Inteligencia Artificial

Travelling Umpire Problem

Benjamín Aguilera

27 de noviembre de 2024

Evaluación

Resumen (5 %):	_____
Introducción (5 %):	_____
Definición del Problema (10 %):	_____
Estado del Arte (35 %):	_____
Modelo Matemático (20 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100 %):	_____

Resumen

El presente informe presenta el Problema de los Vendedores Viajeros, mejor conocido como el Travelling Umpire Problem (TUP), el cual consiste en la asignación de árbitros a juegos o partidos de una liga deportiva sujeto a ciertas restricciones y buscando la minimización de la distancia de viaje de cada juez. Además, se analiza el estado del arte con respecto a la resolución de problemas de este tipo, mostrando diferentes enfoques usados en la literatura desde su formulación hasta el presente. El objetivo de este proyecto es profundizar de manera práctica el entendimiento del funcionamiento de diversos algoritmos de búsqueda usados en proyectos de Inteligencia Artificial, y también tener un primer acercamiento al desarrollo de este tipo de programas.

1. Introducción

El Travelling Umpire Problem tiene su origen en las necesidades y obstáculos reales enfrentados por la organización de diversas ligas o torneos de deportes, en específico, los primeros formuladores de este problema se basaron en las dificultades que enfrentaba la Major League Baseball (MLB) al decidir qué partidos se asignan a cada árbitro o grupo de árbitros de su equipo de jueces, teniendo en cuenta la frecuencia con la que un árbitro es asignado a partidos del mismo equipo y frecuencia de visitas a cada sede de la liga. [7] Debido a la alta complejidad de este problema, se le denomina un problema NP-Completo.

El propósito de este informe es presentar el problema y analizar retrospectivamente los esfuerzos en el modelamiento y resolución de este problema que ya han sido explorados por

otros autores, con el fin de sentar las bases para la implementación de una solución propia en los siguientes avances de este proyecto.

En la sección 2 se define el problema, por medio de una breve descripción de su contexto, variables, restricciones y objetivos. En la sección 3 se presenta el estado del arte, donde se exponen las metodologías usadas a traves de los años para lograr una solución óptima para este problema. Por último, en la sección 4, se formula el modelo matemático asociado a este problema.

2. Travelling Umpire Problem

El Travelling Umpire Problem tiene su origen en el Travelling Tournament Problem (TTP) [7] [3], un problema multi-objetivo que consiste en asignar un itinerario de juegos o partidos de un torneo de tipo doble round robin con el objetivo de minimizar distancias recorridas por los equipos, cumplir con los deseos de cada equipo con respecto a sus juegos en casa y fuera de casa, entre otros. [3]

La investigación de este problema deriva en la formulación del Travelling Umpire Problem, el cual consiste en la asignación de N árbitros a los partidos de un torneo de tipo doble round-robin, conformado por $2N$ equipos. Este torneo consiste en $4N - 2$ rondas de ida y vuelta, jugando una vez en cada sede de ambos equipos. [7]

El TUP, fue originalmente formulado como un problema multi-objetivo, sin embargo, Trick y Yildiz [7] y la mayor parte de la literatura asociada al problema, consideran la versión de un sólo objetivo de este problema, en la cuál se busca minimizar la distancia recorrida por cada árbitro.

2.1. Variables

- Grupo de árbitros a asignar.

2.2. Restricciones

1. Cada partido del torneo es oficiado por exáctamente un árbitro.
2. Cada árbitro dirige exáctamente un partido por ronda.
3. Cada árbitro debe visitar la sede local de cada equipo por lo menos una vez.
4. Ningún árbitro debe visitar la misma sede más de una vez en un número específico de rondas consecutivas.
5. Ningún árbitro debe dirigir al mismo equipo más de una vez en un número específico de rondas consecutivas.

2.3. Objetivo

- El objetivo principal de este problema es minimizar la distancia total recorrida por cada árbitro.

3. Estado del Arte

Desde su formulación y a lo largo de los años, se han usado diversos enfoques para lograr una solución eficiente a este problema, siendo los formuladores del problema, Trick y Yildiz [7], los primeros en proponer técnicas para la resolución del mismo.

