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

RESOLUCIÓN DEL TRAVELING TOURNAMENT PROBLEM UTILIZANDO SIMULATED ANNEALING

SANDRA NICOLE BUSTAMANTE ARRIAGADA

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

DICIEMBRE 2010

Pontificia Universidad Católica de Valparaíso
Facultad de Ingeniería
Escuela de Ingeniería Informática

RESOLUCIÓN DEL TRAVELING TOURNAMENT PROBLEM UTILIZANDO SIMULATED ANNEALING

SANDRA NICOLE BUSTAMANTE ARRIAGADA

Profesor Guía: **Rodrigo Alfaro Arancibia**

Profesor Co-referente: **Broderick Crawford Labrín**

Carrera: **Ingeniería Civil en Informática**

Diciembre 2010

Dedicado a mis padres,

por todo el amor y apoyo entregado.

Agradecimientos

A mis padres y a mi hermano, por todo el apoyo, amor y confianza, que me han brindado durante todos estos años para lograr conseguir este gran sueño.

A mis primos, a mis amigos y toda mi familia, les agradezco el haberme entregado todo el apoyo, colaboración, ánimo y sobre todo cariño y amistad.

Dar gracias a Dios, por darme la oportunidad de llegar a este momento tan especial y dichoso, y por haber puesto a personas tan maravillosas en mi vida.

Al profesor Rodrigo Alfaro por la orientación y apoyo entregado, que han permitido obtener este importante logro.

Finalmente, agradecer a la Universidad por formarme como profesional con valores que me han servido para crecer como persona.

Índice

Resumen

El siguiente proyecto consiste en la resolución del Traveling Tournament Problem, utilizando técnicas incompletas, para ello se investigó, analizó e implementó el algoritmo Simulated Annealing, con el fin de obtener mejores resultados.

En primer lugar, se hace una descripción general de la problemática, la cual corresponde a la planificación de la programación deportiva para los torneos, que realizan las organizaciones deportivas, la cual es una tarea muy compleja. Para realizarla se debe combinar la factibilidad con la optimización, es decir, consiste en minimizar las distancias recorridas por los equipos, pero siempre cumpliendo las condiciones del torneo.

Luego, se explican brevemente las representaciones más utilizadas en la literatura y de manera más detallada aquellas que se utilizarán. Después se considera la búsqueda local y la exploración de vecindarios de soluciones, los cuales comprenden cinco tipos de movimientos que ayudarán a llegar a buenos resultados. Además, estos movimientos son de gran importancia para el algoritmo a utilizar.

Enseguida, se presenta el algoritmo Simulated Annealing para la resolución del Traveling Tournament Problem, con el cual se espera obtener resultados óptimos. Se presenta el diseño del algoritmo, que considera la función objetivo de costo, la estrategia de oscilación del algoritmo, la forma en cómo se obtendrá la programación inicial, la cual se lleva a cabo usando el método del polígono y la asignación de localías.

Finalmente, se muestra un análisis de los resultados obtenidos utilizando el algoritmo con dos representaciones, con sus respectivos gráficos de costos y de temperatura.

Palabras-clave: Simulated Annealing, técnicas incompletas, Traveling Tournament Problem, vecindarios.

Abstract

This project consists in solving the Traveling Tournament Problem using incomplete techniques. Simulated Annealing algorithm was investigated, analyzed and implemented in order to obtain satisfactory results.

First of all, it begins with a general description of the problem, which corresponds to planning the matches of the sport programming that the sport organizations make, which is a very complex task. In order to perform it, the feasibility with the optimization is due to combine, i.e., it consists of decrease the distances that the teams must cross, but always satisfying the constraints of the tournament.

Then, the representations most used in literature are briefly explained, and the ones that will be actually used are explained in detail. Then we consider the local search and exploration of neighborhoods of solutions which include five types of movements that will help to accomplish good results. Besides, these movements are of great importance for the algorithm to use.

Afterward, the Simulated Annealing algorithm is presented for the resolution of the Traveling Tournament Problem, with which is expected to obtain optimal results. We present the design of the algorithm, which considers the cost objective function, the oscillation strategy and the way to obtain the initial programming, which is carried out using the polygon method and the allocation of homecourt.

Finally, the outputs of this project are an analysis of the results obtained using the algorithm with two representations, with their respective graphs of costs and temperature.

Keywords: *incomplete techniques, neighborhood, Simulated Annealing, Traveling Tournament Problem.*

Lista de Figuras

Lista de Tablas

1 Introducción

Los costos de traslado de los equipos deportivos son un gran problema para los organismos o instituciones deportivas de cualquier tamaño, debido a que los jugadores no pueden viajar en malas condiciones, pues son los activos más importantes del club, ya que de su desempeño depende en gran parte la institución. Sin embargo, el traslado es extremadamente costoso aun para las grandes instituciones, pues deben asumir y decidir gastos como buses idealmente sillón cama, o si es en avión o tren no pueden ir en clase turista, siendo obligatoria al menos una clase ejecutiva y tener servicios propios de buses, avión o tren privado.

Debido a esta realidad las federaciones deportivas deben programar su temporada de una forma tal que permita a los equipos ahorrar dinero en sus traslados y permitir así, a los equipos, tener en mejor forma a sus jugadores y, de ser posible, comprar o retener mejores jugadores para aumentar la competitividad. Ahora bien, es posible encontrar varios ejemplos (algunos más claros que otros) para solucionar este problema, como la división entre conferencias en la NBA (este y oeste) o la división entre norte y sur en el torneo de primera B del fútbol chileno. Pero aun con estas divisiones los costos siguen siendo elevados, debido a las distancias que se recorren, puesto que los precios de traslado son proporcionales según cada país.

Es por esto que el problema de cómo armar la programación de los torneos, no es algo que se deba mirar a la ligera, ya que muchos equipos pequeños pueden verse muy afectados por los costos que implica el traslado de su equipo de una ciudad a otra, lo cual los haría menos competitivos y por ende el torneo en si perdería la atracción necesaria, lo que repercute en menos público y menos ingresos, por lo que el deporte en sí no podría sustentarse por sí solo.

Las organizaciones encargadas de administrar y dirigir torneos deportivos tienen la responsabilidad de confeccionar calendarios adecuados, para satisfacer diversos tipos de requisitos y buscando optimizar aspectos como disminución de distancias que deben recorrer los equipos a lo largo de un torneo. La gran diversidad de requerimientos y condiciones que se pueden presentar y que se deben considerar, hace que esta sea una tarea muy compleja.

Existen aún ligas deportivas que organizan la programación deportiva de forma “manual”, pero ha ido en aumento el interés de desarrollar la programación utilizando métodos matemáticos y computacionales con el fin de obtener de forma más rápida una programación más adecuada. Este es el caso del Traveling Tournament Problem, en adelante TTP, que considera un torneo Round-Robin doble el cual debe cumplir con ciertas restricciones, como no más de tres partidos seguidos de local ni de visita, además se busca minimizar la distancia total de viajes de los equipos. La tarea es difícil de resolver incluso para casos de ocho equipos. Esto ha provocado que el TTP despierte gran interés por el desafío que presenta.

Se han utilizado diversas formas de representar la programación deportiva en este texto se explicarán brevemente algunas y se profundizará en la representación de Anagnostopoulos y además se propone una nueva representación, con el fin de obtener soluciones más rápidas.

Entre las técnicas más usadas están la programación entera, programación con restricciones y métodos heurísticos. En este proyecto se propone para la resolución del TTP utilizar técnicas incompletas, las cuales no buscan la solución en todo el espacio, los algoritmos que utilizan esta técnica deben combinar buenas exploraciones con buenas explotaciones. Algunos algoritmos de técnicas incompletas son: Tabu search, Grasp, algoritmos genéticos, Simulated Annealing (Recocido simulado), entre otros.

Para la resolución del problema se ha escogido el algoritmo Simulated Annealing (SA), el cual contiene características interesantes para la obtención de soluciones de calidad:

- SA separa las limitaciones del torneo y las limitaciones del patrón en restricciones fuertes y débiles, además, explora la programación factible y no factible.
- SA utiliza un vecindario grande del tamaño de O (n^3), donde n es el número de equipos.
- SA incluye una estrategia de oscilación estratégica para equilibrar el tiempo de permanencia en regiones factibles y no factibles.
- SA incorpora el concepto de "recalentamiento" para escapar de mínimos locales con muy bajas temperaturas.

SA se ha aplicado a diversos tipos de problemas, y el caso de TTP no ha sido la excepción. Ha coincidido con la mejor solución encontrada para los casos más pequeños, hasta 8 equipos. Se debe señalar que SA es un algoritmo robusto capaz de atacar los duros problemas de optimización combinatoria con éxito.

Por último, se presentan los resultados obtenidos para dos tipos de representaciones utilizando el algoritmo SA.

2 Definición de Objetivos

2.1 Objetivo General

- Desarrollar un algoritmo que realice una búsqueda incompleta en el espacio de soluciones del Problema del Traveling Tournament Problem.

2.2 Objetivos Específicos

- Investigar y entender el Problema del Traveling Tournament Problem y su representación.
- Investigar y entender sobre el algoritmo Simulated Annealing que realiza búsquedas incompletas.
- Implementar el algoritmo Simulated Annealing para resolver el Traveling Tournament Problem.
- Realizar propuestas innovadoras para mejorar el desempeño del algoritmo.

3 Descripción del Problema

El problema de la programación de calendarios deportivos, es comprendido y cercano por las personas, y más aún cuando las personas son entusiastas y practican algún tipo de deporte, y además, debido a que en la mayoría de las ciudades existen torneos deportivos, como de futbol, básquetbol, vóleibol, tenis, entre otros.

Los torneos deportivos conformados de un conjunto de individuos los cuales forman un equipo, estos equipos tienen la función de participar en partidos enfrentándose entre sí, en fechas determinadas llamadas rondas.

Existen varios tipos de torneos, el más conocido es el Round-Robin, donde cada equipo juega contra todos los otros, una cantidad de veces que sea definida, según esa cantidad se tiene:

- Round-Robin simple cuando la cantidad de veces es uno.
- Round-Robin doble cuando la cantidad de veces es dos.

El problema de la programación deportiva consta de una diversidad de requisitos y condiciones, los cuales deben considerarse, algunos ejemplos de ello puede ser aumentar el atractivo del torneo, de los partidos, ayudar a los equipos económicamente. Esta es la razón por la que existe un gran interés así como también es un gran desafío para ser resuelto.

El problema del Traveling Tournament Problem (TTP) fue presentado por K. Easton, G. Nemhauser y M. Trick, el cual contiene aspectos interesantes sobre la confección de la programación deportiva, teniendo en cuenta restricciones y el objetivo de minimizar las distancias recorridas.

El TTP abstrae las características más destacadas de la Major League Baseball (MLB) de los Estados Unidos. La clave del programa de la MLB es un conflicto entre minimizar las distancias de viaje y las restricciones de factibilidad sobre los patrones de local/visita.

La MLB está dividida en dos partes, una de ellas, la National League (NL), la cual posee 16 equipos ubicados en ambas costas de Estados Unidos y Canadá. Las instancias del NL para los cuales se conocen resultados, se generaron tomando subconjuntos de estos 16 equipos. De esta manera la instancia NL*x* está formada por las distancias entre los primeros *x* equipos. Hasta el momento sólo se conoce el valor óptimo para NL4, NL6 y NL8. [2]

Se dice que un equipo juega de local cuando lo hace en su propio estadio, y que juega de visita en el caso que lo hace en el estadio del oponente.

La programación se compone de *n* equipos, con *n* siendo par, y una matriz simétrica *d* de *n*x*n*, tal que d_{ij} representa la distancia que existe entre las casas de los equipos T_i y T_j, tal que la distancia entre T_i y T_j es igual a la distancia entre T_j y T_i.

Una solución es una programación en el que cada equipo juega en dos ocasiones, es decir, cada uno juega con el otro dos veces, una vez en casa de cada equipo. Este programa se conoce como torneo Round-Robin doble, como se ha mencionado antes. Un torneo Round-Robin doble tiene *2n-2* rondas.

Para una determinada programación *S*, el costo que tiene un equipo se define como la distancia total que tiene que viajar partiendo desde su casa, jugando los juegos programados en *S*, y regresando de nuevo a casa. El costo de una solución se define como la suma del costo de cada equipo.

El objetivo es encontrar una programación con un costo mínimo que cumpla las siguientes dos restricciones:

- Restricciones a-lo-más: esta restricción se refiere a que los equipos no pueden jugar más de tres veces consecutivas de local o de visita.

- Restricciones sin-repetición: esta restricción se refiere a que en un juego de local T_i enfrentándose con T_j no puede ser seguido por un juego de local T_j contra T_i. Es decir, ningún par de equipos pueden enfrentarse en dos rondas consecutivas.

De esta manera, una programación Round-Robin doble se considera factible si se respetan las restricciones de a-lo-más y sin-repetición, y no es viable de otra manera.

El TTP modela aspectos importantes prácticos del problema de la programación deportiva, realizando una mezcla entre factibilidad y optimización. Hay que considerar que no existe un historial muy amplio sobre la resolución de este problema, lo que hace que el análisis del TTP sirva como referencia para comparar y estudiar diversas técnicas que intentan resolver el problema. Este problema está lejos de ser sencillo y fácil de resolver, inclusive para una cantidad pequeña de equipos (n=10) aun no se ha podido obtener la solución óptima.

Existen dos tipos de técnicas, las completas y las incompletas.

Las técnicas completas buscan en todo el espacio posible de soluciones la mejor de ellas. Normalmente encuentra una solución óptima. El problema es que para un número de soluciones muy grandes se hace la búsqueda muy lenta, además de hacer al algoritmo más complejo.

Algunos ejemplos de técnicas completas son:

- Programación con restricciones;
- Programación entera.

Las técnicas incompletas no buscan la solución en todo el espacio, por lo que no se asegura de obtener la mejor solución. Sus algoritmos son más simples, pero deben combinar una buena exploración v/s una buena explotación.

Algunas técnicas incompletas son:

- Tabu Search;
- Algoritmos genéticos;
- Grasp;
- Colonia de hormigas;
- Simulated Annealing.

Se han utilizado diversas técnicas para intentar resolver el problema. Easton, Nemhauser y Trick intentaron resolver el problema usando programación con restricciones y programación entera [7]. Benoist, Laburthe y Rottembourgh presentaron un enfoque mixto que combina programación con restricciones y relajación lagrangiana [11]. Crauwels y Oudheusden probaron utilizando la metaheurística Colonia de hormigas [5]. Cardemil realizó una tesis utilizando Tabu search para resolver el TTP [2]. Schaerf y Di Gaspero usaron Tabu search para el enfoque del TTP [12].

3.1 Modelo Matemático

$$M_{k,T} \neq M_{l,T}\ \forall k \neq l\ (T = 1 \dots n; k, l = 1 \dots r) \tag{3.1}$$

$$\left[(i \neq j) \Longrightarrow |M_{k,i}| \neq |M_{k,j}|\right](i, j = 1 \dots n) \tag{3.2}$$

$$\left[M_{i,k} = j \iff M_{j,k} = -i\right] \forall k\colon 1 \leq k \leq r, (i, j = 1 \dots n) \tag{3.3}$$

$$|M_{k,T}| \neq |M_{k+1,T}| \tag{3.4}$$

$$\left[M_{k,T} > 0 \wedge M_{k+3,T} > 0\right] \Longrightarrow \left[M_{k+1,T} < 0 \vee M_{k+2,T} < 0\right] \tag{3.5}$$

$$\left[M_{k,T} < 0 \wedge M_{k+3,T} < 0\right] \Longrightarrow \left[M_{k+1,T} > 0 \vee M_{k+2,T} > 0\right] \tag{3.6}$$

En la ecuación 3.1, todo equipo juega dos veces (partido y revancha) contra todos los demás.

La ecuación 3.2, representa que en cada ronda, todos juegan contra un equipo distinto.

En la ecuación 3.3, si el equipo 1 juega de local contra el equipo 2, entonces el equipo 2 juega de visita contra el equipo 1.

La ecuación 3.4, representa que ningún par de equipos se enfrentan en dos rondas seguidas.

