PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

Resolución del Traveling Tournament Problem mediante algoritmo PSO Discreto y Representación de Cuadrados Latinos

Leandro Francisco Pacheco Vargas

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO CIVIL INFORMÁTICA

Enero 2015

Índice de Contenidos

Resumen

Las Programaciones de las ligas deportivas son una clase de optimización combinatoria, dentro de las cuales se encuentra el *Traveling Tournament Problem*, que aborda las distintas problemáticas de generar una programación deportiva. Esta problemática contiene como parte de sus exigencias la minimización total de las distancias a recorrer por los equipos participantes de la liga dentro de un esquema de programación del tipo Double Round Robin (DRR), para un grupo de *n* equipos, a lo mencionado anteriormente se adhiere algunas restricciones, como no más de tres partidos seguidos de local o visita.

Esta investigación busca encontrar resultados de forma rápida y eficiente, para ello se plantea una nueva representación del problema, la que se denomina "Cuadrados Latinos", en conjunto a lo anterior se utilizara algoritmo PSO como método resolución el cual no ha sido utilizado para resolver ésta problemática en la literatura actual.

Cabe destacar que existen distintas metodologías o técnicas que han sido utilizadas para resolver esta problemática dentro las cuales se pueden distinguir dos tipos de técnicas:

Completas e incompletas, las cuales serán expuestas en el presente documento.

Palabras Claves: TTP, PSO, Cuadrados Latinos, DRR, SRR, Programación.

Abstract

Sports leagues schedules are a class of combinatorial optimization, within which lies the Traveling Tournament Problem, which deals with the different problems of generating a sports programming. This issue contains as part of their demands minimization total distances to travel by the teams of the League within a framework of programming of the type Double Round Robin (DRR), for a group of n teams, the above is adheres some restrictions, such as no more than three straight home games or visit.

This research seeks to find results quickly and efficiently, a new representation is proposed for this problem, which is called "Square Latinos", altogether to the above algorithm PSO as resolution method which has not been used to solve this problem in the current literature is used.

There are different methodologies or techniques that have been used to solve this problem within which one can distinguish two types of techniques:

Complete and incomplete, which will be exhibited in the present document.

Keywords: TTP, PSO, Latin Squares, DRR, SRR, Fixture.

Lista de Figuras

Lista de Tablas

Glosario de Términos

Espacio de Búsqueda: Contiene todas las posibles soluciones, incluyendo aquellas que violan algunas restricciones.

Fixture: Planificación deportiva.

Lista de Abreviaturas o Siglas

ACO: *Ant Colony Optimization,* colonia de hormigas.

CP: Programación con restricciones.

CSP: Programación de satisfacción de restricciones.

DDR: *Double Round Robin.*

EB: Espacio de Búsqueda.

MLB: Liga mayor de Baseball.

PE: Programación entera.

PSO: Particle Swarm Optimization

SA: *Simulated Annealing.*

SRR: *Single Round Robin.*

TS: Tabú Search.

TTP: *Traveling Tournament Problem.*

1 Introducción

La programación de torneos deportivos (Fixture), es una problemática que se ha tratado de resolver en las distintas disciplinas, ya que para estas tiene vital importancia económica. La solución de este implica superar cada restricción particular que pudiese tener algún torneo en específico. Para esto se plantean soluciones las cuales deben satisfacer dichas restricciones y lograr minimizar las distancias totales recorridas por los equipos pertenecientes a la liga, en sus viajes y con esto evitar costos excesivos.

La presente investigación busca encontrar soluciones factibles de realizar y que por lo demás sean óptimas para variados números de equipos (n), con el claro objetivo de reducir la distancia total recorrida y que esta se realice en el menor tiempo posible. Para esto se utilizara los cuadrados latinos como técnica de representación para encontrar los objetivos planteados.

Cabe señalar que esta problemática ya ha sido resuelta para un número reducido de equipos, sin embargo, se hace necesario tratar de reducir los espacios de búsqueda mediante una nueva representación con el fin de disminuir el espacio de búsqueda planteado.

En el presente informe se encuentra el originen de esta problemática, la descripción del problema, las diferentes metodologías utilizadas para su resolución y las diferentes técnicas aplicadas, además de los objetivos que pretende esta investigación.

1.1 Origen del Problema

Al momento de realizar la programación deportiva de una liga se necesita una planificación de manera tal que esta indique de forma clara quien juega con quien, cuando y donde, es en ese momento donde nace la problemática económica, puesto que cada equipo deberá realizar viajes de ida y vuelta a todas las ciudades de los equipos pertenecientes a la liga, además se debe satisfaces las restricciones impuestas por la liga y esto se debe cumplir de igual manera para todos los equipos participantes. Es por esto que se debe realizar la optimización de la planificación de los encuentros a realizar.

Así es como Easton, Trick y Nemhauser en el año 2001, inspirados en las características de la MBL (Mayor Baseball League) de los Estados Unidos, propusieron la problemática del TTP.

2 Objetivos del Proyecto y Plan de Trabajo

En este capítulo se presentaran los objetivos del proyecto.

2.1 Objetivo General

El objetivo de este proyecto es proponer una nueva representación y una nueva resolución a la problemática de la programación (fixture) de torneos deportivos, que no hayan sido utilizados en la literatura actual, y modelarla de manera que esta sea factible.

2.2 Objetivos específicos

- Investigar y recolectar bibliografía de la problemática del TTP y sus métodos de solución.
- Seleccionar el método a utilizar para resolver el problema.
- Adaptar el método de resolución seleccionado.
- Ajustar el método seleccionado.

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.

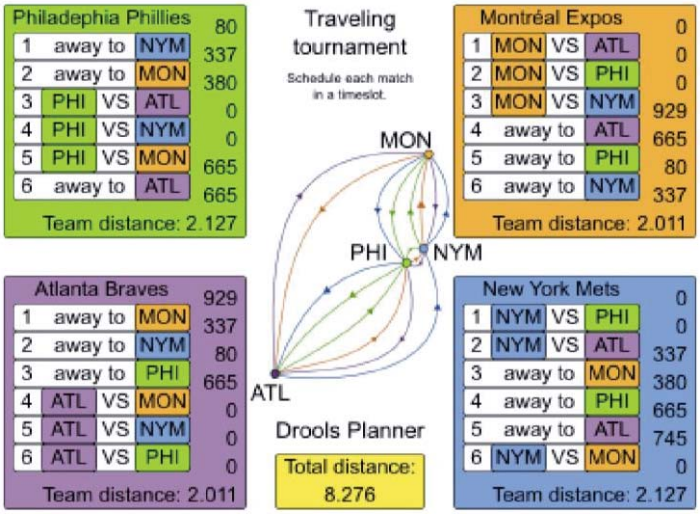

Figura 3.1 Descripción Grafica del TTP

Dada una matriz de distancias DIST de tamaño n x n, el TTP consiste en obtener un fixture Double Round Robin de 2(*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 3.2 Representación matricial para un DRR de 4 equipos.

En definitiva, se puede 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.

