El límite para el número de *backjupms* antes de un reinicio de hormiga se establece como $b \cdot n$ [19], siendo b un escalar y n el número de variables. Una vez alcanzado este límite, el algoritmo reiniciará la hormiga en el índice 1 y reiniciará los dominios y variables junto con el contador de *backjumps*.

- *Pheromone matrix*: Para TTP, una casilla de la matriz de feromonas se define como $\tau_{i,j,k}$, el cual representa que tan conveniente es que el equipo i juegue en casa contra el equipo j durante la ronda k. La feromona se reinician una vez que han pasado una cierta cantidad de ciclos. Para actualizar la matriz de feromonas, el algoritmo utiliza diferentes hormigas para diferentes ciclos. Antes de reiniciar, se compara la mejor hormiga actual y la mejor hormiga global, dejando guardada la mejor de ambas.

- *Local Search*: Una vez que se termine una iteración y se cuente con la mejor hormiga de aquella iteración, la idea es repararla mediante una búsqueda local, haciendo que cada hormiga explote lo más que pueda su área de exploración. Para este trabajo de decidió usar *Hill Climbing with restarts* como algoritmo de búsqueda local, debido a su rapidez, eficiencia y fácil implementación.

3.3. Relaxed Traveling Tournament Problem

RTTP es una variación de TTP propuesto por los mismos autores[2] en el año 2010 que busca instaurar el concepto de *byes* en la planificación deportiva, los cuales son días libres donde un conjunto de equipos no juega en una determinada ronda (se consideran como dias comodines o *jokers*). Como cada equipo debe enfrentarse al resto del grupo, esto hace que en la planificación se agreguen tantas columnas extras como *byes* hayan. Esto quiere decir que si hay 4 equipos que se enfrentan entre si, y cada uno tiene derecho a un día libre, entonces la planificación final deberá tener una columna extra para los *byes*. La mayoría de las instancias de prueba se hacen con un número de *byes* $K \in \{1, 2, 3\}$, agregando K columnas extras a la planificación final, ya que son instancias que tiene resultados interesantes de analizar (cuando $K = 0$ el problema es el original).
En el año 2011 Leslie Perez y la Doctora María Cristina Riff en su paper *New Bounds for the Relaxed Traveling Tournament Problems using an Artificial Immune Algorithm* [16] abordaron el RTTP mediante un algoritmo artificial inmune (CLONALG) [7]. Se Introducen nuevos movimientos que permiten al algoritmo tratar los *byes* de forma óptima a través de movimientos

locales. Se probó el algoritmo en las instancias propuestas por los autores originales, obteniendo muy buenos resultados y demostrando que esa técnica era muy buena para resolver el RTTP.
En el 2013 los autores Brandão y Pedroso publicaron el paper *A complete search method for the relaxed traveling tournament problem* [3] el cual busca resolver RTTP mediante un método de búsqueda completo basado en *branch-and-bound*, metaheurísticas como *hill climbing* y programación dinámica.
Finalmente en el año 2015 nuevamente la doctora María Cristina Riff junto a Elizabeth Montero publicaron un paper que lleva por nombre *RAIS$_{TTP}$ Revisited to Solve Relaxed Travel Tournament Problem* [14]. Ellas buscan métodos y estrategias que permiten simplificar el código de los algoritmos mostrados en su documento sin alterar su rendimiento. En el documento se estudia un algoritmo inmunológico artificial especialmente diseñado para resolver RTTP, el cual lo denominan como *Artificial Immune System for the Relaxed Traveling Tournament Problem* (RAIS$_{TTP}$), que ha logrado obtener nuevos límites para algunas instancias de este problema. Se utiliza además el sintonizador EvoCa (*Evolutionary Calibrator*)[17] para analizar los componentes del algoritmo. Los resultados muestran que el algoritmo es capaz de resolver las instancias al igual que el algoritmo original, obteniendo además nuevos límites para algunas instancias del problema.