La ecuación 3.5 representa que ningún equipo juega más de tres partidos seguidos de local.

La ecuación 3.6 representa que ningún equipo juega más de tres partidos seguidos de visita.

3.2 Representaciones

Para analizar el problema de programaciones deportivas, se deben conocer las diversas formas en que estos se pueden representar.

En [13] se plantean 4 formatos para la representación de las soluciones:

1. Formato Trick: se basa en las iniciales de los equipos y representa en las celdas el oponente de equipos que encabeza la columna, indicando si juega de local o visita mediante "@" al inicio del nombre del equipo. En la tabla 3.1 se presenta un ejemplo para 6 equipos.
2. Formato Urrutia: es una simplificación de la de Trick, ya que considera los mismos elementos, pero en formato de texto, no de matriz.
3. Formato Anagnostopoulos: es la representación más utilizada en la literatura. Es una matriz similar a la de Trick, pero transpuesta. En las celdas se indica el equipo oponente mediante su número secuencial, en este caso los juegos de visita son con números negativos y los de local son positivos.
4. Formato Di Gaspero: es una adaptación de la representación de Anagnostopoulos, que considera que el primer equipo es "00" (y no "1"). En la tabla 3.2 se presenta un ejemplo para 6 equipos.

Tabla 3.1. Matriz de solución en formato Trick.

r\T	ATL	NYM	PHI	MON	FLA	PIT
1	FLA	@PIT	@MON	PHI	@ATL	NYM
2	NYM	@ATL	FLA	@PIT	@PHI	MON
3	PIT	@FLA	MON	@PHI	NYM	@ATL
4	@PHI	MON	ATL	@NYM	PIT	@FLA
5	@MON	FLA	@PIT	ATL	@NYM	PHI
6	@PIT	@PHI	NYM	FLA	@MON	ATL
7	PHI	@MON	@ATL	NYM	@PIT	FLA
8	MON	PIT	@FLA	@ATL	PHI	@NYM
9	@NYM	ATL	PIT	@FLA	MON	@PHI
10	@FLA	PHI	@NYM	PIT	ATL	@MON

Tabla 3.2. Matriz de solución en formato Di Gaspero.

T\r	1	2	3	4	5	6	7	8	9	10
0	+05	+03	+01	-05	-03	-01	+02	-04	-02	+04
1	+02	-04	-00	-02	+04	+00	+03	-05	-03	+05
2	-01	+05	+04	+01	-05	-04	-00	+03	+00	-03
3	-04	-00	+05	+04	+00	-05	-01	-02	+01	+02
4	+03	+01	-02	-03	-01	+02	+05	+00	-05	-00
5	-00	-02	-03	+00	+02	+03	-04	+01	+04	-01

3.2.1 Representación de Anagnostopoulos

La representación de Anagnostopoulos es una tabla la cual contiene los oponentes de los equipos. Cada columna corresponde a una ronda (r), y cada fila corresponde a un equipo (*T*). El oponente del equipo T_i en la ronda r_k es dada por el valor absoluto del elemento *(i,k)*, si (*i,k*) es positivo, el juego tiene lugar en casa de T_i, es decir, T_i jugará de local, si (*i,k*) es negativo, T_i jugará de visita en casa de su oponente.

A continuación en la tabla 3.3 se muestra la programación *S* para 6 equipos, y por tanto, para 10 rondas.

Tabla 3.3. Programación S para 6 equipos.

T\r	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

Se asume que los equipos empiezan en su lugar de origen, es decir, en su casa, y al final del torneo deben regresar al mismo. Además, cuando un equipo juega varios partidos seguidos de visita, este viaja del estadio de un oponente a otro, es decir, sin pasar por su lugar de origen.

Para la programación *S*, mostrada en la tabla 3.3, el equipo T_1 tiene la siguiente calendarización: el equipo T_1 juega de local contra T_6 en la ronda 1, luego de visita contra T_2 en la ronda 2, regresando a casa para jugar en la ronda 3 contra T_4, seguidamente con T_3 en la ronda 4, luego de visita contra T_5 en la ronda 5, seguido contra T_4 en la ronda 6, luego contra T_3 en la ronda 7, volviendo a casa para enfrentarse en a T_5 en la ronda 8 y después en la ronda 9 con T_2, finalmente, en la ronda 10 de visita contra T_6 para luego regresar a casa. Ahora, después del análisis anterior, se muestran los costos de viaje de cada equipo:

Para el equipo T_1, los costos de viaje son:

$$d_{12} + d_{21} + d_{15} + d_{54} + d_{43} + d_{31} + d_{16} + d_{61}$$

Para el equipo T_2, los costos de viaje son:

$$d_{23} + d_{36} + d_{62} + d_{24} + d_{41} + d_{15} + d_{52}$$

Los costos de viaje para el equipo T_3 son:

$$d_{34} + d_{43} + d_{31} + d_{13} + d_{32} + d_{23} + d_{36} + d_{65} + d_{53}$$

Los costos de viaje para el equipo T_4 son:

$$d_{41} + d_{15} + d_{52} + d_{24} + d_{46} + d_{63} + d_{34}$$

Los costos de viaje para el equipo T_5 son:

$$d_{52} + d_{23} + d_{35} + d_{56} + d_{64} + d_{41} + d_{15}$$

Los costos de viaje para el equipo T_6 son:

$$d_{61} + d_{14} + d_{45} + d_{56} + d_{63} + d_{36} + d_{62} + d_{26}$$

3.2.2 Nueva Representación

Se propone una nueva representación, la cual se basa en cuadrados latinos que consiste en una matriz de $n \times n$ elementos, donde cada posición se puede colocar cualquier valor entero entre *1* y *n* con la condición de que ninguno aparezca más de una vez en la misma fila o columna.[2]

La nueva representación reduce el espacio de búsqueda y puede ayudar a aumentar la eficiencia de los algoritmos de búsqueda.

Se define M una matriz de soluciones donde cada elemento (*i*, *j*) corresponde a la ronda en la que juega el equipo *Ti* con el equipo *Tj*.

Algunas características de la matriz son las siguientes:

- La diagonal principal de la matriz es 0, debido a que un equipo nunca juega con él mismo.
- Para cada $(k, x) \Longrightarrow k$ es local, $\forall x \neq k$.
- Para cada $(x, k) \Longrightarrow k$ es visita, $\forall x \neq k$.

A continuación en la tabla 3.4 se muestra el mismo ejemplo de la tabla 3.3, pero con el nuevo formato de representación. Esta representación tiene la ventaja de mayor síntesis y menor tamaño del espacio de búsqueda.

Tabla 3.4. Programación S para 6 equipos con la nueva representación

	1	2	3	4	5	6
1	0	9	4	3	8	1
2	2	0	6	5	1	7
3	7	3	0	10	2	5
4	6	8	1	0	7	2
5	5	10	9	4	0	3
6	10	4	8	9	6	0

4 Búsqueda Local

La búsqueda local es la base de muchos métodos usados en problemas de optimización. Se puede ver como un proceso iterativo que empieza en una solución y la mejora realizando modificaciones locales. Básicamente empieza con una solución inicial y busca en su vecindad por una mejor solución [10].

El diseño de la vecindad es crucial para el desempeño de muchos algoritmos.

Para el algoritmo propuesto para el TTP en este proyecto, Simulated Annealing (SA), este parte de una configuración inicial. Su paso básico se mueve desde la actual configuración *c* a una configuración en el vecindario de *c*. SA se basa en cuatro decisiones principales de diseño:

- Las restricciones son separadas en dos tipos: restricciones fuertes, que son aquellas que siempre se cumplen por las configuraciones, en este caso, son las limitadas por el Round-Robin; y las restricciones débiles, son las que pueden o no estar satisfechas, vendrían siendo las a-lo-más y las sin-repetición. De forma simple, todas las configuraciones en la búsqueda representan un torneo Round-Robin doble, que pueden o no infringir las restricciones sin-repetición y a-lo-más. Para conducir la búsqueda hacia soluciones factibles, SA modifica la función objetivo original incluyendo un término de penalización.

- SA se basa en un gran vecindario de tamaño de O (n^3), donde *n* es el número de equipos. Además, estos movimientos pueden afectar a partes importantes de las configuraciones. Por ejemplo, al cambiar la programación para dos equipos, afecta a 4(*n*-2) entradas en una configuración. Además, algunos de los movimientos pueden ser considerados como una forma de cadenas de expulsiones utilizados a menudo por la búsqueda tabú.

- SA ajusta dinámicamente la función objetivo para equilibrar el tiempo dedicado en las regiones factibles y no factibles.

- SA utiliza recalentamiento para escapar de mínimos locales a bajas temperaturas. El recalentamiento aumenta la temperatura nuevamente y divide la búsqueda en varias fases.

4.1 El Vecindario

En lugar de definir un solo vecindario que dirige la búsqueda en el espacio de soluciones, se definirán varios, los cuales se van alternando en distintas etapas del proceso de exploración.

El vecindario de una programación S es el conjunto de las programaciones (factibles o no) la cual puede ser obtenida mediante la aplicación de uno de seis tipos de movimientos, los cuales son: SwapHomes, SwapRounds, SwapTeams, PartialSwapRound, PartialSwapTeams y un nuevo movimiento que se propone.

Cada uno de estos movimientos puede plantearse en las representaciones tradicionales como también en la propuesta. A continuación se describirán los movimientos:

SwapHomes (S, T_i, T_j). Este movimiento intercambia local/visita de los equipos T_i y T_j. En palabras simples, si el equipo T_i juega de local contra el equipo T_j en la ronda r_k, y de visita contra T_j en la ronda r_l, el movimiento SwapHomes (S, T_i, T_j) es el mismo programa S, sólo que ahora el equipo T_i juega de visita contra T_j en la ronda r_k, y de local contra T_j en la ronda r_l.

Considerar la programación S con la representación Anagnostopoulos en la tabla 4.1.

Tabla 4.1. Programación *S* antes del movimiento SwapHomes.

T\r	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

El movimiento SwapHomes (S, T_2, T_4) para los equipos T_2 y T_4, produce la programación que es mostrada en la tabla 4.2.

Tabla 4.2. Programación S generada mediante SwapHomes (S, T_2, T_4).

T\r	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	-4	3	6	4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	2	1	5	-2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

La programación presentada en la tabla 4.1 se plantea a continuación en la tabla 4.3 con la nueva representación.

Tabla 4.3. Programación S antes del movimiento SwapHomes.

	1	2	3	4	5	6
1	0	9	4	3	8	1
2	2	0	6	5	1	7
3	7	3	0	10	2	5
4	6	8	1	0	7	2
5	5	10	9	4	0	3
6	10	4	8	9	6	0

En la tabla 4.4 se presenta la programación obtenida tras aplicar el movimiento SwapHomes en la nueva representación, para los equipos T_2 y T_4.

Tabla 4.4. Programación S generada mediante SwapHomes (S, T_2, T_4).

	1	2	3	4	5	6
1	0	9	4	3	8	1
2	2	0	6	8	1	7
3	7	3	0	10	2	5
4	6	5	1	0	7	2
5	5	10	9	4	0	3
6	10	4	8	9	6	0

SwapRounds (S, r_k, r_l). Este movimiento es un intercambio simple de rondas r_k y r_l. Considerar la programación S de la tabla 4.5.

Tabla 4.5. Programación S antes del movimiento SwapRounds.

T\r	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

El movimiento SwapRounds (S, r_3, r_5), para los rondas r_3 y r_5, produce la programación que es mostrado en la tabla 4.6.

Tabla 4.6. Programación S generada mediante SwapRounds (S, r_3, r_5).

T\r	1	2	3	4	5	6	7	8	9	10
1	6	-2	-5	3	4	-4	-3	5	2	-6
2	5	1	4	-6	-3	3	6	-4	-1	-5
3	-4	5	6	-1	2	-2	1	-6	-5	4
4	3	6	-2	-5	-1	1	5	2	-6	-3
5	-2	-3	1	4	6	-6	-4	-1	3	2
6	-1	-4	-3	2	-5	5	-2	3	4	1

A continuación, en la tabla 4.7, se presenta la programación S con la nueva representación a la que se le aplicará el movimiento SwapRounds.

Tabla 4.7. Programación S antes del movimiento SwapRounds.

	1	2	3	4	5	6
1	0	9	4	3	8	1
2	2	0	6	5	1	7
3	7	3	0	10	2	5
4	6	8	1	0	7	2
5	5	10	9	4	0	3
6	10	4	8	9	6	0

La nueva solución generada mediante SwapRounds (S, r_3, r_5) , para los rondas r_3 y r_5, con la nueva representación produce la programación que se muestra en la tabla 4.8.

Tabla 4.8. Programación S generada mediante SwapRounds (S, r_3, r_5).

	1	2	3	4	5	6
1	0	9	4	5	8	1
2	2	0	6	3	1	7
3	7	5	0	10	2	3
4	6	8	1	0	7	2
5	3	10	9	4	0	5
6	10	4	8	9	6	0

SwapTeams (S, T_i, T_j). Este movimiento cambia la lista de equipos de T_i y T_j, excepto cuando juegan uno contra el otro. Además, del cambio de las líneas de los equipos T_i y T_j, se deben intercambiar los correspondientes adversarios de T_i y T_j, excepto cuando T_i y T_j se encuentran. En la tabla 4.9 se muestra la programación S sin la aplicación del movimiento.

Tabla 4.9. Programación S antes del movimiento SwapTeams.

T\r	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

A continuación en la tabla 4.10 se muestra la programación que genera el movimiento SwapTeams (S, T_2, T_5).

Tabla 4.10. Programación S generada mediante SwapTeams (S, T_2, T_5).

T\r	1	2	3	4	5	6	7	8	9	10
1	6	-5	4	3	-2	-4	-3	2	5	-6
2	5	-3	6	4	1	-6	-4	-1	3	-5
3	-4	2	5	-1	6	-5	1	-6	-2	4
4	3	6	-1	-2	-5	1	2	5	-6	-3
5	-2	1	-3	-6	4	3	6	-4	-1	2
6	-1	-4	-2	5	-3	2	-5	3	4	1

A continuación, en la tabla 4.11, se presenta la programación S con la nueva representación.

Tabla 4.11. Programación *S* antes del movimiento SwapTeams.

	1	2	3	4	5	6
1	0	9	4	3	8	1
2	2	0	6	5	1	7
3	7	3	0	10	2	5
4	6	8	1	0	7	2
5	5	10	9	4	0	3
6	10	4	8	9	6	0

La misma solución presentada en la tabla 4.10 se plantea con la nueva representación en la tabla 4.12.

Tabla 4.12. Programación *S* generada mediante SwapTeams (S, T_2, T_5).

	1	2	3	4	5	6
1	0	8	4	3	9	1
2	5	0	9	4	1	3
3	7	2	0	10	3	5
4	6	7	1	0	8	2
5	2	10	6	5	0	7
6	10	6	8	9	4	0

Los tres movimientos anteriormente descritos no son suficientes para explorar el espacio de búsqueda completo y, como consecuencia, dan lugar a soluciones subóptimas para instancias grandes. Para mejorar estos resultados, se deben considerar dos movimientos más. Estos movimientos no tienen una interpretación evidente como los anteriores, sin embargo, son similares en estructura, agrandan significativamente la vecindad, dando como resultado un espacio de búsqueda mas conectado. Más concretamente, estos movimientos son parcialmente intercambiados, es decir, se intercambian un subconjunto de la programación en las rondas r_i y r_j o un conjunto de la programación para los equipos T_i y T_j. Los beneficios de estos movimientos se deben al hecho de que no son tan globales como los movimientos SwapTeams y SwapRounds. Debido a esto, pueden lograr un equilibrio mejor entre la factibilidad y la optimización mediante la mejora de la factibilidad de una parte de la programación, mientras no se rompa la factibilidad en otro. También, son más globales que el movimiento SwapHomes.

A continuación se describirán dos movimientos más.

PartialSwapRounds (S, T_i, r_k, r_l). Este movimiento considera un equipo T_i y los intercambios de las rondas de juegos r_k y r_l. Luego, el resto de la programación de las rondas r_k y r_l es actualizada para producir un torneo Round-Robin doble. En la tabla 4.13 se muestra la programación *S*.