Se han desarrollado y utilizado diversas técnicas para intentar resolver el problema. Easton, Nemhauser y Trick [4] intentaron resolver el problema usando *Constraint Programming* y Programación Entera. Benoist, Laburthe y Rottembourgh presentaron un enfoque mixto que combina *Constraint Programming* y Relajación Lagrangiana. Crauwels y Oudheusde probaron utilizando la metaheurística conocida como Colonia de Hormigas. Según el sitio web del *TTP* y lo publicado por Leong, Xingwen Zhang obtuvo algunos resultados interesantes aplicando una combinación de *Constraint Programming*, *Simulated Annealing* y técnicas de *Hill-Climbing*. Anagnostopoulos y otros obtuvieron excelentes resultados utilizando *Simulated Annealing*. Shen y Hantao Zhang 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, quienes también obtuvieron muy buenas soluciones, mejorando incluso algunas de la versión no espejada.

3.1 Definición

Según [4] dados *n* equipos, *n* sea un número par, un torneo en el cual cada equipo juega contra otros en una programación de *Single Round Robin*, se da la siguiente situación:

- El torneo tiene *n-1* casillas en las que se disputan *n*/2 juegos.
- Para cada juego un equipo es local y su oponente el visitante.

En una programación de *Double Round Robin*, se da la siguiente situación:

- El torneo tiene 2(*n*-1) casillas.
- Cada par de equipos se enfrenta 2 veces, alternando su calidad de local y visitante.

Las distancias entre las localidades de los primeros equipos se encuentran representadas en una matriz D (matriz de distancias) cuyas dimensiones son *n x n*, cuando un equipo juega de visitante, se asume que viaja del lugar donde juega de local a la localidad del otro equipo. De otra manera si le tocase viajar nuevamente como visitante se asume que el viaje se realiza desde la localidad del primer equipo rival hacia la localidad del nuevo rival. Cada equipo debe partir y finalizar el torneo en su localidad.

Por consiguiente los desplazamientos son considerados solo cuando el equipo tiene que viajar de visitante. Por ende el desplazamiento total es la suma de todos los viajes de ida y vuelta hacia la localidad, incluyendo los retornos de los equipos que jugaron de visitantes en la última fecha del torneo.

3.2 Formulación

El *Traveling Tournament Problem* se describe a continuación [4]:

Entrada (Input): Equipos y Distancias.

- *n* número par de equipos.
- *D* matriz de distancia *n x n.*
- *L y U* parámetros enteros.

Salida (Output):

- Una matriz planificada con las estructura *Double Round Robin.*
- La distancia total viajada por cada equipo es minimizada, en base a las restricciones.
- La cantidad de juegos Local/Visita consecutivos € [L,U]

Los parámetros L y U hacen referencia a la cantidad de partidos consecutivos que un equipo tiene que jugar de local y como visitante. Tradicionalmente estos toman los valores de L=1 y U=3, de manera que cada planificación pueda tener a lo mas tres juegos consecutivos en el mismo lugar de juego (Visita/Local), además la planificación generada debe ser *espejada* o sin repetidores. 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.

Espejada: Se genera un *Single Round Robin* en las primeras (*n-1*) fechas y en las siguientes (*n*-1) fechas corresponden al mismo SRR anterior, pero con los lugares de los encuentros invertidos. Por consiguiente si el equipo *i* juega de local en la fecha 1 con el equipo *j*, en la fecha *n* jugara de visita con el equipo *j*, tal como se muestra en la tabla 3.1.

Tabla 3.1 Ejemplo de planificación Espejada

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

- **Sin Repetidores:** En fechas continuas no deben darse partidos del tipo *i* con *-j* y luego *j* con *–i*, como se muestra en la tabla 3.2.

Tabla 3.2 Ejemplo de Panificación sin Repetidores

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

La planificación del tipo espejada cumple con la restricción de sin repetidores, por su lógica. El problema debe ser formulado *espejado* o bien sin repetidores.

Adicionalmente a lo formulado anteriormente, cada liga puede tener sus propios requerimientos, los cuales deben ser adicionados como restricciones adicionales al problema.

3.3 Tipos de Representaciones

Actualmente existen variados tipos para modelar el problema de la planificación deportiva (Fixture), por lo que es decisión de la organización de la liga cual usar, además se debe tener en cuenta que el modelo o técnica a utilizar debe satisfacer el tipo de esquema a utilizar.

A continuación se presentaran algunos de los tipos de modelamiento más comunes.

3.3.1 Tablas o Matrices

Una matriz de n(n -1) es la representación más común para fixtures deportivos en donde se presentan todos los partidos que conforma un campeonato de n equipos. Los equipos serán representados con los números del 1 al n, y eventualmente con las n primeras letras del abecedario. Los valores de las celdas Mr,e de una matriz M, representan una planificación de encuentro deportivo indicando el rival con el que juega el e en la ronda r. Para completar la representación, el valor negativo en una celda indica que aquel tipo juega de visita en la ronda r, mientras que por el contrario cuando no existe aquel equipo juega de local. La Tabla 3.3 nos ejemplifica esta situación, donde el equipo A deberá jugar con B en la primera ronda, con B de local, y en la tercera ronda contra D, con D de visitante.

Tabla 3.3 Matriz de Fixture de SRR

Equipos				
Fechas	**A**	**B**	**C**	**D**
1	B	-A	D	-C
2	C	D	-A	-B
3	-D	C	-B	A

3.3.2 Grafos

También se puede utilizar grafos como forma de representación. Aquí explicaremos como hacer corresponder un coloreo de arcos y una 1factorizacion de un grafo completo Kn con un fixture de n equipos.

Coloreo de arcos

Un grafo completo de n nodos G = (V,E) y un coloreo valido de sus arcos también representan un fixture. Dado que G es un grafo completo y n es par, es fácil ver que el coloreo de sus arcos necesita n−1 colores. Supongamos que se utilizan los colores 1, 2..., n − 1. Entonces para armar un fixture a partir de dicho coloreo, lo que se hace es lo siguiente: Para determinar los partidos de la ronda i (recordemos que un fixture SRR tiene n − 1 rondas), se toman todos los arcos pintados con el color i. Cada uno de los arcos representa un partido. Por ejemplo, el arco que une los nodos u y v indica que el equipo representado por el nodo u juega contra el equipo representado por v. En la figura 2.3, se puede ver un coloreo del grafo K4 que representa el mismo fixture que las figuras anteriores.

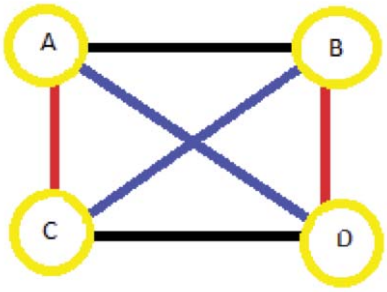

Figura 3.3: Grafo Como Representación del TTP

4 Técnicas de Resolución

La solución del problema se ha planteado desde dos perspectivas. Por un lado utilizando técnica completas (algoritmos más complejos) y por otro las técnicas incompletas (algoritmos más simples), ambas metodologías se presentan con mayor detalle a continuación.

4.1 Espacios de Búsqueda

El espacio de búsqueda en un problema de optimización, se refiere al conjunto de todas las posibles soluciones candidatas, este término no siempre se refiere a una solución factible, por ejemplo, en un problema de satisfacción de restricciones, encontrar una combinación de variables tal que todas las restricciones sean satisfechas es el objetivo del problema, Por esta razón el E.B está constituido por soluciones que violan alguna restricciones, otra forma de definir este concepto, se especifica como el dominio de la función a optimizar.

Al momento de modelar un problema se debe buscar la forma en que este espacio sea lo más reducido posible, ya que de eso va a depender el tiempo de ejecución en búsqueda de la solución óptima, obviamente que también influyen otros factores. Existen diferentes formas de explorar un E.B el que va a depender de la técnica utilizada por la resolución la que se detalla a continuación.