3.4. Movimientos

En la mayoría de los papers se presentan 5 movimientos locales importantes que se usarán para el presente trabajo:

- *Swap Homes*(T_i ,T_j): Se intercambian los partidos de dos equipo T_i y T_j de visita a local y viceversa. Esto quiere decir que se invierten los signos dentro de la planificación deportiva.
- *Swap Rounds*(R_i ,R_j): Se intercambian los partidos de las rondas R_i y R_j, es decir, 2 columnas en la matriz de planificación.
- *Swap Teams*(T_i ,T_j): Intercambia los partidos de los equipos T_i y T_j excepto los que se involucren a si mismo. Esto quiere decir que se intercambian dos filas T_i y T_j de nuestra matriz de planificación, por lo que por columna se intercambian 4 casillas.
- *Partial Swap Rounds*(T_m ,R_i ,R_j): Para un equipo T_m cambiando los partidos de las rondas R_i con la R_j.
- *Partial Swap Teams*(T_i ,T_j ,R_m): Para una ronda R_m, intercambia los partidos de los equipos (filas) T_i y T_j.

Además, en [16] se presentan un movimiento y una técnica que son claves para RTTP y que se ocuparán dentro del algoritmo de búsqueda local:

- *Swap Byes Matches*: Este movimiento intercambiar datos de dos rondas: una ronda en la que los equipos $T_i^{R_i}$ y $T_j^{R_i}$ tienen un *bye*. La segunda ronda es cuando estos dos equipos juegan juntos ($T_i^{R_j}$ y $T_j^{R_j}$).
- *Grouping-Away-Byes*: Como el costo de un calendario de equipos corresponde al costo de sus partidos de visita, la idea es intentar agrupar los viajes de los equipos con costos de viaje más altos utilizando los *byes* según patrones *away* explicados en con mayor detalle en [16].

4. Representación y Modelo Matemático

4.1. Representación

La representación que se usará para la programación deportiva es una matriz de equipos $\times$ rondas ($n \times (2n-2)$) que llamaremos S, la cual es una matriz de números enteros donde a cada equipo T_i (con $i \in \{1, 2, \ldots, n\}$) se le asigna un valor entero entre $[1, n]$ y a cada ronda R_j, con $j \in \{1, 2, \ldots, (2n-2)\}$ se le asigna un valor entre $[1, (2n-2)]$ de la siguiente forma:

T/R	R_1	R_2	R_3	R_4	$\ldots$	R_{2n-2}
T_1	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,4}$	$\ldots$	$S_{1,(2n-2)}$
T_2	$S_{2,1}$	$S_{2,2}$	$S_{3,2}$	$S_{4,2}$	$\ldots$	$S_{2,(2n-2)}$
T_3	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$	$S_{4,3}$	$\ldots$	$S_{3,(2n-2)}$
T_4	$S_{4,1}$	$S_{4,2}$	$S_{3,4}$	$S_{4,4}$	$\ldots$	$S_{4,(2n-2)}$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
T_n	$S_{n,1}$	$S_{n,2}$	$S_{n,3}$	$S_{n,4}$	$\ldots$	$S_{n,(2n-2)}$

Cuadro 4: Representación de la solución para TTP

La entrada $S_{i,j}$ de la matriz del Cuadro 4 representa un partido jugado por el equipo T_i en la ronda R_j, el cual toma un número positivo $t \in \{1, 2, \ldots, n\}$ si el equipo T_i juega contra otro equipo (al cual fue asignado por t) en la ronda R_j en su propia casa. Caso contrario el partido se desarrolla en la ciudad oponente, por lo que t tomará los mismos valores pero en negativo.
Si $K > 0$, tendremos K columnas adicionales en nuestra representación, acompañado de rondas donde ciertos equipos no jugaran. Un ejemplo de esto se muestra en el Cuadro 5, que considera un número de equipos $n = 4$, número de rondas $2 \cdot 4 - 2 = 6$, y un número de exenciones $K = 3$:

T/R	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9
T_1	3	0	-4	-2	-3	0	2	4	0
T_2	0	0	3	1	4	-3	-1	0	-4
T_3	-1	-4	-2	0	1	2	4	0	0
T_4	0	3	1	0	-2	0	-3	-1	2

Cuadro 5: Solución óptima para RTTP con $K = 3$.

Teniendo esta representación, tenemos las siguientes variables para una instancia en particular:

- n: número (par) de equipos.
- L: Cota inferior de la restricción *At_Most*.
- U: Cota superior de la restricción *At_Most*.
- D: matriz de $n \times n$ donde el valor $d_{i,j}$ de la matriz D representa la distancia entre la ciudad i y j.