En primera instancia, Trick *et al* [7] compararon el TUP con otro problema clásico similar, el Travelling Salesman Problem, con la diferencia que el último se puede resolver de manera exacta/óptima para instancias de cientos o miles de variables, mientras que los autores de la publicación muestran que esto no es posible usando programación entera o de restricciones con técnicas de búsqueda completa.

Como resultado de lo expuesto anteriormente, los autores proceden a implementar técnicas de búsqueda incompleta, las cuales sacrifican la obtención de una solución óptima global con el objetivo de obtener soluciones buenas (no óptimas). La principal técnica usada fue una Neighborhood Search guiada por una heurística de matching de tipo greedy con Benders' Cuts (abreviada GBNS). Esta técnica representa el problema como un grafo bipartito en el cual las particiones corresponden a los árbitros y los partidos de cada ronda. Finalmente, se comparan los resultados del uso de programación entera y el uso de GBNS, y queda en evidencia la importante mejora en los tiempos de resolución y la calidad de las soluciones al usar GBNS en comparación a la programación entera. [7]

El año 2011, Trick y Yildiz vuelven a probar el rendimiento de su algoritmo, esta vez ajustando la configuración del solver utilizado para los modelos de programación entera y de restricciones, sin embargo, el uso de GBNS sigue obteniendo mejor rendimiento que estas dos técnicas. [8]

El próximo avance importante fue el uso de un algoritmo genético para resolver el problema, el cuál arrojó muy buenos resultados en muy poco tiempo en comparación con el uso de programación de restricciones y de programación entera. [9]

En el año 2014, de Oliveira *et al* [2] publicaron los resultados de su investigación con respecto a este problema. En este, los autores propusieron un ajuste a la formulación original en programación entera propuesta por Trick y Yildiz [7], y la usaron para implementar una heurística de relajación y fijación de variables y valores, la cuál mejoró la calidad de 24 de un total de 25 mejores soluciones para instancias publicadas en TUP Benchmark [5], a la vez que mejoraron los límites inferiores para estas instancias y por primera vez impusieron límites inferiores para instancias de más de 16 equipos. [2]

Ese mismo año, Wauters *et al* [11] presentan dos algoritmos heurísticos, denominados *Enhanced Iterative Deepening Search with Leaf Node Improvements*, el cual divide el problema y resuelve cada segmento recursivamente y un algoritmo de búsqueda local iterada. Estos métodos lograron mejorar la calidad de las soluciones para instancias medianas.

El siguiente año, Xue *et al* [12] introducen un modelo de flujo de arcos y uno de *set partition*, y basándose en estos, proponen un algoritmo *branch and bound* y un algoritmo *branch and price and cut*. Usando estos algoritmos, los autores lograron, por primera vez, encontrar soluciones óptimas a instancias de 14 equipos, sin embargo, el tiempo de cómputo fue de más de 24 horas.

En 2018, Chandrasekharan *et al* [1] proponen un algoritmo metaheurístico constructivo, el cuál se diseñó con el objetivo de generar soluciones factibles para instancias grandes, objetivo que logra sin problemas. Además, este algoritmo logra mejorar las soluciones de dos instancias de 18 equipos.

4. Modelo Matemático

4.1. Modelo Integer Programming, Trick and Yildiz, 2011

Se presenta el modelo original de programación entera formulado por Trick y Yildiz [8] en su primera publicación sobre el problema, el cuál ha sido usado como base por gran parte de las publicaciones asociadas a este problema.

4.1.1. Parámetros

- $T = \{1, 2, ..., 2N\}$ corresponde al conjunto de equipos

- $S = \{1, 2, ..., 4N - 2\}$ corresponde al conjunto de rondas.
- $U = \{1, 2, ..., N\}$ corresponde al conjunto de árbitros.
- $OPP[s, i] = \begin{cases} j & \text{si el equipo i juega contra el euipo j en la sede de i en la ronda s,} \\ -j & \text{si el equipo i juega contra el equipo j en la sede de j en la ronda s.} \end{cases}$
- d_{ij} = distancia entre la sede i y la sede j.