4.2 Técnicas Completas

Esta técnica explora en todo el E.B todas las soluciones factibles y busca la mejor de ella, por lo que normalmente encuentra la solución óptima (la mejor de las mejores). El problema de esta técnica se encuentra, en que, el número de soluciones puede ser muy grande, ya que como se planteó anteriormente la cantidad de soluciones posibles crece exponencialmente junto con el número de equipos en juego, lo que produce que la búsqueda sea cada vez más lenta, cabe destacar que el algoritmo tiene un alto grado de complejidad.

Para este tipo de metodología se han aplicados técnicas de programación entera y programación con restricciones para evaluar el problema planteado. Se dice que una combinación de ambas técnicas podría obtener buenos resultados.

4.2.1 Programación entera

Programación entera (PE) es una programación lineal (PL) con restricciones adicionales, en donde los valores de las variables de decisión deben ser enteros.

Existen diferentes tipos de PE:

- **PE Pura:** Todas las variables de decisión tienen valores enteros.
- **PE Mixta (PEM):** Alguna de las variables de decisión tiene valores enteros. Las demás cumplen con la suposición de divisibilidad.
- **PE Binaria (PEB):** Utiliza variable binarias (0 o 1).

La resolución de este problema se obtiene analizando las posibles alternativas de valores enteros de esas variables en un entorno alrededor de la solución óptima considerando las variables reales.

El método más famoso para la PE es el Branch and Bound (ramificar y acotar). Este método parte de la adición de nuevas restricciones para cada variable de decisión (acotar) que al ser evaluado independientemente (ramificar) llega a un óptimo entero. Además existen otros métodos como Branch and Cut y Branch and Price.

4.2.1.1 Branch and Bound

Este método es una estratégica sistemática que reduce el número de combinaciones que se deben examinar.

Inicialmente tenemos un solo nodo denominado nodo raíz, en donde ramificar significa dividir la región de soluciones en dos sub-regiones añadiendo restricciones a cada problema, con esta división van naciendo los nodos adicionales los que van aumentando a medida que avanza la búsqueda. Se ramifica siempre que la solución no este entera ramificando el problema será posible llegar a una solución entera mejor que la actual candidata.

¿Cómo se ramifica? Eligiendo al azar una variable, que teniendo que ser entera, no toma un valor entero en la solución actual.

4.2.2 Programación con Restricciones

Programación con restricciones (CP) es una tecnología de software que da una descripción clara y eficaz de problemas difíciles de resolver, particularmente los de tipos combinatorios es áreas de planificación y programación de tareas (calendarización). Se enfoca en el estudio de sistemas computacionales basados en la especificación de un conjunto de restricciones, las que deben ser satisfechas por cualquier solución del problema, en lugar de especificar los pasos para obtener dicha solución.

El enfoque primordial de la CP se basa buscar un estado en el que una gran cantidad de restricciones sean satisfechas simultáneamente. Un problema se define típicamente como un estado de la realidad en el que existe un número de variables con valores desconocidos. Un programa basado en restricciones busca dichos valores para todas las variables.

Dentro de la CP existen diferentes tipos de dominios los que pueden ser booleanos (restricciones del tipo verdadero/falso), lineales (describen y analizan funciones lineales), finitos (las restricciones son definidas finitos) y mixtos (involucra dos o más tipos de dominios), también a éstos se adhieren los dominios con variables enteras y racionales.

La ventaja de los métodos de CP radica en el uso activo de restricciones para podar el E.B., eliminando combinaciones de valores que no pueden aparecer juntas en una solución.

4.3 Técnicas Incompletas

Estas técnicas pueden considerarse como adaptativas ya que ellas comienzan su búsqueda en un punto aleatorio del espacio de búsqueda y la modifican repetidamente utilizando algunas heurísticas hasta que alcanzan la solución (con un cierto número de iteraciones). Los métodos, generalmente, son robustos y buenos para encontrar un mínimo global en espacios de búsquedas grandes y complejos, además tienen un aspecto aleatorio que reduce la oportunidad para converger al mínimo local. Sin embargo, pueden ocurrir situaciones donde el proceso de búsqueda se encuentra en partes erróneas del espacio de solución. Esto generalmente requiere que el sistema reinicie su ejecución empezando en otro punto de partida aleatorio.

Se puede inferir que las técnicas incompletas no buscan una solución de todo el espacio de posibles soluciones, por lo que, no asegura obtener la mejor solución; es decir la solución óptima como si lo hacen las técnicas completas. Estos algoritmos deben combinar una buena exploración con una buena explotación.

Algunos Ejemplos de este tipo de técnicas son las siguientes: *ant colony optomization (ACO), tabu search (TS), Clusterning search, simulated annealin, algoritmos genéticos y grasp.*

4.3.1 Ant Colony Optimization (Colonia de Hormigas)

Ant colony optimization (ACO) estudia los sistemas artificiales de hormigas inspirados en la conducta colectiva de hormigas reales, se utiliza para resolver problemas de optimización combinatorias.

En el mundo natural, las hormigas (inicialmente) vagan de manera aleatoria, al azar, y una vez encontrada comida regresan a su colonia dejando un rastro de feromonas. Si otras hormigas encuentran dicho rastro, es probable que estas no sigan caminando aleatoriamente, puede que estas sigan el rastro de feromonas, regresando y reforzándolo si estas encuentran comida finalmente.

Sin embargo, al paso del tiempo el rastro de feromonas comienza a evaporarse, reduciéndose así su fuerza de atracción. Cuanto más tiempo le tome a una hormiga viajar por el camino y regresar de vuelta otra vez, más tiempo tienen las feromonas para evaporarse. Un camino corto, en comparación, es marchado más frecuentemente, y por lo tanto la densidad de feromonas se hace más grande en caminos cortos que en los largos. La evaporación de feromonas también tiene la ventaja de evitar convergencias a óptimos locales. Si no hubiese evaporación en absoluto, los caminos elegidos por la primera hormiga tenderían a ser excesivamente atractivos para las siguientes hormigas. En este caso, el espacio de búsqueda de soluciones sería limitado.

Por tanto, cuando una hormiga encuentra un buen camino entre la colonia y la fuente de comida, hay más posibilidades de que otras hormigas sigan este camino y con una retroalimentación positiva se conduce finalmente a todas las hormigas a un solo camino. La idea del algoritmo colonia de hormigas es imitar este comportamiento con "hormigas simulada" caminando a través de un grafo que representa el problema en cuestión.

Para una heurística definida para TTP [3], se usa una colonia de hormigas *n*. Cada hormiga construye un calendario a partir de un equipo diferente. En cada paso en el árbol de búsqueda, se construye, una lista de candidatos de los equipos para el partido del día. En la primera iteración de esta lista ordenada de acuerdo a las distancias entre el equipo que se examina a todos sus oponentes, el primer candidato de la lista esta seleccionado en el proceso continúa de forma análoga con el próximo partido para el equipo hasta que una restricción no puede ser satisfecha. A continuación se produce retroceso y el siguiente elemento de la lista de candidatos se ha seleccionado. Este proceso continúa que un horario completo se construye cuando cada hormiga se ha construido su calendario, la feromona de los caminos está establecida, la feromona de la ruta *i y j* se incrementa con un valor dependiendo de la distancia total recorrida cuando el calendario contiene un ruta directa. Desde el equipo *i* equipo *j.* Hay tres situaciones (1) equipo *i* inicia un gira en su ciudad natal y su primer partido en contra del equipo *j,* (2) equipo *i* y *j* son dos consecutivos lejos oponente de un tercer equipo y (3) del equipo *j* pone fin a un gira en su ciudad natal y su último partido de ida es contra el equipo *i.* En las iteraciones

siguientes, el orden en la lista de candidatos no solo se basa en las distancias, sino también en los senderos de la feromonas. De esta manera, la construcción de un calendario no solo es influenciada por los "vecinos más cercanos" sino también por subsecuencias de partidos que forman parte de las buenas soluciones.