Tabla 4.13. Programación *S* antes del movimiento PartialSwapRounds.

T\r	1	2	3	4	5	6	7	8	9	10
1	6	-2	2	3	-5	-4	-3	5	4	-6
2	5	1	-1	-5	4	3	6	-4	-6	-3
3	-4	5	4	-1	6	-2	1	-6	-5	2
4	3	6	-3	-6	-2	1	5	2	-1	-5
5	-2	-3	6	2	1	-6	-4	-1	3	4
6	-1	-4	-5	4	-3	5	-2	3	2	1

El movimiento PartialSwapRounds (S, T_2, r_2, r_9), al intercambiar el juego en las rondas r_2 y r_9 no daría lugar a un torneo Round-Robin, por lo que es necesario intercambiar los juegos del equipo T_1, T_4, y T_6 con lo cual se obtiene la programación mostrada en la tabla 4.14.

Tabla 4.14. Programación *S* generada mediante PartialSwapRounds (S, T_2, r_2, r_9).

T\r	1	2	3	4	5	6	7	8	9	10
1	6	4	2	3	-5	-4	-3	5	-2	-6
2	5	-6	-1	-5	4	3	6	-4	1	-3
3	-4	5	4	-1	6	-2	1	-6	-5	2
4	3	-1	-3	-6	-2	1	5	2	6	-5
5	-2	-3	6	2	1	-6	-4	-1	3	4
6	-1	2	-5	4	-3	5	-2	3	-4	1

En la tabla 4.15 se muestra la programación S con la nueva representación a la cual se le aplicará el movimiento PartialSwapRounds.

Tabla 4.15. Programación S antes del movimiento PartialSwapRounds.

	1	2	3	4	5	6
1	0	3	4	9	8	1
2	2	0	6	5	1	7
3	7	10	0	3	2	5
4	6	8	1	0	7	2
5	5	4	9	10	0	3
6	10	9	8	4	6	0

En la tabla 4.16 se muestra la programación S que genera el movimiento PartialSwapRounds (S, T_2, r_2, r_9).

Tabla 4.16. Programación S generada mediante PartialSwapRounds (S, T_2, r_2, r_9).

	1	2	3	4	5	6
1	0	3	4	2	8	1
2	9	0	6	5	1	7
3	7	10	0	3	2	5
4	6	8	1	0	7	9
5	5	4	9	10	0	3
6	10	2	8	4	6	0

PartialSwapTeams (S, T_i, T_j, r_k). Este movimiento considera una ronda r_k y el intercambio de juego de los equipos T_i y T_j. Luego, el resto del programa para los equipos de T_i y T_j (y sus oponentes) se actualizan para producir un torneo Round-Robin doble. Como fue el caso de SwapTeams, cuatro entradas por ronda serán afectadas. En la tabla 4.17 se muestra la programación S a considerar.

Tabla 4.17. Programación S antes del movimiento PartialSwapTeams.

T\r	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

.

El movimiento PartialSwapTeams (S, T_2, T_4, r_9) produce la programación mostrada en la tabla 4.18.

Tabla 4.18. Programación S generada mediante PartialSwapTeams (S, T_2, T_4, r_9).

T\r	1	2	3	4	5	6	7	8	9	10
1	6	-2	2	3	-5	-4	-3	5	4	-6
2	5	1	-1	-5	4	3	6	-4	-6	-3
3	-4	5	4	-1	6	-2	1	-6	-5	2
4	3	6	-3	-6	-2	1	5	2	-1	-5
5	-2	-3	6	2	1	-6	-4	-1	3	4
6	-1	-4	-5	4	-3	5	-2	3	2	1

En la tabla 4.19 se muestra la programación S con la nueva representación, a la cual se le aplicará el movimiento PartialSwapTeams.

Tabla 4.19. Programación S antes del movimiento PartialSwapTeams.

	1	2	3	4	5	6
1	0	9	4	3	8	1
2	2	0	6	5	1	7
3	7	3	0	10	2	5
4	6	8	1	0	7	2
5	5	10	9	4	0	3
6	10	4	8	9	6	0

El movimiento PartialSwapTeams (S, T_2, T_4, r_9) produce la programación mostrada en la tabla 4.20.

Tabla 4.20. Programación *S* generada mediante PartialSwapTeams (S, T_2, T_4, r_9).

	1	2	3	4	5	6
1	0	3	4	9	8	1
2	2	0	6	5	1	7
3	7	10	0	3	2	5
4	6	8	1	0	7	2
5	5	4	9	10	0	3
6	10	9	8	4	6	0

NuevoMovimiento: se propone este movimiento con el fin de provocar un cambio total en la programación. Este movimiento intercambia local/visita de todos los equipos, es decir, todos los equipos que juegan de local lo harán de visita y los de visita pasarán a ser locales. En la tabla 4.21 se muestra la programación *S* a considerar.

Tabla 4.21. Programación S antes del nuevo movimiento.

T\r	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

El nuevo movimiento produce la programación mostrada en la tabla 4.22.

Tabla 4.22. Programación *S* generada mediante el nuevo movimiento.

T\r	1	2	3	4	5	6	7	8	9	10
1	-6	2	-4	-3	5	4	3	-5	-2	6
2	-5	-1	3	6	-4	-3	-6	4	1	5
3	4	-5	-2	1	-6	2	-1	6	5	-4
4	-3	-6	1	5	2	-1	-5	-2	6	3
5	2	3	-6	-4	-1	6	4	1	-3	-2
6	1	4	5	-2	3	-5	2	-3	-4	-1

En la tabla 4.23 se muestra la programación S con la nueva representación a la cual se le aplicará el nuevo movimiento.

Tabla 4.23. Programación S antes del nuevo movimiento.

	1	2	3	4	5	6
1	0	9	4	3	8	1
2	2	0	6	5	1	7
3	7	3	0	10	2	5
4	6	8	1	0	7	2
5	5	10	9	4	0	3
6	10	4	8	9	6	0

El nuevo movimiento produce la programación mostrada en la tabla 4.24.

Tabla 4.24. Programación *S* generada mediante el nuevo movimiento.

	1	2	3	4	5	6
1	0	2	7	6	5	10
2	9	0	3	8	10	4
3	4	6	0	1	9	8
4	3	5	10	0	4	9
5	8	1	2	7	0	6
6	1	7	5	2	3	0

En definitiva, después del análisis de los movimientos más comunes con distintas representaciones se concluye que con la representación propuesta se requiere ejecutar la mitad de las operaciones para construir una nueva solución.

5 Simulated Annealing

Simulated Annealing (SA) o recocido simulado, fue propuesto por Metrópolis [8] y [4], esta metaheurística ha demostrado ser muy eficaz en la resolución de problemas de optimización combinatoria. SA es una variación de la búsqueda local, que permite realizar movimientos para evitar caer en un óptimo local prematuramente.

Se le dio este nombre porque está basado en un proceso para simular el enfriamiento de material (un proceso denominado “recocido”). Este consistía, en la emulación de un proceso físico mediante el cual un sólido se enfría lentamente de modo que cuando finalmente su estructura está "congelada", esto sucede en una configuración de energía mínima.

Se dice que SA es muy fácil de hacer que funcione, pero lo más difícil es lograr que funcione bien, esto se debe a que no es propiamente un algoritmo, sino una heurística que necesita de varias decisiones para que quede bien configurada, y esto afecta significativamente en las soluciones generadas.

Se explicará a continuación el funcionamiento de SA, se describirán algunos de los factores importantes que hay que considerar para obtener una implementación exitosa y finalmente se verá como es aplicado al problema de TTP.

El principal objetivo de SA es evitar el problema de caer en un óptimo local, que es lo que sucede con varios algoritmos tradicionales de búsqueda local, los cuales comienzan de una solución inicial, y que de manera paulatina es transformada en otra y a su vez son mejoradas al introducirles cambios, como modificar el valor de una o varias variables. Si el cambio produce una mejor solución, la actual solución se sustituye por la encontrada, continuando con el proceso hasta que no se logra obtener mejoras, esto provoca que al finalizar la búsqueda se encuentre con un óptimo local y que probablemente no sea global.

Para evitar este problema, se realizan algunos movimientos hacia soluciones peores. Pero, estos movimientos deben realizarse de forma controlada cuando se acerca a soluciones buenas, para disminuir la probabilidad de esos movimientos hacia soluciones peores ya que previsiblemente estamos cerca del óptimo global, para lograr esto se debe controlar la frecuencia de los movimientos a través de una función de probabilidad.

La fundamentación de este control se basa en el trabajo de [9] en el campo de la termodinámica estadística. Básicamente, se modeló el proceso de recocido simulando los cambios energéticos en un sistema de partículas conforme decrece la temperatura, hasta que converge a un estado estable (congelado). Las leyes de la termodinámica dicen que a una temperatura *t* la probabilidad de un incremento energético de magnitud δE se puede aproximar por:

$$P[\delta E] = \exp\left(-\frac{\delta E}{kt}\right) \tag{5.1}$$

Siendo k una constante física denominada de Boltzmann. En el algoritmo de Metrópolis se genera una perturbación aleatoria en el sistema y se calculan los cambios de energía resultantes: si hay una caída energética, el cambio se acepta automáticamente; por el contrario,

si se produce un incremento energético, el cambio será aceptado con una probabilidad dada por la ecuación (5.1). El proceso se repite durante un número predefinido de iteraciones en series decrecientes de temperaturas, hasta que el sistema esté "frío". [6].

A comienzo de los años 80, publicaciones de [8] y [4] revelaron cómo este proceso se podía aplicar a problemas de optimización, estableciendo una relación entre los elementos claves del proceso de simulación y los de optimización. Ver tabla 5.1.

Tabla 5.1. Relación entre elementos de simulación termodinámica y de optimización.

Simulación termodinámica	Optimización combinatoria
Estados del sistema	Soluciones factibles
Energía	Costo
Cambio de estado	Solución en el entorno
Temperatura	Parámetro de control
Estado congelado	Solución heurística

En definitiva, cualquier búsqueda local es posible convertirla en una implementación SA, eligiendo los elementos del entorno de manera aleatoria, aceptando automáticamente los movimientos dirigidos a una mejor solución así como a una solución peor de acuerdo con la probabilidad de la ecuación (5.1). Con respecto a la constante de Boltzmann *k*, ésta no se considera debido a que no tiene un significado en los problemas de optimización.

Con respecto al parámetro *t* que es la temperatura, este es un parámetro de control. Cuando una solución suponga un incremento δ en la función de costo, esta se aceptará con una probabilidad de $exp(-\delta/t)$. Al permitir que *t* alcance valores suficientemente pequeños, se conseguirá que no haya más movimientos a peores soluciones, consiguiendo una convergencia a un óptimo local. Existen algunos estudios que demuestran que si *t* decrece lo suficientemente lento, el proceso converge a la solución óptima.

Para la implementación del algoritmo es necesario considerar algunas decisiones, están las genéricas y las específicas. La primera, se refiere a cómo controlar la temperatura, el número de iteraciones y las condiciones que indican cuando el sistema esta "frio". Mientras la segunda, se refiere al espacio de soluciones y su estructura de entorno, la función de costo y la forma de obtener la solución inicial. Estas decisiones deben ser cuidadosamente tratadas ya que influyen considerablemente en la calidad de las soluciones.

5.1 Decisiones Genéricas

Las decisiones genéricas están relacionadas con las medidas que dirigen el programa de enfriamiento, como los valores que podrá tomar la temperatura máxima y mínima, la velocidad para reducir esta, y las condiciones de parada. Como se ha dicho antes, cuando el enfriamiento se realiza suficientemente lento, el algoritmo convergerá al óptimo global con probabilidad 1, según la temperatura tiende a 0.

Hay que aclarar que la calidad de la solución final debe ser independiente de la solución inicial de la que se comienza, por lo tanto, la temperatura inicial debería ser independiente de la solución inicial y además debería ser suficientemente alta como para aceptar las soluciones del entorno.

Con respecto al factor de velocidad a la cual se produce el enfriamiento, este se determina por el número de iteraciones que se ejecutarán a cada temperatura, y por una velocidad α a la que se realizará el enfriamiento. En relación con el número de iteraciones, este depende del tamaño del problema, y varía cuando la temperatura va descendiendo.

Finalmente, si la temperatura de parada se fija en un valor muy bajo, en las fases finales se gastará mucho tiempo en búsqueda, y que sería mejor aprovechado en temperaturas superiores. Mientras que en el caso de que la temperatura final se fije muy alta, lo más probable es que la búsqueda no alcancé ningún óptimo local.

5.2 Decisiones Específicas

Las decisiones específicas apuntan principalmente a la decisión del espacio de soluciones, la estructura de los entornos y la función de costo, como también a la elección de la solución inicial. Estas decisiones son significativamente importantes ya que ayudan a la calidad de la solución final.

Algunas condiciones que hay que considerar:

La solución inicial debe obtenerse de forma aleatoria.

Cualquier solución debe ser alcanzable desde cualquier otra por medio de una serie de movimientos válidos usando los entornos.

6 Diseño del Algoritmo

6.1 Algoritmo Simulated Annealing

Para diseñar el algoritmo de resolución del TTP se seguirá el esquema tradicional del algoritmo Simulated Annealing. Dada una temperatura *T*, el algoritmo selecciona al azar uno de los movimientos en el vecindario y calcula la variación Δ en la función objetivo producida por el movimiento. Si la variación $\Delta < 0$ se aplica el movimiento. En caso contrario, se aplica el movimiento con una probabilidad $\exp(-\Delta/T)$.

Como es característico en el Simulated Annealing, la probabilidad de aceptar un movimiento que no mejora disminuye con el tiempo. Este comportamiento se obtiene al disminuir la temperatura de la siguiente manera. Se utiliza una variable *contador* que se incrementa por cada movimiento que no mejora y se reinicializa a cero cuando se encuentra una solución mejor a la mejor encontrada hasta el momento. Cuando el *contador* llega a un límite superior en particular, la temperatura se actualiza a $T*\beta$ (donde β es un valor constante inferior a 1) y el *contador* es reinicializado a cero. [1]

A continuación se muestra el seudocódigo del algoritmo Simulated Annealing más detallado.

```
generar programación S aleatoria;
elMejor = costo(S);
contador = 0;
fase=0;
MIENTRAS fase ≤ maxFase HACER
 contador = 0;
 MIENTRAS contador ≤ contadorLimite HACER
       seleccionar un movimiento m aleatorio del vecindario(S);
       sea S' la programación obtenida a partir de S con m;
       SI costo(S') < costo(S) ENTONCES
             aceptar = true;
       SINO
             aceptar = true con una probabilidad  exp (-△/T),
             aceptar = false en otro caso;
       FIN SI
       SI aceptar ENTONCES
             S=S';
             SI costo(S') < elMejor ENTONCES
                   contador = 0;
                   fase = 0;
                   elMejor = costo(S');
             FIN SI
```

```
        SINO
              contador++;
        FIN SI
    FIN MIENTRAS
    fase++;
    T=T*β;
FIN MIENTRAS
```

6.2 Función Objetivo

Las configuraciones en el algoritmo SA corresponden a programaciones que pueden o no satisfacer las restricciones sin-repetición y a-lo-más. Además, los movimientos no siempre garantizan la factibilidad, incluso habiendo comenzado con una programación factible. La capacidad de explorar las programaciones infactibles resulta crítico para el éxito de Simulated Annealing en el TTP.

Para conducir hacia la solución factible, la función objetivo estándar, la función *costo*, se sustituye por una función objetivo más compleja que combina las distancias de viaje y el número de violaciones de la programación.

La nueva función objetivo *C* se define a continuación:

$$C(S) = \begin{cases} costo(S) & si\ S\ es\ factible \\ \sqrt{costo(S)^2 + [w * f(nbv(S))]^2} & otro\ caso \end{cases} \quad (6.1)$$

Donde *nbv*(*S*) indica el número de violaciones de las restricciones sin-repetición y a-lo-más, *w* es un peso, y $f : N \rightarrow N$ es una función sublineal tal que $f(1)=1$.