4.2. Restricciones

Las restricciones de *Traveling Tournament Problem* se presentan a continuación:

1. C_1: $S_{i,r} = j \leftrightarrow S_{j,r} = -i, \forall r : 1 \leq r \leq (2n-2), i, j \in \{1, 2, \ldots, n\}$
   Esta restricción nos dice que si i juega de local contra j, entonces j juega de visitante contra i.

2. $C_2 : \sum_{r=1}^{2n-2} \Psi(|S_{t,r}|, t') = 2, \forall t \in \{1, 2, \ldots, n\}, t' \in \{1, 2, \ldots, n\}, t \neq t'$
   Donde :

$$\Psi(g, t) = \begin{cases} 1 & \text{si } g = t, \\ 0 & \text{Otro caso.} \end{cases}$$

   Esta restricción nos dice que en cada ronda, todos juegan contra un equipo distinto, ya que cada equipo debe jugar de visita y de local, y por la función Ψ por cada fila el valor debe ser siempre 2 (uno por juego de ida y otro uno por juego de vuelta).

3. C_3: $\sum_{r=1}^{(2n-2)} S_{t,r} = 0, \forall t \in \{1, 2, \ldots, n\}$
   Esta restricción nos dice que todo equipo juega dos veces (ida y vuelta) contra todos los demás. Si juegan ida y vuelta aparecerá el mismo número con signo positivo y negativo, lo que implica que la suma de 0.

4. C_4: $|\sum_{k=0}^{u-1} \Omega(S_{t,r+k})| \leq u, \forall t \in \{1, 2, \ldots, n\}, \forall r \in \{1, 2, \ldots, (2n - 1 - u)\}$

5. C_5: $|\sum_{k=0}^{l-1} \Omega(S_{t,r+k})| \geq l, \forall t \in \{1, 2, \ldots, n\}, \forall r \in \{1, 2, \ldots, (2n - 1 - l)\}$
   Donde:

$$\Omega(g) = \begin{cases} 1 & \text{si } g > 0, \\ \text{-1} & \text{si } g < 0 \\ 0 & \text{Otro caso.} \end{cases}$$

   Las restricciones C_4 y C_5 se aseguran de que las restricciones "A lo más"(*AtMost*) sean satisfechas. Esto quiere decir que se restringe el nímero de partidos de ida y vuelta entre u (C_4) y l (C_5).

6. C_6: $|S_{t,r}| \neq |S_{t,r+1}|, \forall t \in \{1, 2, \ldots, n\}, \forall r \in \{1, 2, \ldots, 2n - 3\}$

   Esta restricción nos garantiza que se cumpla la restricción "Sin repetidores" (*NoRepeaters*), es decir, dos equipos no pueden jugar entre sí en dos rondas consecutivas.

4.3. Función objetivo

Se usará como función objetivo la suma de los recorridos de cada equipo a lo largo del torneo, ya que el objetivo del *Traveling Tournament Problem* es minimizar dicha suma, respetando las restricciones expuestas.
TTP especifica que todos los equipos comienzan (ronda 0) y terminan (ronda $2n - 1$) el torneo en su propio estadio. Siendo así, asumiremos que $S_{k,0}$ y $S_{k,(2n-1)}$ son mayores a 0. Definiremos $\Phi(t, r)$ como la distancia que debe recorrer el equipo k del estadio donde juega la ronda $r - 1$ al estadio donde juega la ronda r de la siguiente manera:

$$\Phi(t, r) = \begin{cases} D(|S_{t,r-1}|, |S_{t,r}|) & \text{si } S_{t,r-1} < 0 \text{ y } S_{t,r} < 0 \text{ (i)}, \\ D(|S_{t,r-1}|, t) & \text{si } S_{t,r-1} < 0 \text{ y } S_{t,r} > 0 \text{ (ii)}, \\ D(t, |S_{t,r}|) & \text{si } S_{t,r-1} > 0 \text{ y } S_{t,r} < 0 \text{ (iii)}, \\ 0 & \text{si } S_{t,r-1} > 0 \text{ y } S_{t,r} > 0 \text{ (iv)}, \end{cases}$$

Donde:

- (i): El equipo k juega de visita en las rondas $r-1$ y r. La distancia recorrida es la distancia entre $S_{t,r-1}$ y $S_{t,r}$ (distancia que viaja el equipo k de una ciudad a otra) en las rondas r y $r - 1$.