También se definen las siguientes constantes para lograr mayor legibilidad:

- $n_1 = n - d_1 - 1$
- $n_2 = \lfloor \frac{n}{2} \rfloor - d_2 - 1$
- $N_1 = \{0, ..., n_1\}$
- $N_2 = \{0, ..., n_2\}$

4.1.2. Variables

- $x_{isu} = \begin{cases} 1 & \text{si el partido jugado en la sede } i \in T \text{ en la ronda } s \in S \text{ es asignado al árbitro } u \in U, \\ 0 & \text{en caso contrario.} \end{cases}$
- $z_{ijsu} = \begin{cases} 1 & \text{si el árbitro } u \text{ está en la sede } i \text{ y se mueve a la sede j en la ronda } s + 1, \\ 0 & \text{en caso cnotrario.} \end{cases}$

4.1.3. Función Objetivo

$$\text{Minimizar} \sum_{i \in T} \sum_{j \in T} \sum_{u \in U} \sum_{s \in S : s < |S|} d_{ij} z_{ijsu}$$

4.1.4. Restricciones

$$\sum_{u \in U} x_{isu} = 1, \quad \forall i \in T, s \in S : OPP[s, i] > 0 \tag{1}$$

$$\sum_{i \in T : OPP[s,i] > 0} x_{isu} = 1, \quad \forall s \in S, u \in U \tag{2}$$

$$\sum_{s \in S : OPP[s,i] > 0} x_{isu} \geq 1, \quad \forall i \in T, u \in U \tag{3}$$

$$\sum_{s_1 \in N_1} x_{i(s+s_1)u} \leq 1, \quad \forall i \in T, u \in U, s \in S : s \leq |S| - n_1 \tag{4}$$

$$\sum_{s_2 \in N_2} \left(x_{i(s+s_2)u} + \sum_{k \in T : OPP[s+s_2,k] = i} x_{k(s+s_2)u} \right) \leq 1, \quad \forall i \in T, u \in U, s \in S : s \leq |S| - n_2 \tag{5}$$

La restricción (1) asegura que cada partido reciba un juez.
La restricción (2) asegura que cada juez oficie exactamente 1 partido en cada ronda.

La restricción (3) impone que cada juez vea a cada equipo por lo menos una vez durante el torneo.

La restricción (4) impone que ningún juez debería estar en la sede de un equipo más de una vez en n_1 rondas consecutivas.

La restricción (5) asegura que ningún juez vea al mismo equipo más de una vez en n_2 rondas consecutivas.

4.2. Modelo de programación entera formulado por Toffolo *et al*

Se presenta también el modelo de flujo presentado por Toffolo *et al*[6], el cual es usado en la mayor parte de publicaciones desde su formulación hasta la fecha. Este modelo, representa el problema como un digrafo, agregando también un nodo sumidero o sink.

4.2.1. Parámetros

- d_e: distancia del arco dirigido e.
- $I = \{1, 2, ..., 2n\}$: conjunto de equipos.
- H_i: conjunto de nodos donde el equipo i juega de local.
- $R = \{1, 2, ..., 4n - 2\}$: conjunto de rondas.
- Q'_{ir}: conjunto de nodos del equipo i jugando de local en las rondas $R \cap \{r, ..., r + q_1 - 1\}$.
- Q''_{ir}: conjunto de nodos del equipo i jugando de local o visita en las rondas $R \cap \{r, ..., r + q_2 - 1\}$.
- $U = \{1, 2, ..., n\}$: conjunto de árbitros.

4.2.2. Variables

$$X_{eu} = \begin{cases} 1, & \text{si el arco } e \text{ es seleccionado para el árbitro } u, \\ 0, & \text{en caso contrario.} \end{cases}$$

4.2.3. Función Objetivo

$$\text{Minimizar} \sum_{e \in E} \sum_{u \in U} d_e X_{eu}$$

Esta función minimiza la distancia total recorrida por cada árbitro.