4.3.2 Tabú Search

Tabú Search (TS) [1,2] 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.

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

La exploración sensible y responsable, proviene del hecho de considerar que una mala elección estratégica (durante el proceso de búsqueda) provee (en general) mayor información que una buena elección aleatoria. La exploración responsable integra los principios de una búsqueda inteligente, es decir, explotar las características de las soluciones buenas, al mismo tiempo que se exploran regiones prometedoras no visitadas. TS trata de encontrar nuevas y mejores formas de aprovechar los mecanismos asociados a estas dos ideas, convirtiéndose en una metodología para resolución de problemas muy flexible y eficaz, como lo han demostrado numerosas aplicaciones a diversos problemas clásicos de optimización combinatoria y problemas reales.

4.3.3 Simulated annealing

Simulated annealing (SA) es un algoritmo de búsqueda meta-heurística para problemas de optimización global; el objetivo general de este tipo de algoritmos es encontrar una buena aproximación al valor óptimo de una función en un espacio de búsqueda grande. A este valor óptimo se lo denomina "óptimo global"

El nombre e inspiración viene del proceso de recocido del acero y cerámicas, una técnica que consiste en calentar y luego enfriar lentamente el material para variar sus propiedades físicas. El calor causa que los átomos aumenten su energía y que puedan así desplazarse de sus posiciones iniciales (un mínimo local de energía); el enfriamiento lento les da mayores probabilidades de recristalizar en configuraciones con menor energía que la inicial (mínimo global).

En cada iteración, el método de recocido simulado evalúa algunos vecinos del estado actual *s* y probabilísticamente decide entre efectuar una transición a un nuevo estado *s′* o quedarse en el estado *s*. En el ejemplo de recocido de metales descrito arriba, el estado *s* se podría definir en función de la posición de todos los átomos del material en el momento actual; el desplazamiento de un átomo se consideraría como un estado vecino del primero en este ejemplo. Típicamente la comparación entre estados vecinos se repite hasta que se encuentre un estado óptimo que minimice la energía del sistema o hasta que se cumpla cierto tiempo computacional u otras condiciones.

El vecindario de un estado *s* está compuesto por todos los estados a los que se pueda llegar a partir de *s* mediante un cambio en la conformación del sistema. Los estados vecinos son generados mediante métodos de Montecarlo.

El método de evaluación de estados vecinos es fundamental para encontrar una solución óptima global al problema dado. Los algoritmos heurísticos, basados en buscar siempre un estado vecino mejor (con energía más baja) que el actual se detienen en el momento que encuentran un mínimo local de energía. El problema con este método es que no puede asegurar que la solución encontrada sea un óptimo global, pues el espacio de búsqueda explorado no abarca todas las posibles variaciones del sistema.

4.3.4 Algoritmo genéticos

Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico. Son llamados así porque se inspiran en la evolución biológica y su base genético-molecular.

Estos algoritmos hacen evolucionar una población de individuos sometiéndola a acciones aleatorias semejantes a las que actúan en la evolución biológica (mutaciones y recombinaciones genéticas), así como también a una Selección de acuerdo con algún criterio, en función del cual se decide cuáles son los individuos más adaptados, que sobreviven, y cuáles los menos aptos, que son descartados.

Es incluido dentro de los algoritmos evolutivos, que incluyen también las estrategias evolutivas, la programación evolutiva y la programación genética. Dentro de esta última se han logrado avances curiosos: Un algoritmo genético es un método de búsqueda dirigida basada en probabilidad. Bajo una condición muy débil (que el algoritmo mantenga elitismo, es decir, guarde siempre al mejor elemento de la población sin hacerle ningún cambio) se puede demostrar que el algoritmo converge en probabilidad al óptimo. En otras palabras, al aumentar el número de iteraciones, la probabilidad de tener el óptimo en la población tiende a uno.

5 Cuadrados Latinos

Un cuadrado latino es una matriz de *n x n* donde, en cada posición, se debe colocar cualquier valor entre 1 y n, pero con la condición que ninguno aparezca más de una vez en la misma fila o columna.

Los cuadrados latinos son un ejemplo de representación los cuales se construyen a partir de una solución factible inicial, la cual va ir iterando mediante alguna regla especial con el objetivo de ir encontrando nuevas soluciones las cuales sean mejores a la inicialmente utilizada.

Para construir cuadrados latinos existen una serie de pasos los cuales se ejemplifican a continuación para un esquema SRR:

1) Se debe construir un cuadrado latino de (n-1) x (n-1) en el que no se repita ningún valor en la diagonal, como se aplica en la Tabla 5.1.

Tabla 5.1 Cuadrado Latino de (n-1) x (n-1)

1	3	2
3	2	1
2	1	3

2) Añadir una nueva fila y columna copiando los valores de la diagonal, tal como se muestra en la Tabla 5.2.

Tabla 5.2 Cuadrado Latino con una nueva fila y columna

1	**3**	**2**	**1**
3	**2**	**1**	**2**
2	**1**	**3**	**3**
1	**2**	**3**	

3) Finalmente los valores de la diagonal son eliminados, tal como se muestra en la Tabla 5.3.

Tabla 5.3 Cuadrado Latino como representación de planificación.

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

El resultado, es una matriz de n × n cuyas celdas representan el número de ronda en el que se deben enfrentar los distintos equipos. Tal como se muestra en la tabla 5.3 en la fecha 3 el equipo A jugara contra B.

Debido a las propiedades matemáticas que tienen los cuadrados latinos, estos han sido utilizados en varios contextos para analizar distintos aspectos de los fixture.

La Tabla 5.4 muestra la cantidad de cuadrados latinos posibles para cierta cantidad de *n.*

Tamaño del Cuadrado	Nro de Formas típicas	Valor de n!(n-1)!	Número total de cuadrados diferentes
3 x 3	1	12	12
4 x 4	4	144	576
5 x 5	56	2880	161280
6 x 6	9408	86400	812851200

6 Análisis de los cuadrados latinos como una nueva representación

Los Cuadrados Latinos pueden generar muchos fixture, pero no quiere decir que todos sean factibles o más bien dicho óptimos. Por otra parte al ocupar ésta representación se hace mucho más fácil trabajar ya que su estilo matricial, facilita la adaptación al método de resolución a utilizar (PSO), no obstante el EB que se genera es un tanto complejo.