- (ii): El equipo k juega de visita en la ronda $r-1$ y de local en la ronda r. La distancia recorrida es la distancia del equipo k con la ciudad de su rival en la ronda $r-1$.
- (iii): El equipo k juega de local en la ronda $r-1$ y de visita en la ronda r. La distancia recorrida es la que separa a la ciudad del equipo k con la de su rival en la ronda r.
- (iv): El equipo juega de local en las rondas $r-1$ y r. Siendo así, el equipo permanece en su ciudad y la distancia que recorre entre esas 2 rondas es 0, ya que no se mueve de dicha ciudad.

Dicho esto, definimos nuestra función de evaluación como:

$$f(S) = \sum_{t=1}^{n} C_t$$

Donde C_t indica la distancia total recorrida por el equipo t a lo largo de todo el torneo según S .El valor de $C_t, t \in \{1, 2, \ldots, n\}$ se calcula de la siguiente manera:

$$C_t = \sum_{r=1}^{2n-1} \Phi(t, r)$$

5. Descripción del algoritmo

Para la construcción de la solución se uso un esquema similar al de David C. Uthus, Patricia J. Riddle y Hans W. Guesgen [4]. Se Aprovechará las ideas *looking forward* con la propagación de restricciones y *looking backwards* con *backjumping*, integrando ACO con FC-CBJ (*forward checking and conflict-directed back-jumping*). Este algoritmo propuesto por Dechter [9] se conoce como AFC (*ant colony optimization with forward checking and conflict-directed back-jumping*). Se uso como método de búsqueda local *Hill Climbing* para reparar la solución, y además se uso el método *Unsafe backjumping* para obtener soluciones más rapidas en cada iteración.

Algoritmo 3: AFC_{RTTP}

```
Result: S_global
Input: S, τ, η, k, n, m
i ← 1;
j ← 1;
S_Global = Planificación vacía ;
S_Local = Planificación vacía ;
while a = 1, 2, ..., iteraciones do
    while m = 1, 2, ..., hormigas do
        while 1 ≤ j ≤ (2 · n − 2 + k) do
            while 1 ≤ i ≤ n do
                L ← Lista de candidatos de la casilla S_{i,j} de S_Local;
                Constraints(S_local, L_local) ← Se eliminan de L valores según restricciones
                 de S_{i,j} de S_Local;
                if L.size() == 0 and #backjumps == limit then
                    RestartAnt ← Se reincia la hormiga ;
                    i ← 1;
                    j ← 1;
                end
                else if L.size() == 0 then
                    Unsafe Backjumping ;
                end
                else
                    SelectValueACO(L) ;
                    i ← i + 1;
                    j ← j + 1;
                end
            end
        end
        LocalSearch(S_Local) ← Reparar solución obtenida;
        PheromoneUpdate;
        AllLSolution.append(S_Local);
    end
    if m == 0 then
        S_Global = bestCost(AllLSolution);
    end
    else
        S_aux ← bestCost(AllLSolution);
        if cost(aux) < cost(S_global) then
            S_global = aux;
        end
    end
end
```

El algoritmo comienza con una planificación vacía, y para un cierto número de iteraciones crea m hormigas que empezarán desde distintos equipos a construir soluciones. Para cada hormiga, se empieza a iterar por cada fila y se insertan en una lista de candidatos (L) los valores que pueden ser colocados en esta casilla. Luego, se aplica la función $Constraints(S_{local}, L)$, la cual quita de la lista de candidatos los valores que violan alguna de las restricciones de la matriz de planificación. Si el tamaño de la lista es 0 y además se alcanza el limite de *backjumps* entonces se reinicia la hormiga y se comienza la búsqueda nuevamente. Por otra parte, si el tamaño de

L es 0 pero aun no se alcanza el limite de *backjumps* entonces se procede a realizar un *unsafe backjump*, y el valor que se encontraba en la casilla destino del *unsafe backjump* es quitado de la lista de candidatos. Finalmente, si no sucede ninguna de esas condiciones, se procede a seleccionar de la lista de candidatos un valor mediante la función $SelectValueACO$ usada en [4]. Cabe destacar que al momento de escoger un *bye* (0) dentro de la planificación, el oponente a elegir que también tendrá día libre será escogido al azar. Luego de que una hormiga termina de construir su solución, se repara con *hill climbing*, se actualiza la matriz de feromonas y finalmente es agregada la solución a una lista de soluciones. Finalmente, se guarda en S_{global} la mejor solución de todas las guardadas en el arreglo (es decir, la que tenga menor costo).