4.2.4. Restricciones

$$\sum_{e \in \delta(j)} \sum_{u \in U} X_{eu} = 1, \quad \forall j \in V \backslash \{fuente, sink\} \tag{1}$$

$$\sum_{e \in \delta(j)} X_{eu} - \sum_{e \in \omega(j)} X_{eu} = \begin{cases} -1, & \text{si } j \text{ es la fuente,} \\ +1, & \text{si j es el sink,} \\ 0, & \forall j \in V \setminus \{fuente, sink\} \end{cases}, \forall u \in U \tag{2}$$

$$\sum_{e \in \delta(H_i)} X_{eu} \geq 1 \quad \forall i \in I, \forall u \in U. \tag{3}$$

$$\sum_{e \in \delta(Q'_{ir})} X_{eu} \leq 1 \quad \forall i \in I, r \in R, u \in U. \tag{4}$$

$$\sum_{e \in \delta(Q''_{ir})} X_{eu} \leq 1 \quad \forall i \in I, r \in R, u \in U. \tag{5}$$

$$X_{eu} \in \{0, 1\} \quad \forall e \in E, \forall u \in U. \tag{6}$$

La restricción (1) impide que se asigne más de un árbitro a un partido.

La restricción (2) representa las restricciones de preservación de flujo, lo cual se asegura de que un árbitro continúe su camino luego de oficiar un pártido.

La restricción (3) se encarga de que cada árbitro visite cada sede por lo menos una vez.

Las ecuaciones (4) y (5) representan las restricciones de tensión o ajuste.

Finalmente, la ecuación (6) se asegura de que la variable de decisión asuma valores binarios.

5. Representación

Se realiza la lectura del archivo de texto que contiene los datos del problema y se obtienen los siguientes datos:

- $nUmpires$(Número de jueces (int))
- $nRounds$(Número de rondas (int)) $= 4 * nUmps - 2$
- $nTeams$(Número de equipos (int)) $= 2 * nUmps$
- $nGamesPerRound$(Número de juegos por ronda (int))
- $distMatrix$(Matriz de distancias (vector de dos dimensiones))
- $oppMatrix$(Matriz de oponentes (vector de dos dimensiones))

Además, se genera una matriz auxiliar que contiene los datos de cada juego. Esta matriz tiene la siguiente estructura:

gamesMat(Matriz de dimensiones R x N x 3, con elementos de la forma Team1, Team2, Sede)

Toda esta información es guardada en una clase denominada TUP para su procesamiento.

Por otro lado, se tiene la clase Solution, la cuál es generada por el algoritmo greedy como solución inicial, o es resultado del algoritmo principal de Simulated Annealing. La clase Solution contiene la siguiente información:

- $problem$(Guarda el problema correspondiente a la solución (TUP*))
- $d1$(Guarda el valor de d_1 (int))
- $d2$(Guarda el valor de d_2 (int))
- $visitsMatrix$(Matriz de asignaciones de sedes (vector de dos dimensiones))
- $distance$(Almacena la distancia total calculada para la asignación de sedes)

Similar a la clase TUP, se crea una matriz de 3 dimensiones que almacena las asignaciones de juegos a cada juez por cada ronda.

La creación de estas dos matrices fue necesaria para el manejo más fácil de los datos del problema.

6. Descripción del Algoritmo

Basándose fuertemente en el trabajo postulado por *Trick et al* en el año 2012[10], se genera una solución inicial haciendo uso de un algoritmo de tipo greedy, el cuál será mejorado mediante la ejecución del algoritmo Simulated Annealing. Por temas de tiempo, se ignoró la restricción (3) durante el desarrollo de este proyecto.

6.1. Heurística Greedy Para La Creación de una Solución Inicial

Para la generación de una solución inicial, se utiliza una heurística constructiva de tipo Greedy, la cuál, por cada ronda del torneo, asigna jueces a partidos bajo el criterio de minimizar el costo de cada asignación en cada ronda.

Este enfoque se apoya de un grafo bipartito, en el que se representan, por un lado los equipos y por el otro los jueces, sobre el cuál se resuelve un problema de perfect matching en cada ronda.