7 Propuesta de Solución

7.1 Espacio de soluciones

Se Notara como F al conjunto de soluciones factibles, que se representaran mediante matrices. Dado que el TTP es un torneo DRR, se tiene que la cantidad R de rondas del torneo es 2 x (n-1) y que la celda Mre toma valores enteros con su módulo entre 1 y n. Asumiremos que un valor positivo en la celda Mre indica que el equipo que juega de local en la ronda r contra el equipo Mre, mientras que un valor negativo (representado por - en las figuras) indica que dicho equipo juega de visitante en la ronda r con el equipo –Mre. 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):

$$Mr,e \neq Ms,e \; \forall r \neq s \; (e = 1 \ldots n; r,s = 1..R) \qquad (1)$$

$$[(i \neq j) => \; |Mr,i| \neq |Mr,j|\;] \; (i,j = 1..n) \qquad (2)$$

$$[Mi,r = j \leftrightarrow Mj,r = -i] \; \forall r : 1 \leq r \leq R \; , (i,j = 1..n) \qquad (3)$$

→ Restricciones particulares para TTP:

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

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

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

(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.

7.2 Solución inicial

Se implementara un algoritmo para generar la solución inicial. El cual genera fixtures en forma aleatoria, pero lo hace siempre siguiendo un mismo "esquema" o "patrón" establecido. El algoritmo genera fixtures válidos para torneos DRR, pero no siempre son válidos para el TTP, además de no cumplir con las características de los cuadrados latinos.

7.3 Función de Evaluación

Se utilizara 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, se define que la matriz DIST de dimensión n x n es la matriz de distancias y, por lo tanto, el valor denotado por DISTij 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 M0,k > 0 y que MR+1,k>0 ($\forall$k: 1,...,n). Luego, definimos a Tkr 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:

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

(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: F→N de la siguiente manera:

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

Donde Ce indica la distancia total recorrida por el equipo e a lo largo de todo el torneo según el fixture x. El valor de Ce (e=1...n) se calcula de esta forma:

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

7.4 Modelo

La forma de modelar el TTP será basándose en un cuadrado latino para su representación, la que puede ser definida de distintas maneras.

En este capítulo, el modelo presentado será resuelto mediante algoritmo PSO.

7.5 Parámetros para los modelos

En esta sección se definen los parámetros propuestos que contiene el modelo propuesto:

- n, número de equipos.
- $Dist_{i,j}$, matriz de distancia entre los equipos.
- U, cantidad máxima de partidos que se pueden jugar en forma consecutiva tanto de local como de visita, en este caso se tomara con un valor de 3.
- R, numero de fechas para crear la planificación, que sería 2(n-1).

7.6 Representación Utilizando Formato de Anagnostopulos

Se utilizara la representación propuesta por Anagnostopulos para el desarrollo de esta problemática [6]. Esta representación se describe en la Tabla 7.1, la cual indica los equipos por números de forma secuencial, cada celda contiene un entero que representa el oponente, en este caso los juegos de visita son negativos y los de local son positivos. Concretamente la matriz representa los equipos y en que ronda les toca enfrentarse a los demás, ya sea de local o de visita.

T/R	1	2	3	4	5	6
1	2	3	4	-2	-3	-4
2	-1	4	3	1	-4	-3
3	4	-1	-2	-4	1	2
4	-3	-2	-1	3	2	1

8 OPTIMIZACIÓN POR ENJAMBRE DE PARTÍCULAS (PSO)

El algoritmo de Optimización por Enjambre de Partículas - PSO, del inglés Particle Swarm Optimization -[Kennedy, 1995] es un algoritmo bioinspirado, que se basa en una analogía con el comportamiento social de ciertas especies. Se reconocen dos influencias básicas de inspiración en PSO:

1. El movimiento de bandadas de aves, en las que cada individuo se desplaza mediante reglas simples de ajuste de su velocidad en función de las observaciones que realiza sobre los individuos próximos en la bandada.
2. Un modelo social, en el que cada partícula representa una creencia, y las influencias entre individuos, la aproximación de unos individuos a otros por difusión de dichas creencias.

En líneas generales, el algoritmo busca encontrar el óptimo global de una función mediante el movimiento de un conjunto de partículas – enjambre - en un espacio definido por el número de parámetros de la función. Cada partícula es una posible solución, es decir, un conjunto de parámetros, que se evalúa calculando el valor de fitness que les correspondería. En dicho movimiento, las partículas utilizan información histórica - el resultado de su exploración pasada -, así como información sobre sus vecinos - otras partículas del enjambre.

Por otro lado, para completar la especificación del algoritmo, hay que definir ciertos parámetros del algoritmo - criterio de parada, tipo de vecindario, valores de algunas constantes, forma de inicialización -. El enjambre comienza disperso por el espacio de búsqueda, pero con el tiempo las partículas convergen hacia una zona reducida del espacio en la que intensifican la búsqueda de la solución.

8.1 Terminología del PSO

Esta heurística se basa en el modelo psico-social de entidades colectivas. PSO posee influencia y aprendizaje social [Kennedy, 2003], que se refleja en cada agente de la entidad. Los individuos - partículas - en la heurística emulan una característica simple: adaptan su comportamiento al de los individuos dentro de su propio vecindario o de la población completa - en inglés, swarm -.

Una partícula se mueve dentro del espacio de búsqueda siguiendo una trayectoria definida por su velocidad, y la memoria de dos buenos valores: el mejor encontrado por ella misma en su pasado y el mejor valor encontrado por alguna partícula de la población. Estos dos valores generan una “atracción” hacia los lugares más prometedores, y a esta atracción o fuerza se la denomina presión social.

8.2 ESTRUCTURA DE UNA PARTÍCULA

La estructura básica de una partícula consta de cinco componentes:

•Valor objetivo o aptitud fitness, representa la calidad de la solución representada por el vector X, obtenido a través del cálculo de la función de evaluación correspondiente al problema específico.

• Un vector V que representa la velocidad de la partícula. Este vector, al igual que, se modifica en cada iteración del algoritmo reflejando así el cambio de dirección que sufre la partícula dentro del espacio de búsqueda.

• pbest, mejor valor de fitness encontrado por la partícula hasta el momento.

• gbest, mejor valor de fitness encontrado por alguna partícula del enjambre - swarm -.Este valor está presente si se utiliza un modelo global, es decir, un modelo en el que los individuos son influenciados por el mejor de toda la población.

• lbest, mejor valor de fitness encontrado en el vecindario de una partícula. Este valor está presente si se utiliza un modelo local, es decir, un modelo en el que los individuos son influenciados por el mejor de un grupo pequeño de individuos - vecindario -. La determinación del vecindario depende de la forma en la que se efectúa el agrupamiento de individuos de la población.

8.3 TRAYECTORIA DE UNA PARTÍCULA

La trayectoria de la partícula dentro del espacio de búsqueda está definida con base en el vector de ubicación X y el V de velocidades, los cuales son sumados para obtener la nueva dirección. Justamente estos cambios de trayectoria son los que determinan la característica principal del algoritmo PSO, ya que a través de los mismos las partículas son forzadas a buscar soluciones en las áreas más promisorias del espacio de búsqueda.

Cabe destacar que si los valores V de no se modificaran, la partícula sólo se movería con pasos uniformes en una única dirección. Más específicamente, en cada iteración del algoritmo, la dirección que tomaría la partícula es modificada considerando el valor de pbest - la atracción hacia la mejor posición alcanzada por la partícula - y el vector gbest - la atracción hacia la mejor posición alcanzada por alguna partícula del cúmulo o lbest si se consideran vecindarios.