6. Experimentos

En esta sección se detallan los experimentos realizados para probar el algoritmo propuesto, además de los parámetros utilizados para conducir los experimentos.

6.1. Parámetros del algoritmo

Debido a que algoritmo usado para resolver RTTP es basado en hormigas, debemos mencionar todos los parámetros que pueden hacer variar su calidad, tiempo de ejecución y cantidad de iteraciones necesarias para converger a un óptimo. Estos parámetros son:

- τ_0: Parámetro inicial con el que se rellena la matriz de feromonas, además de estar presente en la formula de actualización local de la misma.
- α: Parámetro asociado a la importancia que se le da a la matriz de feromonas.
- β: Parámetro asociado a la importancia que se le da a la matriz de visibilidad.
- q_0: Número usado para la regla de transición de ACS.
- r: Cantidad de *restart* que usará *hill climbing*.
- $\#backjumps$: Cantidad de *backjumps* consecutivos que hará el algoritmo si se queda sin opciones.

Para la mayoría de los parámetros se usaron las configuraciones presentadas por Dorigo [10] y David Uthus [4]. Para la cantidad de *restarts* se usó $r = 10$, un número pequeño pero suficiente para que el algoritmo trabaje de forma eficiente y sin tardarse demasiado.

6.2. Ejecución del algoritmo

Para la ejecución del algoritmo se debe tomar en cuenta las semillas, el hardware y las instancias a utilizar.

- **Semillas y *Hardware***: Para el caso de las semillas se usó la libreria *random* y *ctime* de C++ para poder setear los números escogidos al azar con la hora del reloj. Para que los experimentos puedan ser reproducibles, al momento de generar la planificación final también se muestra y guarda la semilla ocupada para realizar este proceso, pudiendo así tener control de los resultados finales. Con respecto al *Hardware* se ocupó un computador con procesador Intel Core $i5$, GPU de 2,7 GHz, 8 gb de RAM y con sistema operativo Mac.
- **Instancias**: Las instancias escogidas para aplicar el algoritmo fueron algunas de las que se encuentran en la pagina CSPLib del *Relaxed Traveling Tournament Problem*, la cuales son las siguientes:

- NL4, con $K = 1, 2, 3$.
- NL6, con $K = 1, 2, 3$.
- NL8, con $K = 1, 2, 3$.
- NL10, con $K = 1, 2, 3$.
- NL12, con $K = 1, 2, 3$.
- NL14, con $K = 1, 2, 3$.
- NL16, con $K = 1, 2, 3$.

Estas instancias cumplen con las siguientes reglas:

- Torneos *Double round robin*, con n equipos pares que generan $2 \cdot n - 2$ casillas.
- La data contiene matrices de distancia cuadradas y simétricas.

7. Resultados

Luego de analizar los experimentos se realiza una comparación entre los resultados entregados por las demás investigaciones. Se realizaron 5 iteraciones para cada instancia y para cada K, para obtener varios resultados de cada instancia y comprender de mejor forma el rendimineto del algoritmo.

NLX	K	Perez - Riff	Montero - Riff	Brandão - Pedroso	Este documento
NL4	$K = 1$	8160	8160	8160	8160
	$K = 2$	8160	8160	8160	8160
	$K = 3$	8044	8044	8044	8044
NL6	$K = 1$	23124	23124	23124	23124
	$K = 2$	22557	22557	22557	22557
	$K = 3$	22557	22557	-	22557
NL8	$K = 1$	39128	39142	39128	39128
	$K = 2$	38761	38845	38761	38761
	$K = 3$	38680	38859	38670	38680
NL10	$K = 1$	59425	58825	-	58825
	$K = 2$	59373	60477	-	59528
	$K = 3$	59582	58834	-	60330
NL12	$K = 1$	117680	119570	-	117680
	$K = 2$	119067	118037	-	119582
	$K = 3$	116082	116942	-	116532
NL14	$K = 1$	209616	208823	-	208823
	$K = 2$	209137	209331	-	209525
	$K = 3$	205690	206134	-	206134
NL16	$K = 1$	307125	300531	-	-
	$K = 2$	300188	293575	-	-
	$K = 3$	297426	284982	-	-

Cuadro 6: Comparación de métodos de los estados del arte con el método de este paper.