Se calcula el costo de cada arco de la siguiente manera

$$Costo(u,(i,j)) = distancia(k,i) - incentivo(u,i) + penalizacion * infracciones(u,i,t) \quad (1)$$

Donde

- k es la ubicación actual del juez u en la ronda $t-1$.
- $distance(k,i)$ es la distancia entre la sede actual y la sede del equipo i, donde se juega el partido candidato a asignar.
- $incentivo(k,i)$ es un valor positivo que se asigna en caso de que el juez u nunca haya estado en la sede i.
- *penalizacion* es un valor muy grande que se agrega al costo cada vez que ocurre una infracción a una de las restricciones.
- $infracciones$ es la cantidad total de infracciones a las restricciones de arbitraje a equipos y visita de sedes en periodos definidos de tiempo.

6.2. Simulated Annealing

Para la resolución del problema, se implementa el algoritmo de búsqueda incompleta llamado Simulated Annealing. Este algoritmo fue originalmente introducido por *Kirkpatrick et al*[4] en su publicación el año 1983, basándose en el proceso de recocido de acero y la reacción de sus átomos frente a los cambios de temperatura.

En un principio, se tiene una temperatura inicial, una solución inicial, la cual puede ser generada de manera aleatoria o de alguna otra forma, como es el caso de este proyecto. A esta solución, llamada solución actual, se le realiza un movimiento aleatorio y se compara su costo con el de la solución actual. En caso de que la nueva solución sea mejor que la actual, esta siempre se acepta. En caso contrario, donde la solución es peor que la actual, se acepta o rechaza de acuerdo a un criterio de aceptación basado en la temperatura del sistema, la cual va disminuyendo de manera constante durante la ejecución del algoritmo.

La probabilidad de aceptación es proporcional a la temperatura del sistema y la diferencia Δ entre solución actual y la nueva solución.

Como función de evaluación para el algoritmo, se reusó la función presentada en el apartado del algoritmo greedy que genera la solución inicial, debido a que, por temas de tiempo, no fue posible implementar un movimiento que asegurara la factibilidad.

Algorithm 1 Simulated Annealing for TUP

```
1: procedure SimuAnnealing
2:     temp ← Temperatura inicial
3:     alpha ← Coeficiente mediante el cuál va disminuyendo la temperatura
4:     currSol ← Solución inicial obtenida con greedy
5:     bestSol ← Solución inicial obtenida con greedy
6:     minTemp ← Temperatura mínima
7:     maxIters ← Número máximo de iteraciones por temperatura
8:     while temp ¿minTemp do
9:         It ← 0
10:        while It ¡maxIters do
11:            newSol ← randomSwap(currSol)
12:            delta ← newSol,Cost − currSol.cost
13:            if delta < 0 or rand[0,0; 1,0] < exp(−abs(delta/temp)) then
14:                currSol ← newSol
15:            if newSol < bestSol then
16:                bestSol ← newSol
17:            It + +
18:        temp∗ = alpha
19:    ReturnbestSol
```

7. Experimentos

En un principio, se utilizó un método random para la generación de la solución inicial, la cual no era muy eficiente, y generaba soluciones que infringán una gran cantidad de restricciones, y por consecuencia, provocaban un rendimiento muy pobre al ejecutar Simulated Annealing.

7.1. Parametrización del sistema

El funcionamiento de Simulated Annealing puede ser ajustado fácilmente gracias a sus parámetros, que corresponden a:

- *alpha*: Factor por el cuál va disminuyendo la temperatura después de transcurrido el máximo número de iteraciones por valor de temperatura.
- *iters*: Número de iteraciones a realizar por cada valor de temperatura.
- T_0: Valor inicial de temperatura.
- T_{min}: Valor mínimo de temperatura.

Se realizaron experimentos de variación y fijación de estos parámetros, pero no se ha visto mayor impacto en los resultados. Mayor exploración es necesaria.

8. Resultados

En general, el algoritmo funciona correctamente y mejora la solución inicial dada, pero al no ser esta factible en la mayoría de los casos, y al no tener un movimiento que asegure factibilidad, las soluciones entregadas por el algoritmo tienen un buen valor de distancia, pero rara vez son factibles.

Sin embargo, en un momento del desarrollo, el algoritmo entregaba consistentemente soluciones óptimas para un par de instancias, pero la configuración de parámetros que hicieron esto

posible fue perdida al comenzar el proceso de experimentación y no ha podido ser encontrada de nuevo.