La elección de la función *f* tiene directa relación con que la primera violación debe ser más costosa que las siguientes violaciones, ya que una violación adicional en una programación con 6 violaciones no marca ninguna diferencia, puesto que las restricciones ya no se estarían cumpliendo. La función *f* escogida se muestra en la Ecuación 6.2.

$$f(v) = 1 + \frac{\sqrt{v} \ln v}{2} \quad (6.2)$$

6.3 Oscilación Estratégica

SA incluye una estrategia de oscilación estratégica que ha sido a menudo utilizada en la búsqueda tabú cuando la búsqueda local explora la región factible y la no factible. La idea clave es la variación del parámetro de peso *w* durante la búsqueda. En aplicaciones avanzadas de búsqueda tabú, la penalidad se actualiza conforme a las frecuencias de las configuraciones factibles y no factibles en las últimas iteraciones. Una estrategia como esa tiene mucho sentido en ese contexto, pero no es particularmente apropiado para el SA, debido a que serian seleccionados muy pocos movimientos. SA utiliza un esquema muy simple. Cada vez que se

genere una nueva mejor solución, SA multiplica *w* por una constante $\delta > 1$ si la nueva solución es infactible o divide *w* por alguna constante $\theta > 1$ si la nueva solución es factible.[1]

La lógica es mantener un equilibrio entre el tiempo dedicado a explorar la región factible y el tiempo dedicado a explorar programaciones infactibles. Después de haber pasado mucho tiempo en la región infactible, el peso w, y por lo tanto la penalización por las violaciones, aumentará e impulsará la búsqueda hacia soluciones factibles. Del mismo modo, después de haber pasado mucho tiempo en la región factible, el peso w, y por lo tanto la penalización por las violaciones, disminuirá lo que impulsará la búsqueda hacia soluciones infactibles. Por simplicidad se consideró $\theta = \delta$.

A continuación se muestra el seudocódigo de SA integrado con la oscilación estratégica:

```
generar programación S aleatoriamente;
mejorFactible = ∞;
nbf = ∞;
mejorInfactible = ∞;
nbi = ∞;
contador = 0;
fase = 0;
MIENTRAS fase ≤ maxFase HACER
       contador = 0;
       MIENTRAS contador ≤ contadorLimite HACER
             seleccionar un movimiento m del vecindario(S);
             sea S' la programación obtenida a partir de S con m;
             SI (C(S')<C(S) o (nbv(S')==0 y C(S')<mejorFactible) o
                              (nbv(S')>0 y C(S')<mejorInfactible))
             ENTONCES
                   aceptar = true;
             SINO
                   aceptar = true con una probabilidad exp(-△C/T),
                   aceptar = false en otro caso;
             FIN SI
             SI aceptar ENTONCES
                   S = S';
                   SI nbv(S) == 0 entonces
                         nbf = min(C(S), mejorFactible);
                   SINO
                         nbi = min(C(S), mejorInfactible);}
                   FIN SI
                   SI nbf<mejorFactible o nbi<mejorInfactible ENTONCES
```

```
                        contador = 0;
                        fase = 0;
                        mejorTemperatura = T;
                        mejorFactible = nbf;
                        mejorInfactible = nbi;
                        SI nbv(S) == 0 ENTONCES
                              w = w/θ;
                        SINO
                              w = w*δ;
                        FIN SI
                  FIN SI
            SINO
                  contador++;
            FIN SI
      FIN MIENTRAS
      fase++;
      T=T*β;
FIN MIENTRAS
```

6.4 Recalentamiento

La idea básica del recalentamiento es que, cuando el recocido simulado alcanza temperaturas muy bajas, tiene dificultades para escapar de mínimos locales, ya que la probabilidad de aceptar movimientos no decrecientes es muy bajo. La idea del recalentamiento es aumentar la temperatura de nuevo para escapar del mínimo local actual.

Se utiliza un método de recalentamiento relativamente simple. La idea es recalentar al finalizar el bucle exterior mediante el aumento de la temperatura al doble de su valor cuando la mejor solución fue encontrada. El algoritmo SA ahora termina cuando el número de recalentamientos consecutivos sin mejorar la mejor solución alcanza un límite determinado.

El algoritmo incluyendo todas estas modificaciones se muestra a continuación:

```
generar programación S aleatoriamente;
mejorFactible = ∞;
nbf = ∞;
mejorInfactible = ∞;
nbi = ∞;
recalentar = 0;
contador = 0;
fase = 0;
MIENTRAS recalentar ≤ maxRecalentar HACER
```

```
        MIENTRAS fase ≤ maxFase HACER
              contador = 0;
              MIENTRAS contador ≤ contadorLimite HACER
                    seleccionar un movimiento m del vecindario(S);
                    sea S' la programación obtenida a partir de S con m;
                    SI (C(S')<C(S) o (nbv(S')==0 y C(S')<mejorFactible)
                    o (nbv(S')>0 y C(S')<mejorInfactible)) ENTONCES
                          aceptar = true;
                    SINO
                          aceptar = true con una probabilidad exp(−ΔC/T),
                          aceptar = false en otro caso;
                    SI aceptar ENTONCES
                          S=S';
                          SI nbv(S) == 0 ENTONCES
                                nbf = min(C(S), mejorFactible);
                          SINO
                                nbi = min(C(S), mejorInfactible);
                          SI (nbf<mejorFactible o nbi<mejorInfactible)
                          ENTONCES
                                contador = 0;
                                fase = 0;
                                mejorTemperatura = T;
                                mejorFactible = nbf;
                                mejorInfactible = nbi;
                                SI nbv(S)==0 ENTONCES
                                      w = w/θ;
                                SINO
                                      w = w*δ;
                                FIN SI
                          SINO
                                contador++;
              FIN MIENTRAS
              fase++;
              T=T*β;
        FIN MIENTRAS
        recalentar++;
        T = 2*mejorTemperatura;
FIN MIENTRAS
```

En la figura 6.1 se muestra un diagrama de flujo con los principales pasos del algoritmo, que permite ver con mayor claridad la ejecución de este.

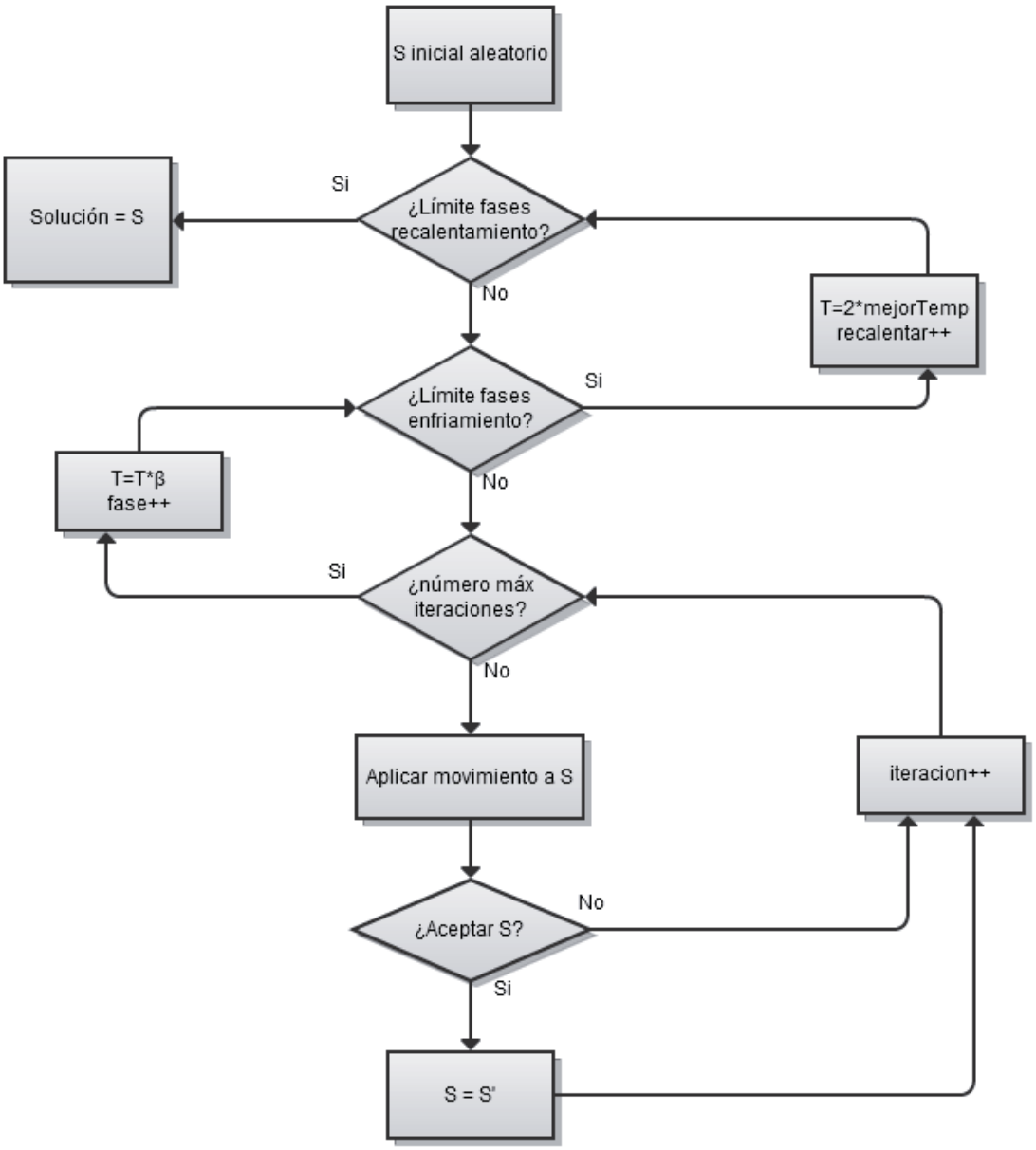

Figura 6.1. Diagrama de flujo del algoritmo Simulated Annealing.

A continuación en la figura 6.2 se presenta un diagrama de flujo más detallado de la parte del criterio de aceptación de S.

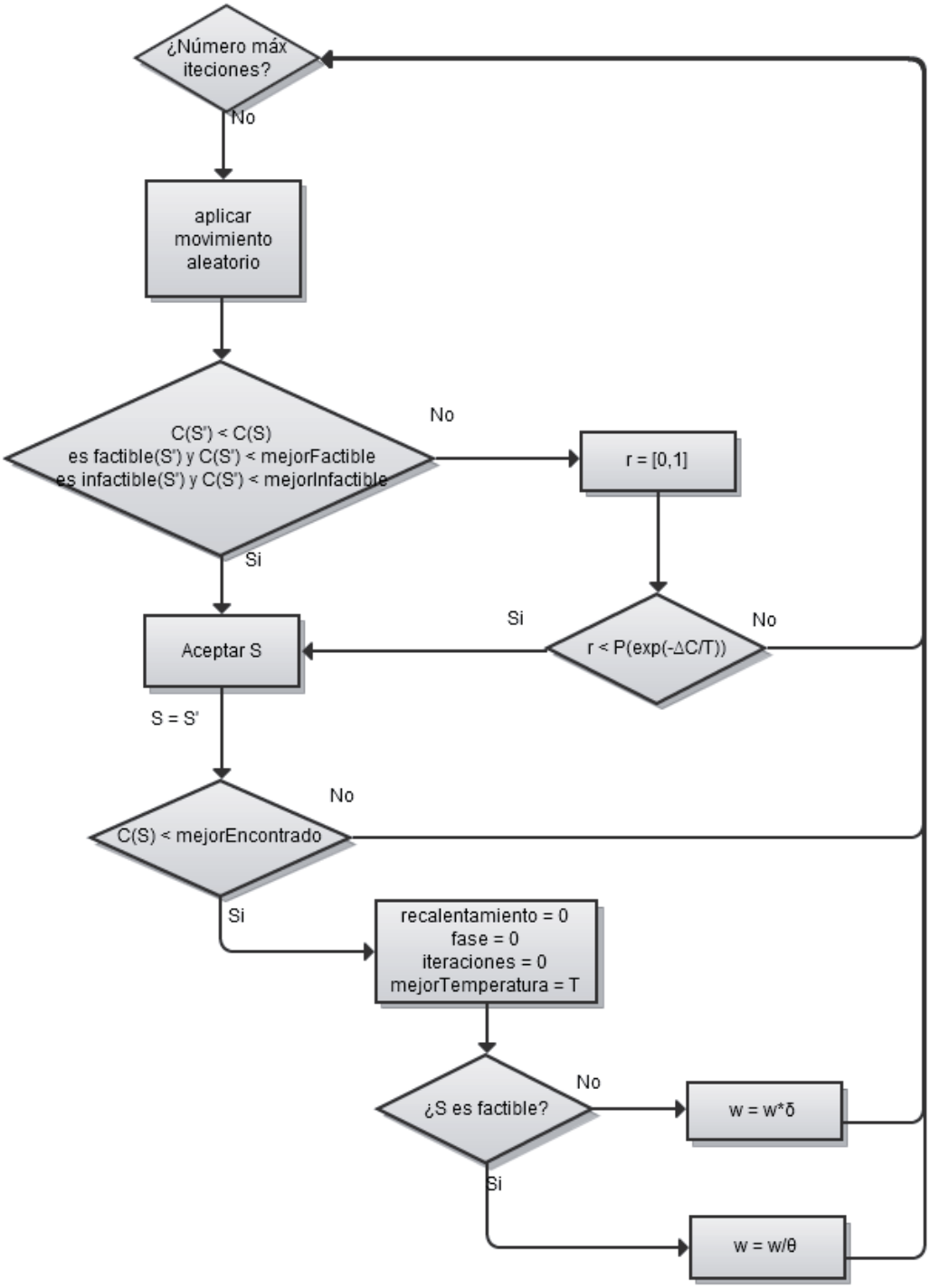

Figura 6.2. Diagrama de flujo del algoritmo Simulated Annealing detallado.

6.5 Generación de la Programación Inicial

El primer paso del funcionamiento del algoritmo SA es el proceso de inicialización de la programación, es decir, generar la programación inicial la cual es aleatoria, y se obtendrá mediante el método del polígono.

6.5.1 Método del Polígono

El método del polígono permite crear programaciones de manera aleatoria. Es un método bastante fácil y eficiente, es decir, las configuraciones generadas no necesitan reparación. La programación que se obtiene es sin la asignación de localías. [3]

Se tienen *n* equipos los cuales son ubicados desde el equipo 1 hasta el equipo *n-1* en los nodos de un polígono regular de *m* nodos: el equipo 1 en el nodo 1, el equipo 2 en el nodo 2, y así sucesivamente, ubicando finalmente el equipo *n* fuera del polígono. Para asignar los enfrentamientos entre los equipos en la primera ronda, se realiza enfrentando al equipo del nodo 1 con el que quedó fuera del polígono, luego se enfrenta el equipo del nodo 2 con el equipo del nodo *m*, el equipo del nodo 3 con el equipo del nodo *m-1* y así sucesivamente, sin repetir asignaciones. Finalizada la asignación de la primera ronda, se gira el polígono en el sentido de las manecillas del reloj realizando la asignación de las siguientes rondas de la misma manera.