En el cuadro 6 se muestran los distintos valores de la función objetivo que tuvieron los distintos autores que resolvieron el RTTP, comparando sus resultados con los obtenidos en este trabajo. Los resultados de NL16 no se pudieron obtener debido al tiempo que demoró. Sin embargo, para los demás experimentos el algoritmo encontró óptimos similares a los del estado del arte.

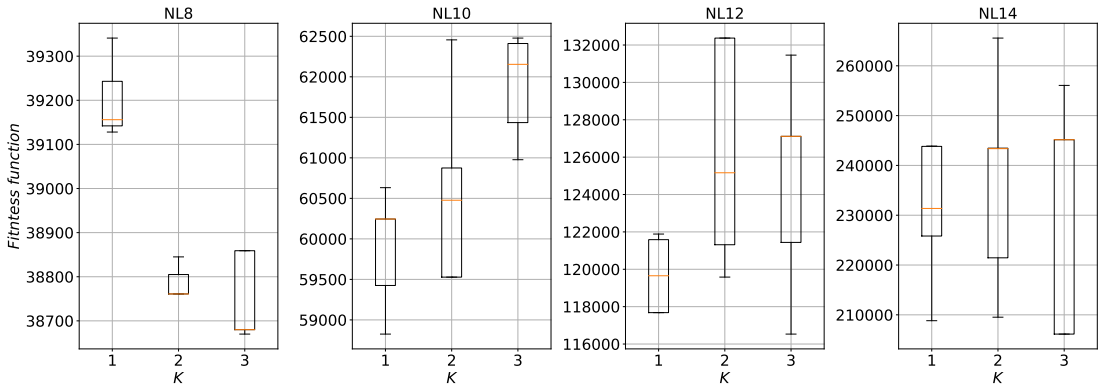

Figura 2: *Boxplots* de la función objetivo de cada instancia para cada K

NLX	K	*Best*	*Average*	*Worst*
NL8	$K = 1$	39128	39202	39341
	$K = 2$	38761	38786.6	38845
	$K = 3$	38670	38749.6	38859
NL10	$K = 1$	58825	59874.4	60632
	$K = 2$	59528	60572.8	62456
	$K = 3$	60977	61891	62478
NL12	$K = 1$	117680	119696.6	121885
	$K = 2$	119582	126160.2	132371
	$K = 3$	116532	124731.4	131459
NL14	$K = 1$	208823	230742.4	243882
	$K = 2$	209525	236674	265566
	$K = 3$	206134	231729.4	256065

Cuadro 7: Calidades de la solución según mejor, peor y calidad promedio.

En la figura 2 junto con el cuadro 7 se muestra el comportamiento que tuvo el algoritmo en las distintas instancias. Podemos observar que para $K = 3$ la calidad de la solución es más óptima comparada con $K = [1, 2]$. Esto se debe a la mayor holgura que tiene el algoritmo para poner *byes* en las casillas.

8. Conclusiones y trabajo futuro

Se logró integrar el algoritmo de hormigas al *Relaxed Traveling Tournament problem*, pudiendo integrar correctamente las matrices de feromonas y de visibilidad al problema. Por otra parte, esta variante del problema según [2] era más complicada y extensa de resolver debido a la inclusión de los *byes* en el problema, lo que hace que el espacio de búsqueda aumente y por consecuencia, que sea más costoso computacionalmente obtener un resultado, sin embargo, fue un desafió que ACO pudo resolver bastante bien. Además, se comprobó que el algoritmo de búsqueda local *Hill Climging* con *restarts* es un buen método a incorporar para optimizar la planificación de cada hormiga. El algoritmo fue probado con distintas instancias del CSPLib y con distintos valores de K, obteniendo resultados óptimos y comparables con el estado el arte de otros autores, demostrando así que el algoritmo *Ant Colony System* puede usarse para obtener resultados óptimos para RTTP.
Como trabajo futuro se podría variar el método de búsqueda local a uno más elaborado o más