9. Conclusiones

Desde su primera publicación, ha quedado en evidencia la dificultad de este problema, lo que ha generado cierto interés en la investigación de este, sin embargo, en la actualidad aún no se tiene un algoritmo que resuelva este problema al nivel de otros problemas similares, tales como el Travelling Salesman Problem.[7]

Algo que llama la atención sobre este problema y sus publicaciones asociadas, es el hecho de que el objetivo a resolver se ha mantenido, fijando el foco en la minimización de las distancias a recorrer por los árbitros.

Durante el transcurso de esta investigación, se han conocido las diversas técnicas que se han ido probando con el fin de resolver este complicado problema, técnicas tales como *Neighborhood Search*, algoritmos genéticos, técnicas *relax-and-fix*, entre muchas otras. En la actualidad, el método más prometedor, y el más reciente con buenos resultados es el presentado por Chandrasekharan *et al*[1], el cuál consiste en un algoritmo metaheurístico constructivo usando el modelo de programación entera propuesto por Toffolo *et al* [6].

Durante el transcurso de la implementación de la solución, quedaron en evidencia las razones del poco avance que han tenido las investigaciones para la resolución óptima de este problema en el último tiempo, ya que este es un problema con una representación de los datos extremadamente compleja, como ya queda en evidencia en la sección 5, donde se ejemplifican las estructuras de datos auxiliares que se han debido ocupar para facilitar un poco el manejo de los datos del problema.

Si bien no ha sido posible la implementación total del solver que se tenía en mente, y teniendo en cuenta el tiempo acotado destinado al desarrollo, los avances son satisfactorios, y se espera continuar avanzando en la investigación y resolución de este interesante problema.

Referencias

[1] Reshma Chirayil Chandrasekharan, Túlio A.M. Toffolo, and Tony Wauters. Analysis of a constructive matheuristic for the traveling umpire problem. *Journal of Quantitative Analysis in Sports*, 15(1):41–57, 2019.

[2] Lucas de Oliveira, Cid C. de Souza, and Tallys Yunes. Improved bounds for the traveling umpire problem: A stronger formulation and a relax-and-fix heuristic. *European Journal of Operational Research*, 236(2):592–600, 2014.

[3] Kelly Easton, George Nemhauser, and Michael Trick. The traveling tournament problem description and benchmarks. In Toby Walsh, editor, *Principles and Practice of Constraint Programming — CP 2001*, pages 580–584, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[4] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[5] Túlio A. M. Toffolo, Tony Wauters, and Michael Trick. An automated benchmark website for the traveling umpire problem, June 2015.

[6] Túlio A.M. Toffolo, Tony Wauters, Sam Van Malderen, and Greet Vanden Berghe. Branch-and-bound with decomposition-based lower bounds for the traveling umpire problem. *European Journal of Operational Research*, 250(3):737–744, 2016.

[7] Michael A. Trick and Hakan Yildiz. Bender's cuts guided large neighborhood search for the traveling umpire problem. In Pascal Van Hentenryck and Laurence Wolsey, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 332–345, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[8] Michael A. Trick and Hakan Yildiz. Benders' cuts guided large neighborhood search for the traveling umpire problem. *Naval Research Logistics (NRL)*, 58(8):771–781, 2011.

[9] Michael A. Trick and Hakan Yildiz. Locally optimized crossover for the traveling umpire problem. *European Journal of Operational Research*, 216(2):286–292, 2012.

[10] Michael A. Trick, Hakan Yildiz, and Tallys Yunes. Scheduling Major League Baseball Umpires and the Traveling Umpire Problem. *INFORMS Journal on Applied Analytics*, 42(3):232–244, 2012.

[11] Tony Wauters, Sam Van Malderen, and Greet Vanden Berghe. Decomposition and local search based methods for the traveling umpire problem. *European Journal of Operational Research*, 238(3):886–898, 2014.

[12] Li Xue, Zhixing Luo, and Andrew Lim. Two exact algorithms for the traveling umpire problem. *European Journal of Operational Research*, 243(3):932–943, 2015.