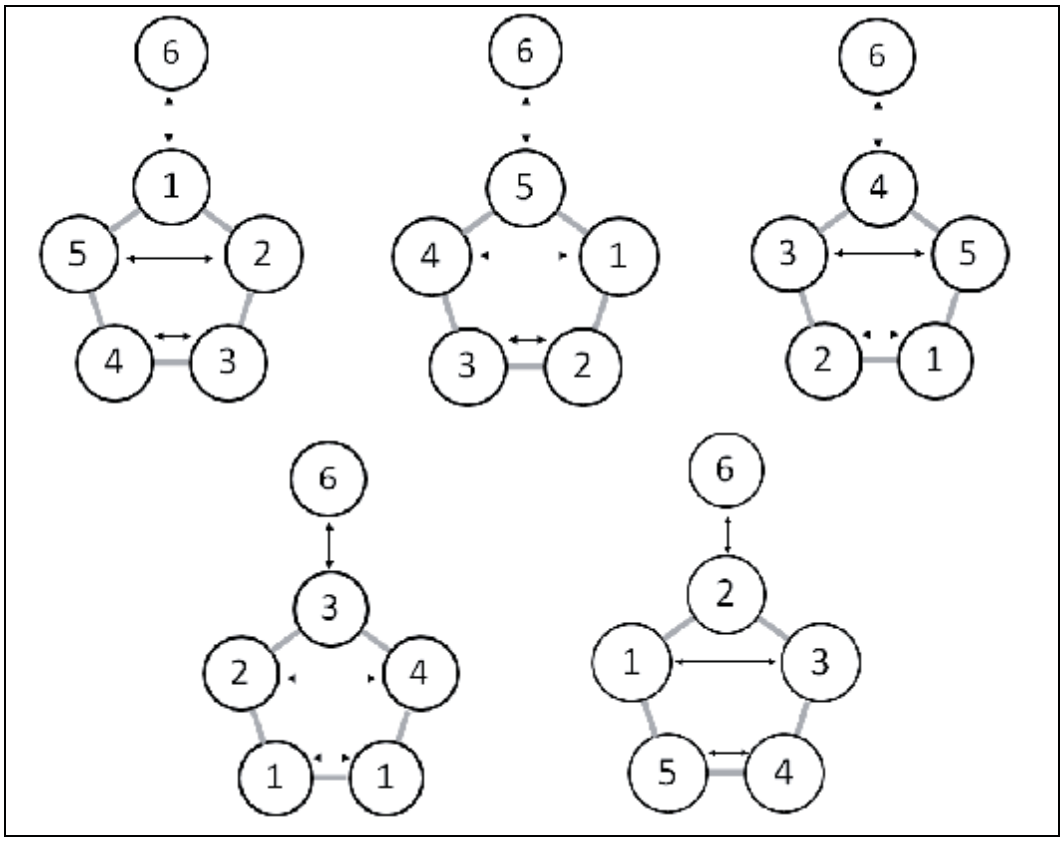

Figura 6.3. Ejemplo de método del polígono para una programación de 6 equipos.

En la figura 6.3 se muestra la construcción de la programación a través del método del polígono, para 6 equipos. Las flechas indican los partidos que se jugarán en cada ronda, por ejemplo, en la ronda 1 se enfrentará el equipo 6 contra el equipo 1, el equipo 2 contra el 5 y el equipo 3 contra el 4. Luego, se hace girar el polígono para generar la siguiente ronda, y de esa manera se van creando todas las rondas siguientes.

En la tabla 6.1 se muestra la programación obtenida con el método del polígono, donde cada columna representa una ronda y las filas muestran el orden de los rivales para cada equipo. Como se ha mencionado anteriormente esta programación se le debe aún aplicar la asignación de localías.

Tabla 6.1. Programación mediante el método del polígono.

T\r	1	2	3	4	5	6	7	8	9	10
1	6	4	2	5	3	6	4	2	5	3
2	5	3	1	4	6	5	3	1	4	6
3	4	2	5	6	1	4	2	5	6	1
4	3	1	6	2	5	3	1	6	2	5
5	2	6	3	1	4	2	6	3	1	4
6	1	5	4	3	2	1	5	4	3	2

6.5.2 Asignación de Localías

Para la asignación de localía de los juegos de la primera ronda estos son designados al azar. Desde la ronda *2* a *n-2*, se utilizará la siguiente estrategia para asignar una localía para el partido entre los equipos T_1 y T_2. Se designará por n_i el número de partidos que el equipo *i* ha jugado de forma consecutiva en las rondas anteriores, ya sea como de local o de visita. [3]

A continuación se describen los diferentes casos:

- Caso 1: $n_{T_2} > n_{T_1}$
  - Si T_2 jugó su último partido de local, se programa el partido en el estadio de T_1.
  - En otro caso, se programa en el estadio de T_2.

- Caso 2: $n_{T_2} < n_{T_1}$
  - Si T_1 jugó su último partido de local, el partido se programará en el estadio de T_2.
  - En otro caso, el partido se programará en el estadio de T_1.

- Caso 3: $n_{T_2} = n_{T_1}$
  - Si T_1 ha jugado su último partido de local y T_2 de visitante, el partido se programará en el estadio de T_2.
  - Si T_2 ha jugado su último partido de local y T_1 de visitante, el partido se programará en el estadio de T_1.
  - En otro caso, se asigna aleatoriamente el estadio del partido.

En la tabla 6.2 se muestra la programación con la asignación de localías aplicado a la programación de la tabla 6.1 utilizando el método anteriormente descrito.

Tabla 6.2. Programación con las asignaciones de localías.

T\r	1	2	3	4	5	6	7	8	9	10
1	6	-4	2	5	-3	-6	4	-2	-5	3
2	-5	3	-1	4	-6	5	-3	1	-4	6
3	4	-2	-5	6	1	-4	2	5	-6	-1
4	-3	1	6	-2	-5	3	-1	-6	2	5
5	2	-6	3	-1	4	-2	6	-3	1	-4
6	-1	5	-4	-3	2	1	-5	4	3	-2

A continuación se detalla el proceso de asignación de localías:

Las localías de la ronda 1 son asignadas aleatoriamente.

En la ronda 2:

- El equipo T_1 se enfrenta con el equipo T_4 ($n_{T_4} = n_{T_1}$). T_1 en su último partido jugó de local y T_4 de visita, por lo tanto, en la ronda 2, T_1 juega de visita y T_4 de local.
- El equipo T_2 se enfrenta con el equipo T_3 ($n_{T_3} = n_{T_2}$). T_2 en su último partido jugó de visita y T_3 de local, por lo tanto, en la ronda 2, T_2 juega de local y T_3 de visita.
- El equipo T_5 se enfrenta con el equipo T_6 ($n_{T_6} = n_{T_5}$). T_5 en su último partido jugó de local y T_6 de visita, por lo tanto, en la ronda 2, T_5 juega de visita y T_6 de local.

En la ronda 3:

- El equipo T_1 se enfrenta con el equipo T_2 ($n_{T_2} = n_{T_1}$). T_1 en su último partido jugó de visita y T_2 de local, por lo tanto, en la ronda 3, T_1 juega de local y T_2 de visita.
- El equipo T_3 se enfrenta con el equipo T_5 ($n_{T_5} = n_{T_3}$). T_3 en su último partido jugó de visita y T_5 de visita, por lo tanto, en la ronda 3, la asignación es aleatoria, T_3 juega de visita y T_5 de local.
- El equipo T_4 se enfrenta con el equipo T_6 ($n_{T_6} = n_{T_4}$). T_4 en su último partido jugó de local y T_6 de local, por lo tanto, en la ronda 3, la asignación es aleatoria, T_4 juega de local y T_6 de visita.

En la ronda 4:

- El equipo T_1 se enfrenta con el equipo T_5 ($n_{T_5} = n_{T_1}$). T_1 en su último partido jugó de local y T_5 de local, por lo tanto, en la ronda 4, la asignación es aleatoria, T_1 juega de local y T_5 de visita.
- El equipo T_2 se enfrenta con el equipo T_4 ($n_{T_4} > n_{T_2}$). T_2 en su último partido jugó de visita y T_4 de local, por lo tanto, en la ronda 4, T_2 juega de local y T_4 de visita.
- El equipo T_3 se enfrenta con el equipo T_6 ($n_{T_6} < n_{T_3}$). T_3 en su último partido jugó de visita y T_6 de visita, por lo tanto, en la ronda 4, T_3 juega de local y T_6 de visita.

Para la ronda 5, se asignan de la misma manera.

6.6 Instancias NL

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

La Tabla 6.3 muestra el cuadro de distancias para la instancia NL8, la cual se encuentra publicada en [13]. Para la representación utilizada a lo largo de este trabajo, los equipos son nombrados numéricamente, es decir, para ATL le corresponde el valor 1, para NYM el 2, para MON el 3 y así sucesivamente para los otros equipos.

Tabla 6.3. Cuadro de distancia para la instancia NL8

	ATL	NYM	PHI	MON	FLA	PIT	CIN	CHI
ATL	0	745	665	929	605	521	370	587
NYM	745	0	80	337	1090	315	567	712
PHI	665	80	0	380	1020	257	501	664
MON	929	337	380	0	1380	408	622	646
FLA	605	1090	1020	1380	0	1010	957	1190
PIT	521	315	257	408	1010	0	253	410
CIN	370	567	501	622	957	253	0	250
CHI	587	712	664	646	1190	410	250	0

6.7 Instancias CIRC

Debido a que el TTP no es fácil de resolver, para lograr estudiarlo de forma mas simple se armó la clase de instancias circulares.

Las instancias CIRC, corresponden a distancias ficticias donde la ubicación de los estadios se sitúan en círculo.

Una instancia circular de n equipos tiene distancias generadas por el grafo circular de n nodos con distancias equivalentes a una unidad. Si se numeran los nodos como $0,1,\ldots,n-1$, se tiene que en este grafo existen arcos de i a $i+1$ ($i=0\ldots n-1$) y de $n-1$ a 0, todos con longitud 1. De esta forma, se tiene que la distancia de i a j ($i > j$) es el mínimo entre $i-j$ y $j-i+n$.

7 Resultados

Luego de haber presentado la definición del problema, las representaciones que se usarán, el algoritmo a utilizar, los componentes de la solución, y el diseño, a continuación se muestran los resultados obtenidos con sus costos asociados y el tiempo transcurrido de ejecución. Se comparan con los mejores resultados obtenidos hasta la fecha para algunas instancias del TTP.

Para la implementación del algoritmo Simulated Annealing se utilizó Scilab que es un lenguaje de programación de alto nivel para cálculos numéricos, de código abierto y disponible para múltiples plataformas.

El algoritmo se ejecutó sólo para las instancias NL4, NL6, NL8, NL10 y NL12, disponibles en [13] y los resultados son contrastados con los mejores conocidos hasta la fecha.

A continuación se muestran las mejores programaciones obtenidas tras varias ejecuciones del algoritmo en su formato original.

7.1 Instancia NL4

En la tabla 7.1 se muestra los parámetros utilizados para cada representación, el tiempo que transcurrió en la ejecución del algoritmo y la solución final obtenida. Para el caso de la instancia NL4 para las dos representaciones se obtuvo un costo de 8276 el cual corresponde al mejor costo encontrado hasta la fecha.

Tabla 7.1. Valores de parámetros, tiempo de ejecución y solución para NL4.

Representación	*w*	θ	δ	β	T	Recalenta-miento	fases	iteración	Tiempo (seg)	solución
Anagnostopoulos	800	1.04	1.04	0.98	400	1	10	300	11.624	8276
Nueva rep.	800	1.04	1.04	0.98	400	1	10	300	15.723	8276

A continuación en la tabla 7.2 se muestra la solución obtenida para 4 equipos con la representación de Anagnostopoulos.

Tabla 7.2. Programación para la instancia NL4.

T\r	1	2	3	4	5	6
1	-4	-2	-3	2	4	3
2	3	1	-4	-1	-3	4
3	-2	-4	1	4	2	-1
4	1	3	2	-3	-1	-2

En la tabla 7.3 se muestra la solución obtenida para 4 equipos con la nueva representación.

Tabla 7.3. Programación para la instancia NL4 con la nueva representación.

	1	2	3	4
1	0	5	6	4
2	2	0	1	6
3	3	4	0	5
4	1	3	2	0

En la figura 7.1 y en la figura 7.2 se puede ver las temperaturas tras la ejecución del algoritmo. Donde la temperatura inicial es de 400. Las fases de recalentamiento consideradas fueron 1, en cada una de estas fases de recalentamiento la temperatura se eleva al doble de la temperatura en la que se encontró una mejor solución. Cada una de estas fases está compuesta por 10 fases de enfriamiento en donde la temperatura se enfría lentamente con un $\beta = 0.98$. Cada fase de enfriamiento está compuesta de 300 iteraciones en donde se calcula nuevas programaciones.

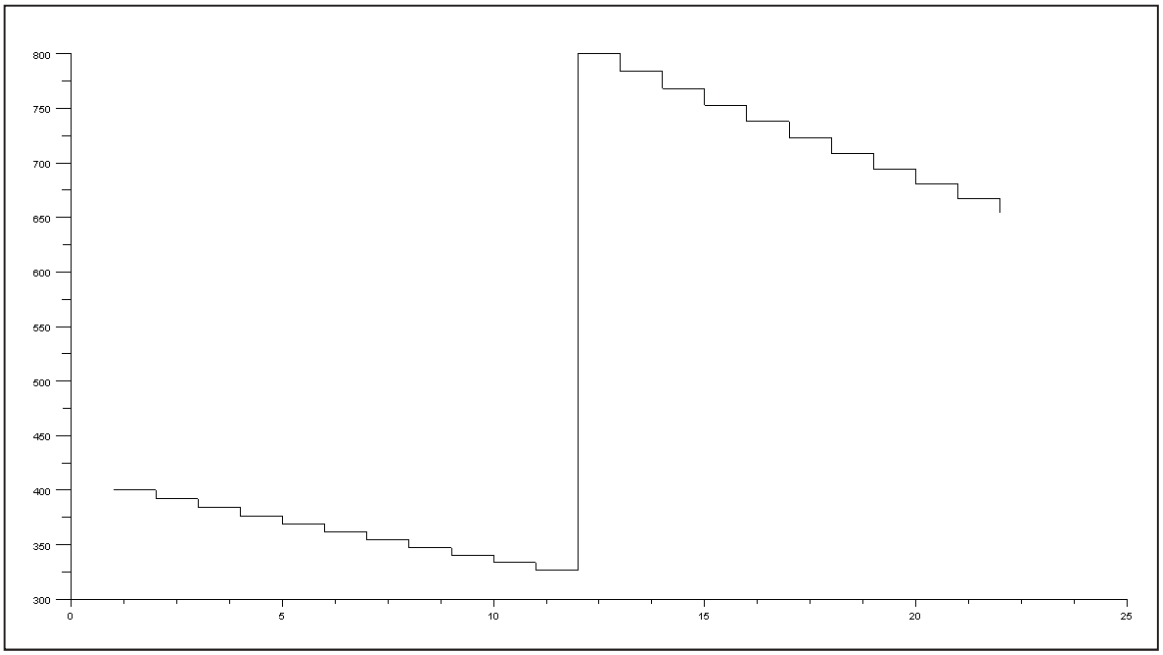

Figura 7.1 Gráfico de temperatura para NL4 representación Anagnostopoulos.

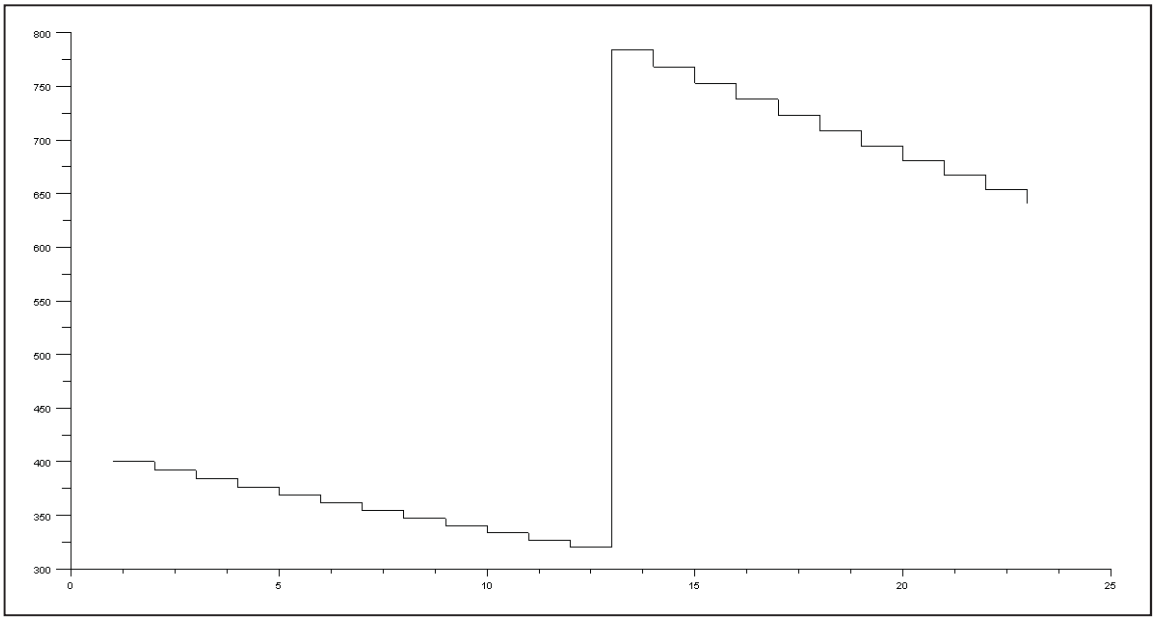

Figura 7.2 Gráfico de temperatura para NL4 representación nueva.