Además PSO introduce valores aleatorios de ajuste para las dos últimas atracciones. Estos valores aseguran que el tamaño del paso que realizan las partículas dentro del espacio sea variable, asegurando que las mismas no “caigan en una rutina”, es decir, que no se muevan siempre con el mismo paso. La ilustración 12

ilustra la obtención de la nueva posición de la partícula como consecuencia de la suma de la velocidad y las memorias de los mejores. La influencia relativa de la memoria que lleva asociada cada partícula - pbest - es denominada influencia cognitiva, mientras que la memoria del mejor valor encontrado por alguna partícula de la población - gbest o lbest - es denominada influencia social. Ambas influencias determinan la velocidad de aprendizaje del cúmulo y esto permite determinar con qué rapidez el cúmulo convergerá a la solución.

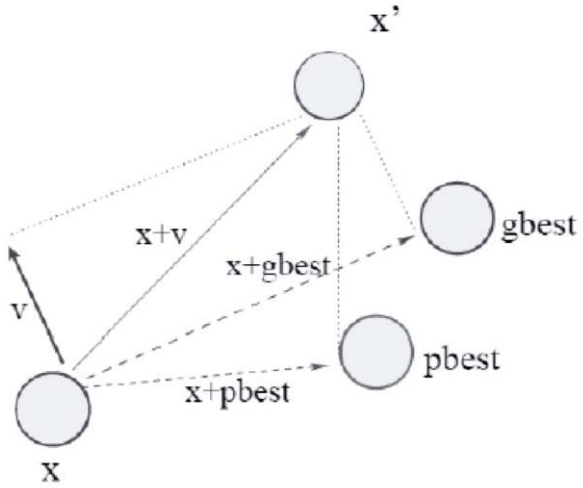

Figura 8.1: Trayectoria Partícula X

Matemáticamente, el proceso de aprendizaje puede describirse utilizando una fórmula para la actualización de la velocidad de cada partícula y otra para la actualización de su posición dentro del espacio de búsqueda.

8.4 ALGORITMO

Considerando un problema de i partículas, el aprendizaje está determinado por las siguientes ecuaciones:

$$v_i = v_i \circ w_1 \otimes (pbest_i \ominus x_i) \circ w_2 \otimes (gbest_i \ominus x_i)$$

$$X_i = x_i \oplus v_i$$

Donde v_i es el valor de la velocidad de la partícula i, w_i es el factor de aprendizaje cognitivo, w_2 es el factor de aprendizaje social, x_i es la posición corriente de la partícula i, $pbest_i$ es el valor de la partícula con el mejor valor objetivo encontrado por la partícula i, $gbest_i$ y es el valor del individuo del enjambre que encontró el mejor valor objetivo. Como puede observarse en la fórmula, un incremento de w_2 sobre w_1 aumenta la influencia del valor gbest, lo cual resulta en una mayor exploración del espacio de soluciones; decrementando el valor de w_2 sobre w_1 causa que la partícula se mueva en dirección más cercana a $pbest_i$, lo cual resulta en una mayor explotación del espacio.

Cuando se habla de exploración se considera la habilidad del algoritmo para explorar las diferentes regiones del espacio de búsqueda a fin de encontrar buenas soluciones. Mientras que la explotación es la habilidad que el algoritmo posee de concentrarse en una porción del espacio que sea promisoria con el fin de refinar soluciones buenas, y encontrar posiblemente el óptimo.

Además de estas fórmulas, un algoritmo PSO cuenta con los siguientes parámetros:

- N: número de partículas involucradas en la resolución del problema.
- D: número de variables del problema, equivalente al número de dimensiones de una partícula.
- Xmax y Xmin: parámetros que limitan el área de búsqueda.
- Vmax y Vmin: parámetros opcionales que limitan la velocidad de las partículas. Son opcionales porque si bien es necesaria una restricción de los valores que la velocidad puede alcanzar, existen otros métodos como el factor de constricción que cumplen con la finalidad de limitar dichos valores.
- Condición de Parada del algoritmo: puede usarse un criterio que establezca un nivel de error aceptable entre el ideal y el óptimo obtenido, una cantidad máxima de iteraciones - ciclos de vuelo - o alguna otra que se prefiera.

Observando el Algoritmo, la fase de Inicialización del enjambre asigna a cada dimensión de cada partícula de la población un valor aleatorio en el rango establecido por [Xmin,Xmax]. Este rango depende exclusivamente del problema que se pretende resolver con la heurística. Lo mismo sucede con los vectores de velocidad - uno por cada partícula -, los que son inicializados en un rango [Vmin ,Vmax]. Los valores objetivos son calculados y almacenados. A cada partícula pbest se le copia el valor de cada dimensión de su individuo correspondiente, al igual que la partícula gbest a la que se le asigna el individuo con mejor valor objetivo. La segunda y más importante es la fase de búsqueda, en la cual el enjambre recorre el espacio para hallar la mejor solución posible. En un proceso repetitivo - hasta que se cumpla una condición de parada -, se actualizan las fórmulas de aprendizaje - velocidad y posición - con base en los valores actuales de pbest y gbest. Los valores de fitness son recalculados y utilizados para remplazar las partículas pbest y gbest, si es que en la presente iteración se obtuvieron mejores valores objetivo.

Algoritmo PSO para permutación de enteros.

S← InicializarCumulo()
While no se alcance la condición de termino do
For i = 1 to size(S) do
Evaluar cada partícula x_i del cúmulo S
If fitness(x_i) es mejor que fitness() then

$$pbest_i \leftarrow x_i \; fitness\,(pbest_i) \leftarrow fitness\,(x_i)$$

End if
If fitness($pbest_i$) es mejor que fitness(g_i) then

$$g_i \leftarrow pbest_i; fitness(g_i) \leftarrow fitness(pbest_i)$$

End if
End For
For i=1 to size (S) do

$$v_i = v_i \circ w_1 \otimes (pbest_i \ominus x_i) \circ w_2 \otimes (gbest_i \ominus x_i)$$

$$X_i = x_i \oplus v_i$$

End For
End while
Salida: La mejor solución encontrada.

8.5 Vecindarios en PSO

Una de las principales características que posee PSO es la interacción social que se observa entre los individuos. Por eso es importante el estudio de las posibles estructuras sociales que pueden incluirse en PSO. Todos los individuos pertenecientes a la misma estructura social comparten información, aprenden y siguen a los mejores dentro de su propia estructura.

La comunicación entre los individuos de un grupo o población así como su desempeño, están íntimamente ligados a la estructura social que posee el grupo. Así también, en PSO, dichas interacciones se observan particularmente entre los vecinos inmediatos adyacentes, y son estas interacciones las que provocan que el algoritmo funcione, ya que cada partícula por sí misma no podría evolucionar demasiado.

La estructura más simple que se puede plantear es aquella donde todos los individuos son influenciados exactamente de la misma forma, por la mejor solución encontrada por algún miembro de la población. Este tipo de modelo recibe el nombre de gbest - mejor global -, y es equivalente a una estructura donde todos los individuos están conectados entre sí. Otra estructura que surge naturalmente recibe el nombre de lbest - mejor local -.

La elección de la estructura social es uno de los principales inconvenientes que presenta la implementación de PSO, debido a la importancia y fuerte influencia que ésta ejerce en el desempeño del algoritmo. El modelo gbest tiende a converger prematuramente, debido a la pérdida de la diversidad. El modelo lbest es más lento de converger, pero conserva la diversidad [Kennedy, 2001]. Los resultados obtenidos con algoritmos que incluyen vecindarios suelen ser, generalmente, mejores cuando son comparados con los obtenidos con modelos gbest.