veloz, con el objetivo de disminuir el tiempo de ejecución del algoritmo, o mejorar la calidad de la solución para instancias mayores. Otra variante sería modificar el algoritmo para que las hormigas construya en paralelo sus planificaciones, ya que una hormiga no depende de otra para construir su camino (se tendría que modificar también la matriz de feromonas locales). Además, otro punto importante es la determinación de un oponente para un equipo con *bye* en su casilla, ya que podría determinarse en base a la planificación realizada una mejor manera de asignar que oponente tendrá un día libre, con tal de minimizar el costo del viaje de dicho equipo. Finalmente, sería de gran utilidad usar un sintonizador de parámetros para obtener los mejores parámetros que hagan que el algoritmo obtenga resultados óptimos con más precisión.

Referencias

[1] A. Anagnostopoulos, L. Michel, P. Van Hentenryck, and Y. Vergados. A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling*, 9(2):177–193, Apr 2006.

[2] Renjun Bao and Michael A Trick. The relaxed traveling tournament problem extended abstract. *PATAT 2010*, page 472.

[3] Filipe Brandão and Joao Pedro Pedroso. A complete search method for the relaxed traveling tournament problem. *EURO Journal on Computational Optimization*, 12 2013.

[4] David C. Uthus, Patricia Riddle, and Hans Guesgen. An ant colony optimization approach to the traveling tournament problem. pages 81–88, 01 2009.

[5] Pai-Chun Chen, Graham Kendall, and Greet Vanden Berghe. An ant based hyper-heuristic for the travelling tournament problem. pages 19–26, 05 2007.

[6] H Crauwels and Dirk Oudheusden. Ant colony optimization and local improvement. 12 2003.

[7] Leandro De Castro and Jon Timmis. *Artificial immune systems: A new computational intelligence approach*. 06 2002.

[8] Rina Dechter. Constraint processing. The Morgan Kaufmann Series in Artificial Intelligence, page ii. Morgan Kaufmann, San Francisco, 2003.

[9] Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[10] Marco Dorigo and Luca Maria Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66, 1997.

[11] Kelly Easton, George Nemhauser, and Michael Trick. The traveling tournament problem description and benchmarks. In *International Conference on Principles and Practice of Constraint Programming*, pages 580–584. Springer, 2001.

[12] Kelly Easton, George Nemhauser, and Michael Trick. Solving the travelling tournament problem: A combined integer programming and constraint programming approach. In Edmund Burke and Patrick De Causmaecker, editors, *Practice and Theory of Automated Timetabling IV*, pages 100–109, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[13] Luca Di Gaspero and Andrea Schaerf. A composite-neighborhood tabu search approach to the traveling tournament problem. *Journal of Heuristics*, 13:189–207, 2007.

[14] Elizabeth Montero María Cristina Riff. Raisttp revisited to solve relaxed travel tournament problem. In Sara Silva, editor, *GECCO 2015 - Proceedings of the 2015 Genetic and Evolutionary Computation Conference*, pages 121–128. Association for Computing Machinery, Inc, 7 2015.

[15] George L. Nemhauser and Michael A. Trick. Scheduling a major college basketball conference. *Oper. Res.*, 46(1):1–8, January 1998.

[16] Leslie Pérez Cáceres and María Riff. New bounds for the relaxed traveling tournament problems using an artificial immune algorithm. pages 873–879, 06 2011.

[17] M. Riff and E. Montero. A new algorithm for reducing metaheuristic design effort. In *2013 IEEE Congress on Evolutionary Computation*, pages 3283–3290, June 2013.

[18] Robert Russell and Janny Leung. Devising a cost effective schedule for a baseball league. *Operations Research*, 42:614–625, 08 1994.

[19] Andrea Schaerf. Scheduling sport tournaments using constraint logic programming. *Constraints*, 4(1):43–65, Feb 1999.

[20] Robert Thomas Campbell and Der-San Chen. A minimum distance basketball scheduling problem. 01 1976.