7.2 Instancia NL6

En la tabla 7.4 se muestra los parámetros utilizados para cada representación, el tiempo que transcurrió en la ejecución del algoritmo y la solución final obtenida. Para el caso de la instancia NL6 para la representación Anagnostopoulos se obtuvo un costo de 23916 el cual corresponde al mejor costo encontrado hasta la fecha, mientras que para la nueva representación se obtuvo un costo mayor.

Tabla 7.4. Valores de parámetros, tiempo de ejecución y solución para NL6.

Representación	*w*	θ	δ	β	T	Recalenta-miento	fases	Itera-ción	Tiempo (seg)	solución
Anagnostopoulos	2000	1.04	1.04	0.98	400	5	50	300	1235.481	23916
Nueva rep.	2000	1.04	1.04	0.98	400	5	50	300	539.450	24085

A continuación en la tabla 7.5 se muestra la solución obtenida para 6 equipos con la representación de Anagnostopoulos.

Tabla 7.5. Programación para la instancia NL6.

T\r	1	2	3	4	5	6	7	8	9	10
1	5	2	6	-3	-4	-6	3	4	-2	-5
2	-6	-1	-5	4	5	-3	-4	6	1	3
3	-4	5	4	1	-6	2	-1	-5	6	-2
4	3	-6	-3	-2	1	5	2	-1	-5	6
5	-1	-3	2	6	-2	-4	-6	3	4	1
6	2	4	-1	-5	3	1	5	-2	-3	-4

En la figura 7.3 y en la figura 7.4 se puede ver las temperaturas tras la ejecución del algoritmo. Donde la temperatura inicial es de 400. Las fases de recalentamiento consideradas fueron 5, en cada una de estas fases de recalentamiento la temperatura se eleva al doble de la temperatura en la que se encontró una mejor solución. Cada una de estas fases está compuesta por 50 fases de enfriamiento en donde la temperatura se enfría lentamente con un $\beta = 0.98$. Cada fase de enfriamiento está compuesta de 300 iteraciones en donde se calculan nuevas programaciones.

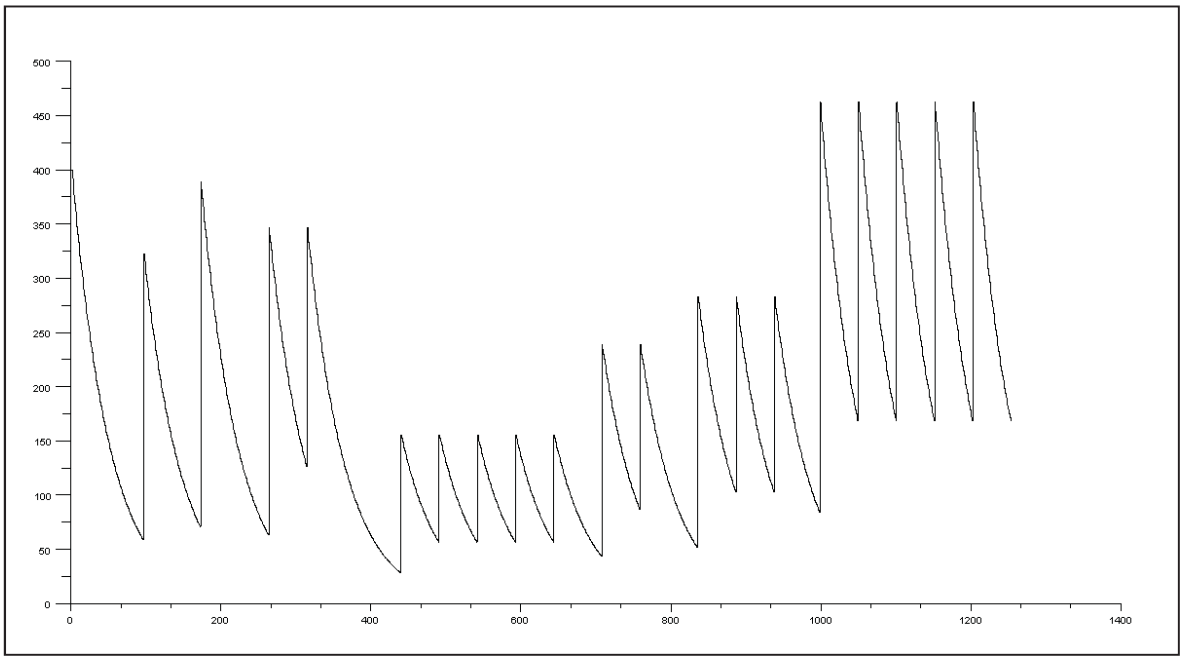

Figura 7.3 Gráfico de temperatura para NL6 representación Anagnostopoulos.

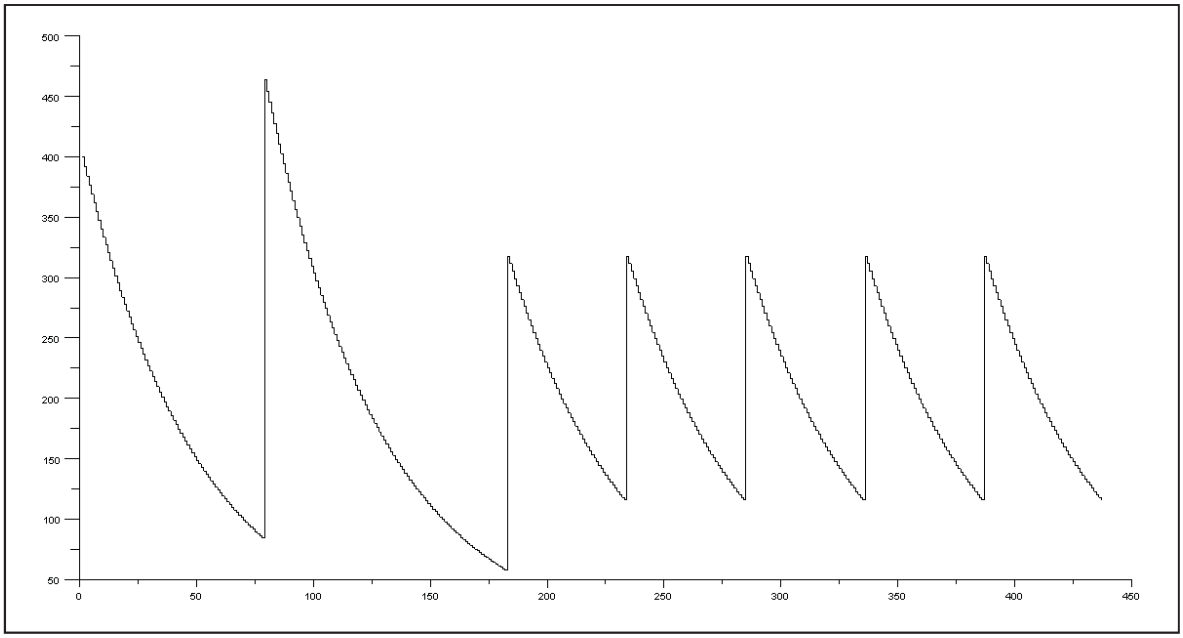

Figura 7.4 Gráfico de temperatura para NL6 representación nueva.

La figura 7.5 muestra el gráfico para la instancia NL6, donde se puede ver cómo evolucionan los resultados para 6 equipos. Inicialmente se tiene un costo de 35640, y se ve a través de la curva como se van encontrando mejores soluciones hasta llegar al óptimo de 23916 para la representación de Anagnostopoulos y de 24085 para la nueva representación.

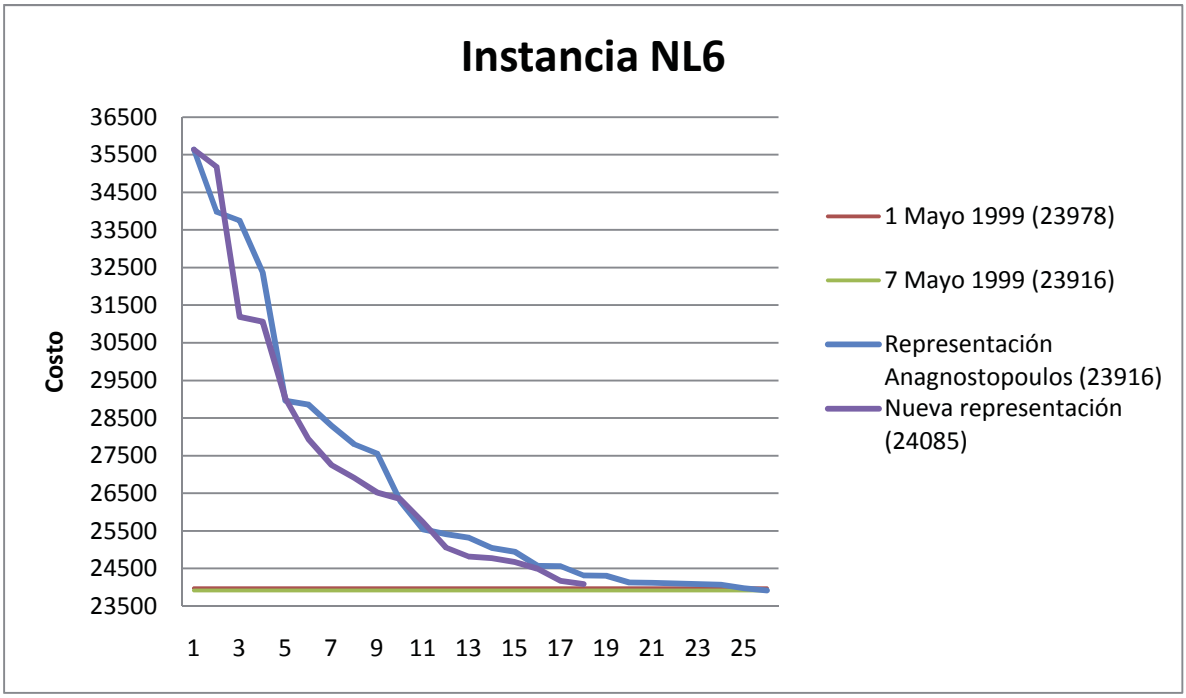

Figura 7.5 Gráfico de costos para NL6.

7.3 Instancia NL8

En la tabla 7.6 se muestra los parámetros utilizados para cada representación, el tiempo que transcurrió en la ejecución del algoritmo y la solución final obtenida. Para el caso de la instancia NL8 para la representación Anagnostopoulos después de ejecutar varias veces el algoritmo se obtuvo un costo de 40416 el cual se sitúa cercano al mejor encontrado hasta la fecha, que es de un costo de 39721, mientras que para la nueva representación se obtuvo un mayor costo de 41650.

Tabla 7.6. Valores de parámetros, tiempo de ejecución y solución para NL8.

Representación	*w*	θ	δ	β	T	Recalenta-miento	fases	Itera-ción	Tiempo (seg)	solución
Anagnostopoulos	4000	1.04	1.04	0.98	600	5	100	300	1092.914	43368
Anagnostopoulos	3000	1.04	1.04	0.98	600	5	100	300	2860.204	40416
Nueva rep.	4000	1.04	1.04	0.98	600	5	100	300	2264.544	41650
Nueva rep.	3000	1.04	1.04	0.98	600	5	100	300	1676.442	43362

A continuación en la tabla 7.7 se muestra la programación para 8 equipos de costo 40416, con la representación de Anagnostopoulos.

Tabla 7.7. Programación para la instancia NL8.

T\r	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	5	6	8	-7	-4	-8	7	2	-6	-3	-2	4	3	-5
2	4	7	-3	-6	3	6	-5	-1	5	8	1	-7	-8	-4
3	7	4	2	-4	-2	5	6	-8	-7	1	8	-5	-1	-6
4	-2	-3	7	3	1	-7	-8	6	8	5	-6	-1	-5	2
5	-1	8	6	-8	-6	-3	2	7	-2	-4	-7	3	4	1
6	8	-1	-5	2	5	-2	-3	-4	1	7	4	-8	-7	3
7	-3	-2	-4	1	8	4	-1	-5	3	-6	5	2	6	-8
8	-6	-5	-1	5	-7	1	4	3	-4	-2	-3	6	2	7

En la figura 7.5 y en la figura 7.6 se puede observar los cambios de temperaturas tras la ejecución del algoritmo. Donde la temperatura inicial es de 600. Las fases de recalentamiento consideradas fueron 5, en cada una de estas fases de recalentamiento la temperatura se eleva al doble de la temperatura en la que se encontró una mejor solución. Cada una de estas fases está compuesta por 100 fases de enfriamiento en donde la temperatura se enfría lentamente con un $\beta = 0.98$. Cada fase de enfriamiento está compuesta de 300 iteraciones en donde se calcula nuevas programaciones.

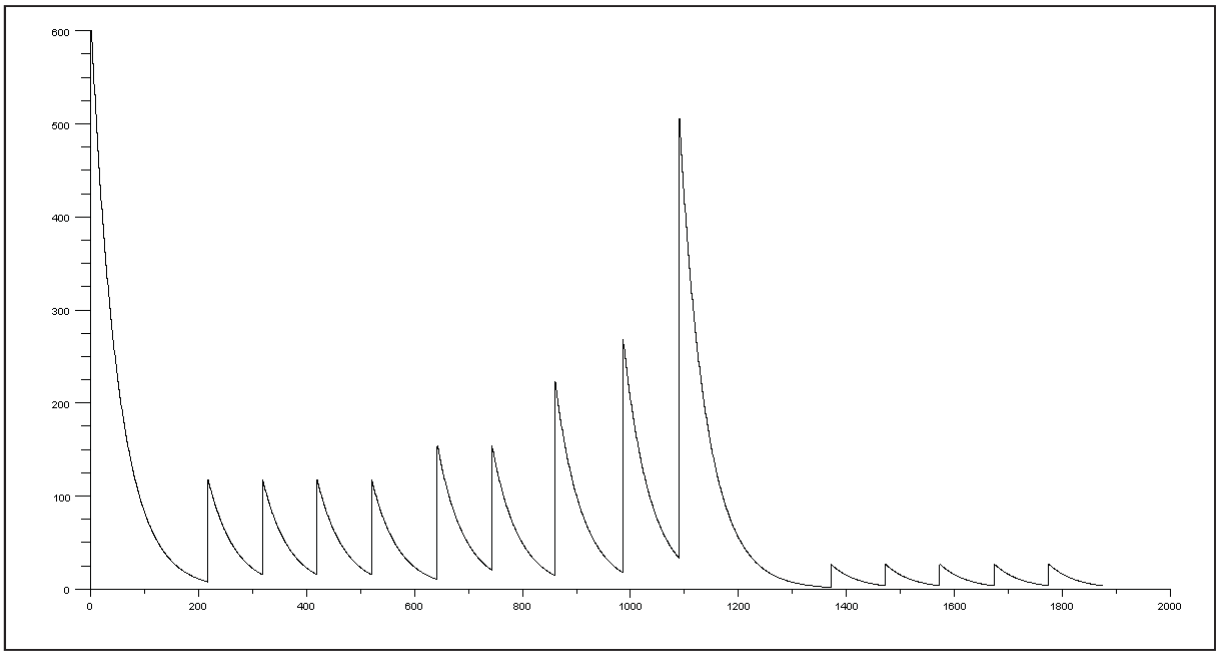

Figura 7.6 Gráfico de temperatura para NL8 representación Anagnostopoulos.