Varias estructuras lbest pueden ser utilizadas, algunas de estas son [Kennedy, 1999]:

• Circular o Anular: cada individuo está conectado con sus K vecinos inmediatos, intentando imitar al mejor de ellos. En la Figura a puede observarse un ejemplo con K=2 . En este tipo de estructura, cada vecindario se encuentra solapado para facilitar el intercambio de información y, de esta manera, converger a una única solución al final del proceso de búsqueda. La información entre vecindarios fluye lentamente, lo cual provoca que la convergencia sea más lenta pero al mismo tiempo más segura. En funciones multimodales - aquéllas que poseen varios óptimos - este tipo de estructura suele obtener un buen desempeño con respecto a otras estructuras. Muchas variantes de este tipo de estructura pueden ser creadas, en donde la conexión de las partículas puede ser elegida aleatoriamente o siguiendo algún criterio, dependiendo del problema. La Figura b muestra un ejemplo.

• Rueda: un único individuo - el central - está conectado a todos los demás, y todos éstos están conectados solamente al individuo central. La Figura c ilustra el concepto. En esta estructura una única partícula sirve de punto focal, y toda la información entre los individuos es canalizada por ese punto. La partícula focal compara el desempeño de todas las partículas en el vecindario para ajustar las posiciones a la mejor. Tan pronto como la mejor posición sea detectada, la información es enviada a todas las partículas para que efectúen el cambio. La estructura de rueda propaga las buenas soluciones en forma demasiado lenta lo cual puede no ser del todo bueno, porque el proceso de búsqueda se tornaría demasiado lento. Algunas variantes pueden ser introducidas, como desconectar algunas partículas de la partícula focal y conectarlas a otras partículas en el vecindario - Figura d-.

• Arcos Aleatorios: para N partículas, N conexiones simétricas son asignadas entre pares de individuos. Como su nombre lo indica, las conexiones son asignadas de forma aleatoria entre las N partículas. La Figura e muestra esta estructura para un tamaño de población de 12 individuos.

• Pirámide: la idea de esta topología es la creación de una estructura tridimensional que comunique a las partículas. Se asemeja a un modelo de alambre - wire-frame - tridimensional, como se observa en la Figura f. • Toroidal: Esta topología puede utilizarse como estructura de vecindario [Gardner, 1975]. Es una superficie cerrada con base en la topología de malla - Figura g- pero a diferencia de esta última donde cada partícula posee dos vecinos, en la toroidal posee 4 vecinos directos - Figura h-.

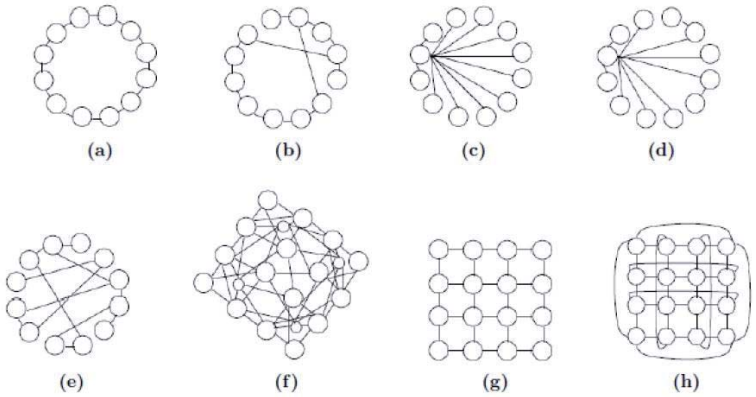

Figura 8.2: TOPOLOGÍAS DE VECINDARIOS DE PARTÍCULAS

Cada una de estas topologías, a su vez, puede variar algunas de sus conexiones, para constituir subvecindarios. Lo importante es lograr una topología que mejore el desempeño del algoritmo. Actualmente, no existe una topología que sea buena para todo tipo de problema. Generalmente, las estructuras completamente conectadas son mejores para problemas unimodales - un único óptimo -, mientras que las menos conectadas funcionan mejor para problemas multimodales, dependiendo del grado de interconexión entre las partículas [Kennedy, 1999] y [Peer, 2003].

9 PSO para permutaciones de Enteros

Los problemas basados en permutaciones de enteros son otro gran subconjunto dentro de los problemas de optimización. Sin ir más lejos, para el problema del viajante de comercio (TSP), bien conocido en la literatura, la forma más usual de codificar las soluciones es mediante este tipo de representación.

9.1 Representación de las Partículas

La posición X de un partícula está formada por una matriz de (n -1) x (n – 1) posibles valores enteros sin que existan repeticiones ni omisiones (donde n es igual a la cantidad de equipos). La posición representa una solución al problema (un cuadrado latino).

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix}$$

La velocidad V está representada por una matriz de pares de enteros$(\mathrm{i} \rightarrow \mathrm{j})$. Cada uno de estos pares representa un intercambio a realizar sobre los elementos de la posición X, dependiendo de la fila en la cual se encuentre. Por ejemplo, si aplicamos el intercambio del par $(1 \rightarrow 3)$ a la primera fila de la posición anterior tendremos la nueva posición.

$$X' = \begin{bmatrix} 3 & 2 & 1 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix}$$

Así, el movimiento de una partícula viene dado por la sucesiva aplicación de pares de la lista velocidad a la lista posición (fila a fila). Un ejemplo de matriz velocidad es el siguiente:

$$V = \begin{bmatrix} (1 \rightarrow 3) & (2 \rightarrow 3) & (3 \rightarrow 1) \\ (2 \rightarrow 3) & (3 \rightarrow 2) & (1 \rightarrow 2) \\ (3 \rightarrow 1) & (1 \rightarrow 3) & (1 \rightarrow 2) \end{bmatrix}$$

Aplicando esta lista V a la posición anterior se obtiene:

$$X' = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 3 & 2 \\ 3 & 2 & 1 \end{bmatrix}$$

9.2 Operador actualización de velocidad

La ecuación empleada para realizar la actualización de la velocidad tiene distinto formato que para las versiones para valores continuos del PSO, es por esto que es necesario describir los operadores de suma, diferencia y multiplicación para permutaciones de enteros.

$$v_i = v_i \circ w_1 \otimes (pbest_i \ominus x_i) \circ w_2 \otimes (gbest_i \ominus x_i)$$

9.2.1 Operador Resta de Posiciones (⊖)

Al restar dos posiciones el resultado es una matriz velocidad, es decir, una matriz de pares. Un par (i →j) está formado por el i-ésimo (i) valor del segundo operando (X2) y el i-ésimo (j) valor de primer operando(X1) de la resta. Por ejemplo, si restamos las siguientes posiciones X1 ⊖ X2:

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix} \ominus \begin{bmatrix} 2 & 1 & 3 \\ 1 & 3 & 2 \\ 3 & 2 & 1 \end{bmatrix}$$

El resultado será la velocidad:

$$V = \begin{bmatrix} (2 \to 1) & (1 \to 2) & (3 \to 3) \\ (1 \to 2) & (3 \to 3) & (2 \to 1) \\ (3 \to 3) & (2 \to 1) & (1 \to 2) \end{bmatrix}$$