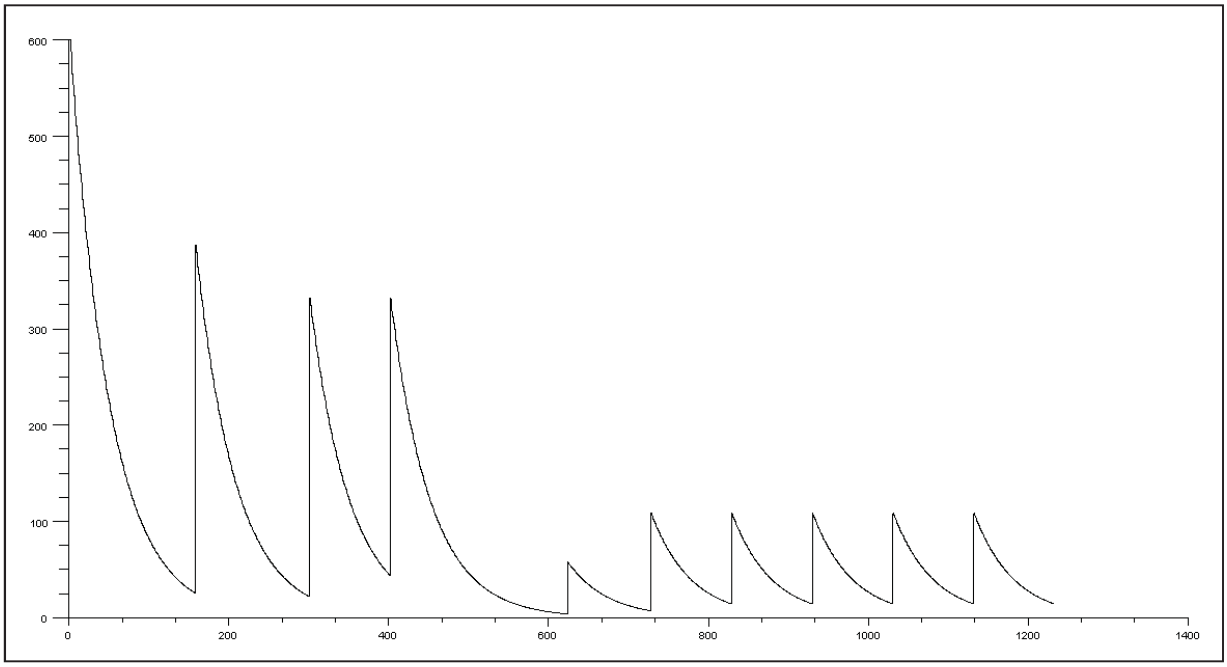

Figura 7.7 Gráfico de temperatura para NL8 representación nueva.

En la figura 7.8 se muestra el gráfico de costos para la instancia NL8, donde se puede ver cómo evolucionan los resultados para 8 equipos. Inicialmente se tiene un costo de 61883, y se ve a través de la curva como se van encontrando mejores soluciones hasta llegar al costo de 41421 para la representación de Anagnostopoulos y de 41650 para la nueva representación.

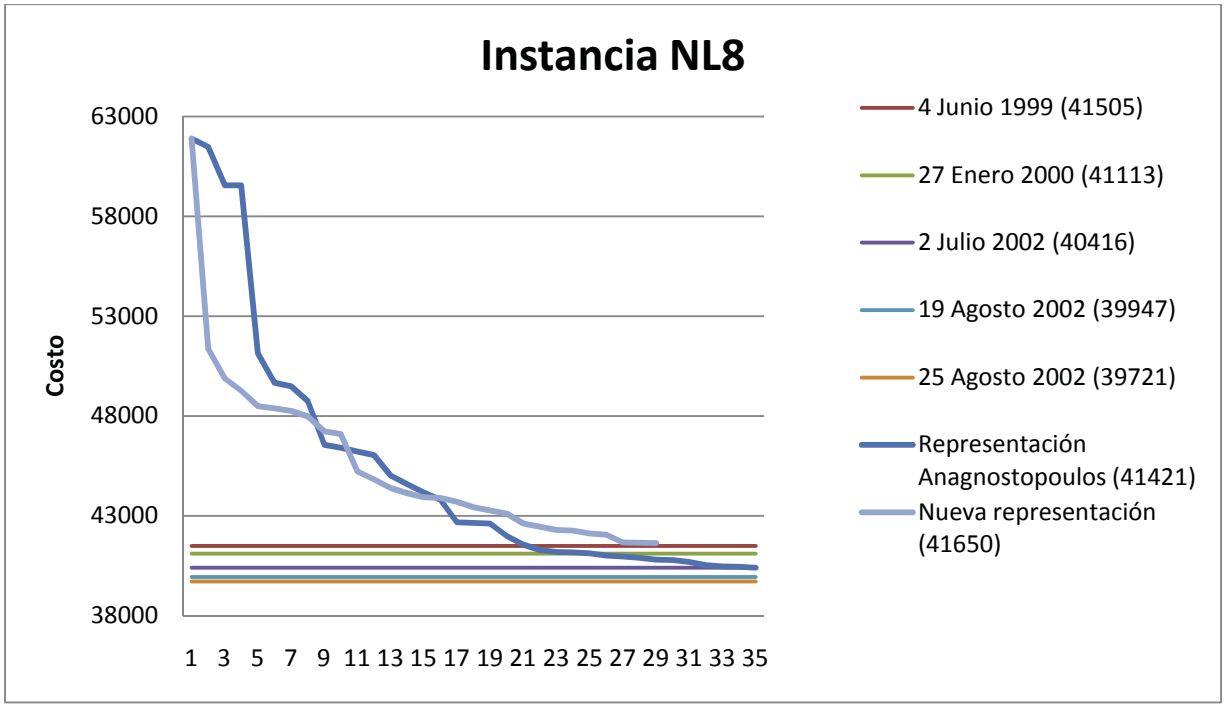

Figura 7.8 Gráfico de costos para NL8.

7.4 Instancia NL10

En la tabla 7.8 se muestra los parámetros utilizados para cada representación, el tiempo que transcurrió en la ejecución del algoritmo y la solución final obtenida. Para el caso de la instancia NL10 para la representación Anagnostopoulos se obtuvo un costo de 66811, mientras que para la nueva representación se obtuvo un costo de 67026 ambos costos se alejan del mejor encontrado hasta la fecha, que es de un costo de 59436.

Tabla 7.8. Valores de parámetros, tiempo de ejecución y solución para NL10.

Representación	*w*	θ	δ	β	T	Recalenta-miento	fases	Itera-ción	Tiempo (seg)	solución
Anagnostopoulos	6000	1.04	1.04	0.98	600	5	100	300	1655.416	66811
Anagnostopoulos	6000	1.04	1.04	0.98	500	6	150	300	11558.20	65984
Nueva rep.	6000	1.04	1.04	0.98	600	5	100	300	6056.380	67026

A continuación en la tabla 7.9 se muestra la programación para 10 equipos de costo 66811, con la representación de Anagnostopoulos.

Tabla 7.9. Programación para la instancia NL10.

T\r	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	-4	-8	-10	6	8	5	-7	-5	2	-6	-3	-2	10	3	-9	7	4	9
2	5	-4	-3	4	10	9	-5	6	-1	-7	-9	1	7	8	-6	-8	-10	3
3	-8	-9	2	10	9	-6	-10	-7	6	-4	1	4	-5	-1	8	5	7	-2
4	1	2	8	-2	-7	10	6	-8	-10	3	-6	-3	9	7	5	-9	-1	-5
5	-2	-6	-7	8	6	-1	2	1	7	-9	-10	-8	3	10	-4	-3	9	4
6	10	5	-9	-1	-5	3	-4	-2	-3	1	4	-7	8	9	2	-10	-8	7
7	-9	-10	5	9	4	-8	1	3	-5	2	8	6	-2	-4	10	-1	-3	-6
8	3	1	-4	-5	-1	7	-9	4	9	10	-7	5	-6	-2	-3	2	6	-10
9	7	3	6	-7	-3	-2	8	10	-8	5	2	-10	-4	-6	1	4	-5	-1
10	-6	7	1	-3	-2	-4	3	-9	4	-8	5	9	-1	-5	-7	6	2	8

En la figura 7.9 se puede ver las temperaturas tras la ejecución del algoritmo. Donde la temperatura inicial es de 500. Las fases de recalentamiento consideradas fueron 6, en cada una de estas fases de recalentamiento la temperatura se eleva al doble de la temperatura en la que se encontró una mejor solución. Cada una de estas fases está compuesta por 150 fases de enfriamiento en donde la temperatura se enfría lentamente con un $\beta = 0.98$. Cada fase de enfriamiento está compuesta de 300 iteraciones en donde se calcula nuevas programaciones.

En la figura 7.10 se ven las temperaturas tras la ejecución del algoritmo. Donde la temperatura inicial es de 600. Las fases de recalentamiento consideradas fueron 5, en cada una de estas fases de recalentamiento la temperatura se eleva al doble de la temperatura en la que se encontró una mejor solución. Cada una de estas fases está compuesta por 100 fases de enfriamiento en donde la temperatura se enfría lentamente con un $\beta = 0.98$. Cada fase de enfriamiento está compuesta de 300 iteraciones en donde se calcula nuevas programaciones.

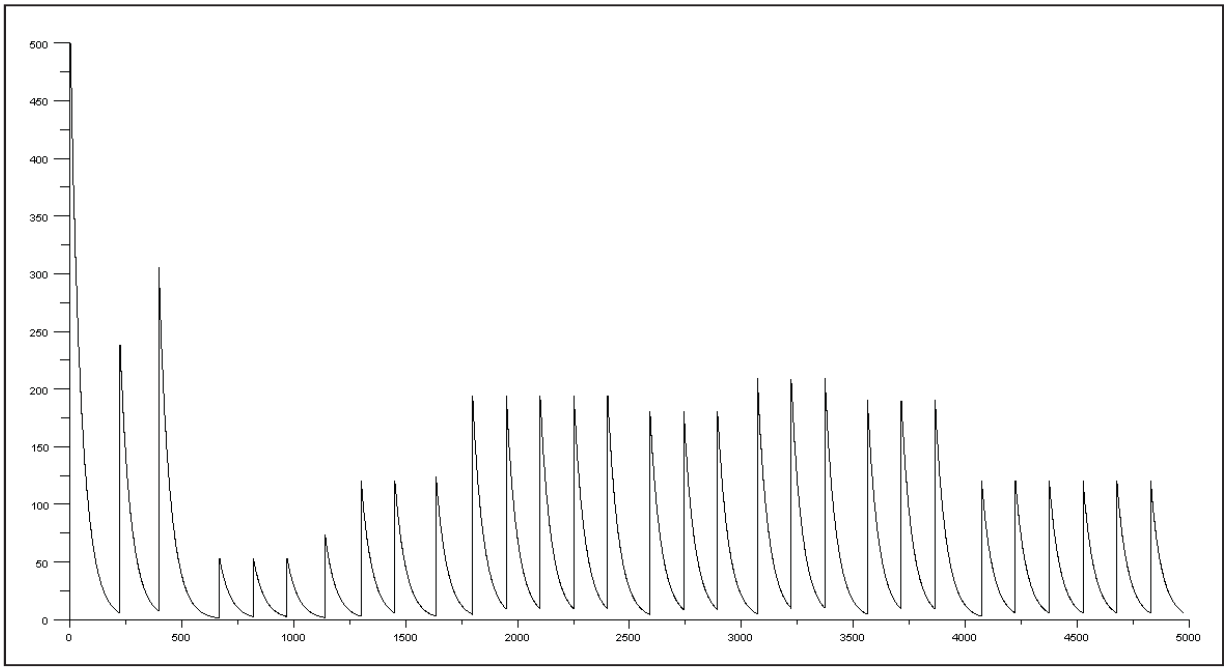

Figura 7.9 Gráfico de temperatura para NL10 representación Anagnostopoulos.

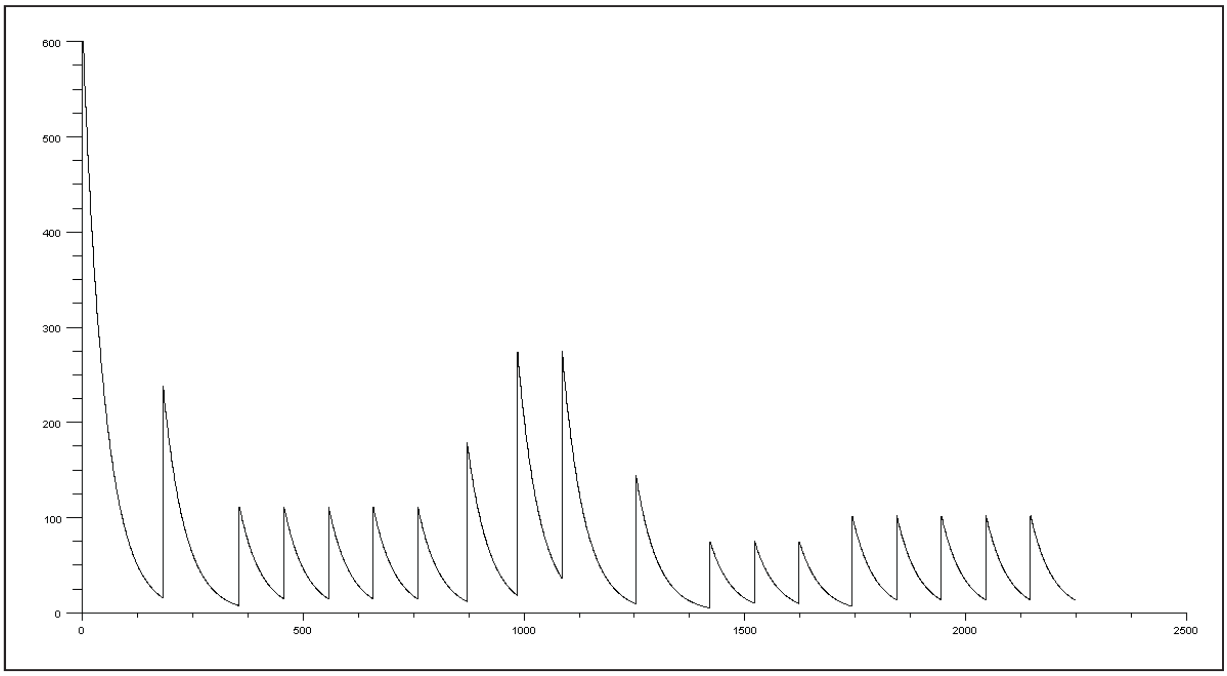

Figura 7.10 Gráfico de temperatura para NL10 representación nueva.

La figura 7.11 se muestra el gráfico de costos para la instancia NL10, donde se puede ver cómo evolucionan los resultados para 10 equipos. Inicialmente se tiene un costo de 98077, y se ve a través de la curva como se van encontrando mejores soluciones hasta llegar al costo de 65984 para la representación de Anagnostopoulos y de 67026 para la nueva representación.

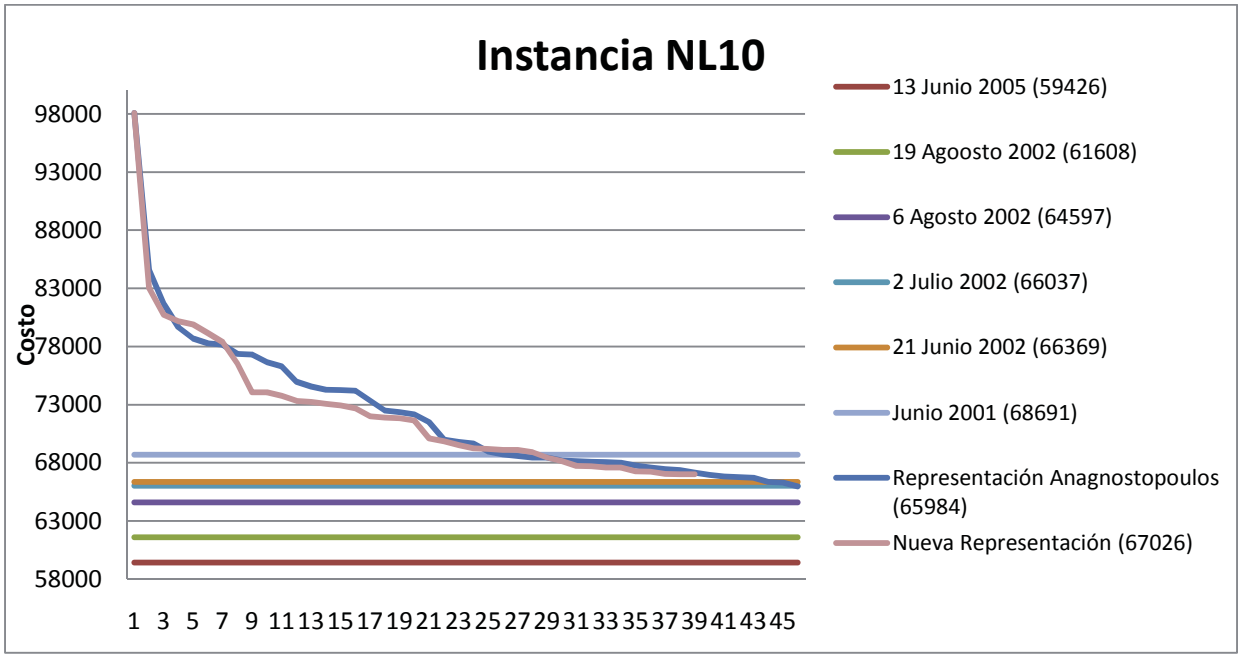

Figura 7.11 Gráfico de costos para NL10.