9.2.2 Operador Suma de Velocidades (∘)

La suma de dos velocidades consiste ir "solapando" los pares de las listas de dichas velocidades, de manera que se obtiene una nueva lista de pares (se compara el valor final del primer par con el valor inicial del segundo par). Para que se produzca un nuevo par, deben coincidir los valores final e inicial de los pares involucrados en la suma, es decir, los pares (i → j) y (j → k) se solapan formando el par (i → k). Si dos pares correspondientes no se solapan, el resultado será el primer par (i → j). A continuación se muestra un ejemplo completo de suma de velocidades:

$$V = \begin{bmatrix} (2 \to 1) & (1 \to 2) & (3 \to 3) \\ (1 \to 2) & (3 \to 3) & (2 \to 1) \\ (3 \to 3) & (2 \to 1) & (3 \to 1) \end{bmatrix} \circ \begin{bmatrix} (2 \to 2) & (1 \to 3) & (3 \to 1) \\ (1 \to 3) & (3 \to 1) & (2 \to 2) \\ (3 \to 1) & (2 \to 2) & (1 \to 2) \end{bmatrix}$$

Dando como resultado:

$$V = \begin{bmatrix} (2 \to 1) & (1 \to 2) & (3 \to 1) \\ (1 \to 2) & (3 \to 1) & (2 \to 1) \\ (3 \to 1) & (2 \to 1) & (3 \to 2) \end{bmatrix}$$

9.2.3 Operador Producto coeficiente Velocidad (⊗)

Mediante el producto coeficiente velocidad se realiza la multiplicación de un coeficiente real *W* y una matriz velocidad. Se da el siguiente caso:

- Si $W \in [0,1]$,se obtiene un W'= rand(0,1) con $\begin{cases} w' \leq w \to (i,j) \to (i,i) \\ w' > w \to (i,j) \to (i,j) \end{cases}$

Por ejemplo, si multiplicamos (con W = 0,5):

$$0{,}5 \otimes \begin{bmatrix} (2 \to 1) & (1 \to 2) & (3 \to 3) \\ (1 \to 2) & (3 \to 3) & (2 \to 1) \\ (3 \to 3) & (2 \to 1) & (1 \to 2) \end{bmatrix}$$

A continuación se presenta la siguiente secuencia de valores para la matriz:

$$W' = \begin{bmatrix} (0{,}2) & (0{,}8) & (0{,}5) \\ (0{,}5) & (0{,}3) & (0{,}7) \\ (0{,}4) & (0{,}9) & (0{,}1) \end{bmatrix}$$

Se debe comparar cada posición de la matriz W' con el valor del coeficiente W y ejecutar la operación que corresponda según la siguiente lógica $\begin{cases} w' \leq w \rightarrow (i,j) \rightarrow (i,i) \\ w' > w \rightarrow (i,j) \rightarrow (i,j) \end{cases}$, para este ejemplo se obtiene como resultado la siguiente matriz de velocidad:

$$\begin{bmatrix} (2 \rightarrow 2) & (1 \rightarrow 2) & (3 \rightarrow 3) \\ (1 \rightarrow 1) & (3 \rightarrow 3) & (2 \rightarrow 2) \\ (3 \rightarrow 3) & (2 \rightarrow 2) & (1 \rightarrow 2) \end{bmatrix}$$

Nota: El coeficiente W es un valor que puede estar entre 0 y 1, el valor antes presentado solo es de referencia, este se puede ajustar según se desee.

9.3 Operador Movimiento

El movimiento se modela mediante la suma de la velocidad a la posición de la partícula.

$$X_i = x_i \oplus v_i$$

9.3.1 Operador Suma de posición con Velocidad ($\oplus$)

Mediante este operador, se realiza el proceso de permutar los valores de la posición de una partícula para generar una nueva posición. Por cada par (i → j) de la matriz de velocidad, se lleva a cabo en la matriz posición un intercambio de los elementos i y j, fila por fila. El resultado es la nueva posición a la que se mueve la partícula tras la realización de sucesivos intercambios. Por ejemplo, sean V una matriz velocidad y X la posición de la partícula, al sumarlas:

$$\begin{bmatrix} 2 & 1 & 3 \\ 1 & 3 & 2 \\ 3 & 2 & 1 \end{bmatrix} \oplus \begin{bmatrix} (2 \rightarrow 1) & (1 \rightarrow 2) & (3 \rightarrow 1) \\ (1 \rightarrow 2) & (3 \rightarrow 1) & (2 \rightarrow 1) \\ (3 \rightarrow 1) & (2 \rightarrow 1) & (3 \rightarrow 2) \end{bmatrix}$$

Se obtiene como resultado:

$$X = \begin{bmatrix} 2 & 3 & 1 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \end{bmatrix}$$

10 Conclusiones

El TTP es una problemática la cual ha sido abordada variadas técnicas o modelamientos, este se caracteriza por el gran espacio de búsquedas factibles las cuales mediantes combinatorias encontrar mejores posibles soluciones a las utilizadas inicialmente.

El estudio de esta problemática, comenzó con el estudio de las distintas técnicas completas e incompletas y la exposición de los modelos de representación con los cuales pueden ser o ha sido analizado el TTP.

La literatura actual entrega variada información sobre las técnicas completas las cuales aseguran el óptimo, pero el costo que ésta conlleva es altísimo puesto que estas técnicas son complejas y abordan el espacio de soluciones por completo. Por su contraparte están las técnicas incompletas las cuales utilizan algoritmos más simples y mejores para encontrar mínimos globales dentro de los espacios de búsqueda, pero a su vez su gran defecto radica en la reducción de la probabilidad de converger en un mínimo local.

Los métodos de representación ayudan a entender de mejor manera el fixture a realizar, pero el gran espectro de soluciones factible que se pueden dar al menos en los cuadrados latinos los convierte en un proceso complejo.

El algoritmo PSO ayuda a explorar las soluciones factibles y no factibles del TTP, con el algoritmo PSO para permutación de enteros se pueden realizar pruebas a la problemática del TTP, no así con el PSO original el cual bajo ningún caso es aplicable al TTP.

Finalmente se puede establecer gracias a la literatura que el TTP es un problema complejo y que puede ser enfocado de distintas perspectivas de solución, por lo que se considera abierta la posibilidad de continuar investigando, mediante distintos métodos de resolución y/o modelos de representación.

11 Referencias Bibliográficas

[1] Andrés Cardemil, Guillermo Duran, "Un algoritmo Tabú Search para el traveling Tounternament Problem", Revista ingeniería de sistemas, Volumen XVIII, N° 1, pág. 95 -115.

[2] Adres Cardemil, "Opmtimizacion de fixtures deportivos: Estado del Arte y el algoritmo Tabú Search para el traveling tournament problema", Tesis de Licenciatura, Departamento de Computación, Universidad de Buenos Aires, diciembre 2002.

[3] Benjamin Barán, Marta Almiron, "Colonia de Hormigas en un Ambiente Paralelo Asíncrono", Centro Nacional de Computación, Universidad de Asunción.

[4] Kelly Easton, George Nemhauser, and Michael Trick, "The Traveling Tournament Problem Description and Benchmarks", School of industrial and System Engineering, Georgia Institute of Tecnology, Atlanta, Georgia, USA.

[5] Michael Trick, "Formulations and Reformulations in Integer Programming", Tepper School of Business, Carnegie Mellon, Pittsburgh, PA, USA.

[6] A. Anagnostopulos, L. Michel, P. Van Hentenryck and Y, Vergados, "A Simulated Annealing Approach to the Traveling Tournament Problem", Journal of Scheduling, pág. 2, 2006.

[7] Garcia Nieto, Jose Manuel, "Algoritmos Basados en Cúmulos de Partículas Para la Resolución de Problemas Complejos", pág. 43, 2006.