7.5 Instancia NL12

En la tabla 7.10 se muestra los parámetros utilizados para cada representación, el tiempo que transcurrió en la ejecución del algoritmo y la solución final obtenida. Para el caso de la instancia NL12 para la representación Anagnostopoulos se obtuvo un costo de 130752, mientras que para la nueva representación se obtuvo un costo de 132456 ambos costos se alejan del mejor encontrado hasta la fecha, que es de un costo de 110729.

Tabla 7.10. Valores de parámetros, tiempo de ejecución y solución para NL12.

Representación	*w*	θ	δ	β	T	Recalenta-miento	fases	Itera-ción	Tiempo (seg)	solución
Anagnostopoulos	10000	1.04	1.04	0.98	600	5	100	300	10738.54	127295
Nueva rep.	10000	1.04	1.04	0.98	600	5	100	300	4024.002	132456

En la figura 7.12 y en la figura 7.13 se puede ver las temperaturas tras la ejecución del algoritmo. Donde la temperatura inicial es de 600. Las fases de recalentamiento consideradas fueron 5, en cada una de estas fases de recalentamiento la temperatura se eleva al doble de la temperatura en la que se encontró una mejor solución. Cada una de estas fases está compuesta por 100 fases de enfriamiento en donde la temperatura se enfría lentamente con un $\beta = 0.98$. Cada fase de enfriamiento está compuesta de 300 iteraciones en donde se calcula nuevas programaciones.

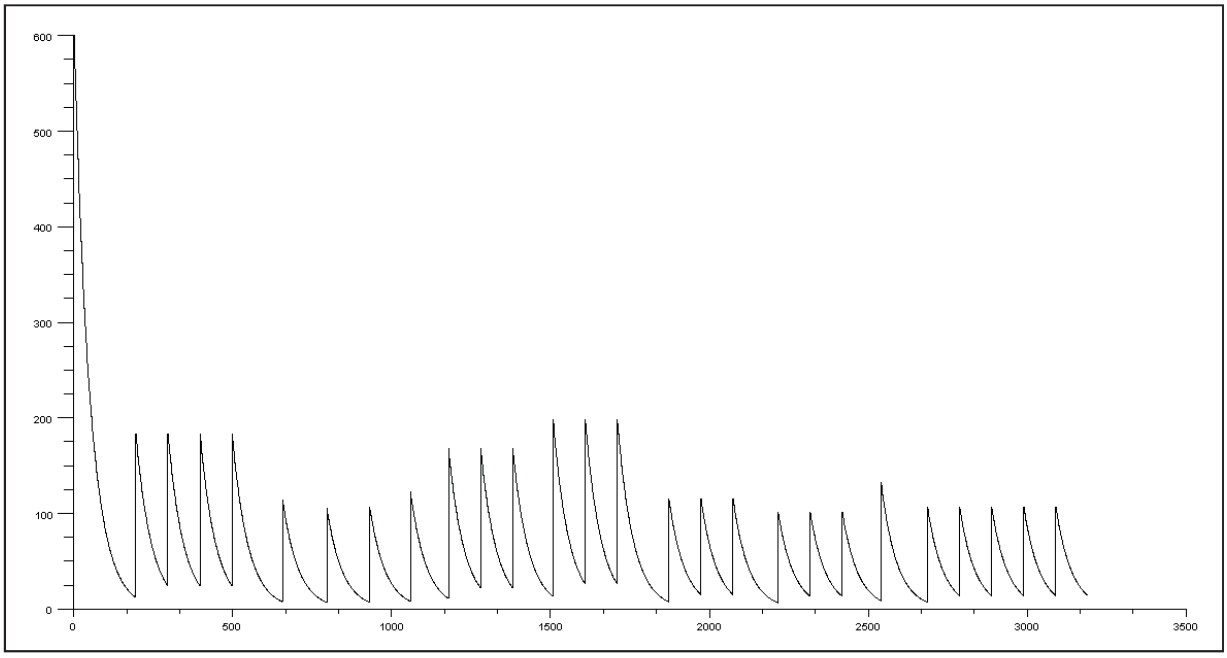

Figura 7.12 Gráfico de temperatura para NL12 representación Anagnostopoulos.

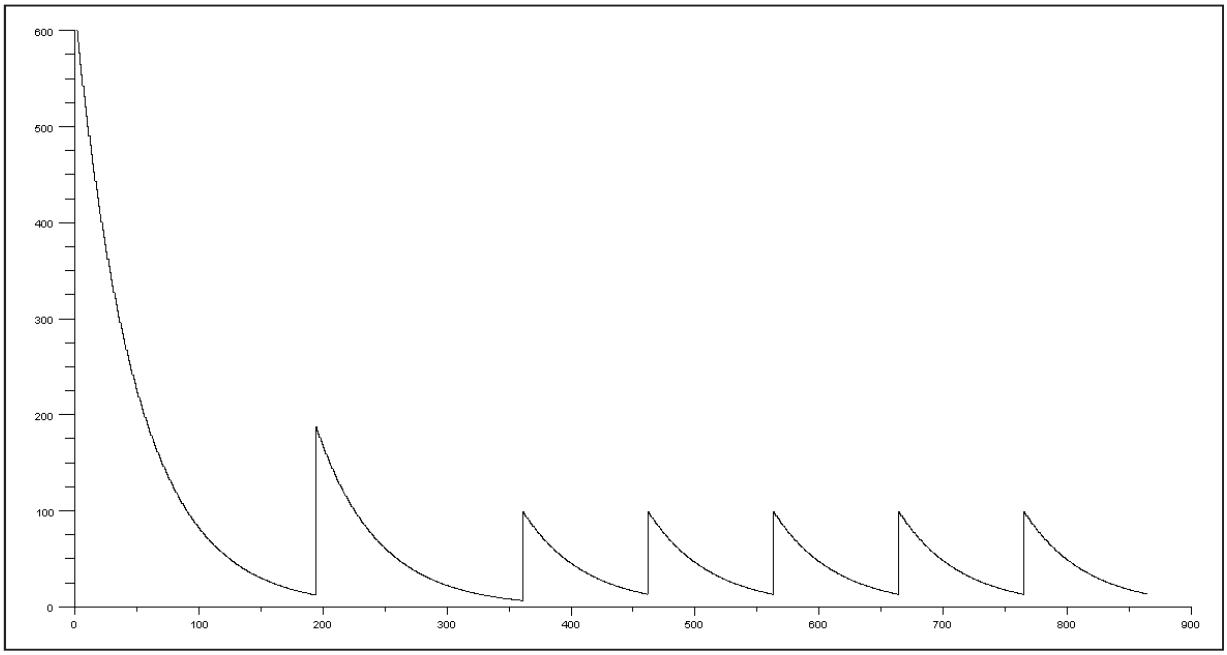

Figura 7.13 Gráfico de temperatura para NL12 representación nueva.

La figura 7.14 se presenta el gráfico de costos para la instancia NL12, donde se puede ver cómo cambian los resultados para 12 equipos. Inicialmente se tiene un costo de 191878, y se ve a través de la curva como se van encontrando mejores soluciones hasta llegar al costo de 127295 para la representación de Anagnostopoulos y de 132456 para la nueva representación.

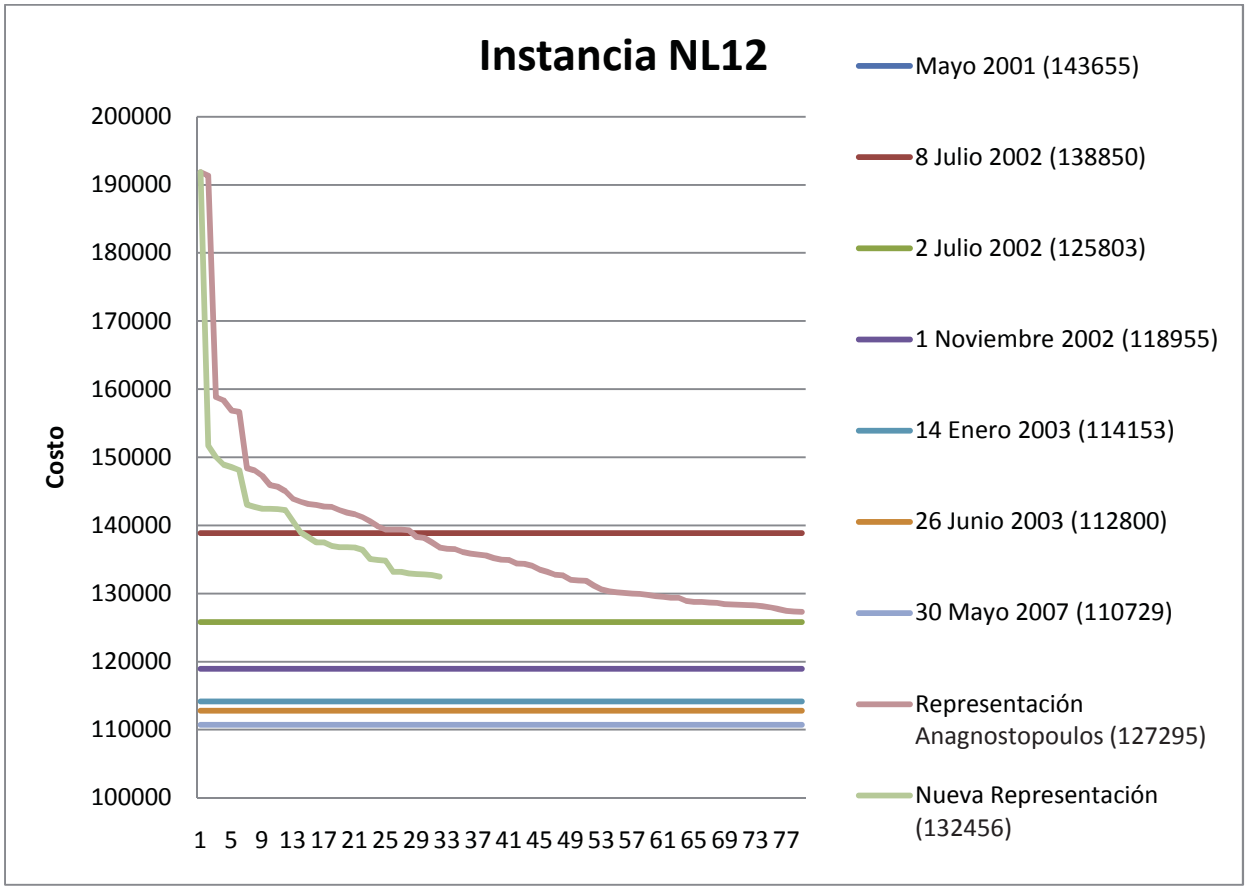

Figura 7.14 Gráfico de costos para NL12.

En la tabla 7.11 se muestra un resumen de los resultados obtenidos para la representación Anagnostopoulos.

Tabla 7.11. Resumen de resultados para la representación Anagnostopoulos.

NLx	w	θ	δ	β	T	Recalenta-miento	fases	iteración	Tiempo (seg)	Solución	Mejor solución
NL4	800	1.04	1.04	0.98	400	1	10	300	11.62	8276	8276
NL6	2000	1.04	1.04	0.98	400	5	50	300	1235.48	23916	23916
NL8	4000	1.04	1.04	0.98	600	5	100	300	1092.91	43368	39721
NL8	3000	1.04	1.04	0.98	600	5	100	300	2860.20	40416	39721
NL10	6000	1.04	1.04	0.98	600	5	100	300	1655.41	66811	59436
NL10	6000	1.04	1.04	0.98	500	6	150	300	11558.20	65984	59436
NL12	10000	1.04	1.04	0.98	600	5	100	300	10738.54	127295	110729

En la tabla 7.12 se presenta un resumen de resultados para la nueva representación.

Tabla 7.12. Resumen de resultados para la nueva representación.

NLx	w	θ	δ	β	T	Recalenta-miento	fases	iteración	Tiempo (seg)	Solución	Mejor solución
NL4	800	1.04	1.04	0.98	400	1	10	300	15.72	8276	8276
NL6	2000	1.04	1.04	0.98	400	5	50	300	539.45	24085	23916
NL8	4000	1.04	1.04	0.98	600	5	100	300	2264.54	41650	39721
NL8	3000	1.04	1.04	0.98	600	5	100	300	1676.44	43362	39721
NL10	6000	1.04	1.04	0.98	600	5	100	300	6056.38	67026	59436
NL12	10000	1.04	1.04	0.98	600	5	100	300	4024.00	132456	110729

7.6 Análisis de Resultados

La ejecución del algoritmo implicó la variación de los distintos parámetros que este contempla. Se variaron las fases de enfriamiento y de recalentamiento así también como la velocidad de enfriamiento y las cantidades de iteraciones en las distintas áreas del algoritmo. Se probó con distintos valores de peso para las penalizaciones de programaciones infactibles obteniendo diferentes resultados con las distintas combinaciones posibles.

Los resultados obtenidos tras la ejecución del algoritmo para la instancia NL4 resultan siempre en la solución óptima, con ambas representaciones. Por otra parte, para la instancia NL6 para la representación Anagnostopoulos se obtiene la mejor solución conocida hasta la fecha, y para la nueva representación se obtuvo una solución bastante cercana al óptimo. En cambio, los resultados obtenidos para las instancias NL8, NL10 y NL12 difieren bastante del óptimo.

La obtención de mejores resultados tras la ejecución del algoritmo va fuertemente ligada a la estrategia de enfriamiento elegida y a la cantidad de fases de recalentamiento escogida, así también como a la correcta elección de los parámetros de penalización.

Es conocido por la literatura [1], que para obtener buenos resultados se debe escoger una estrategia de enfriamiento más lenta (con $\beta = 0{,}999$) y con fases más largas, lo que permite que el algoritmo pueda obtener temperaturas más bajas. También para evitar penalizar demasiado la exploración de la región infactible los parámetros δ y θ se deben escoger cercanos a 1 ($\approx 1{,}04$).

Lamentablemente estas elecciones hacen que los tiempos de ejecución del algoritmo sean muy extensos y requiera de una máquina dedicada para calcular estos resultados, es por este motivo que para la obtención de los resultados presentados sólo se tomó en consideración los valores de los parámetros δ y θ, y se probó con distintos sistemas de enfriamiento y pocas fases de recalentamiento.

Para mejorar los resultados se realizará un estudio con respecto a los movimientos, luego se implementarán para realizar las ejecuciones, y finalmente cuando se tengan los resultados se compararán con las soluciones anteriormente obtenidas. Se espera que estas mejoras logren disminuir los tiempos de ejecución del algoritmo.

8 Estudio para Realizar Mejoras en el Algoritmo

Para mejorar la ejecución del algoritmo se ha propuesto realizar un estudio a los movimientos que son utilizados durante la ejecución, con la finalidad de que sean aplicados de forma más eficiente.

Este estudio consiste en obtener una muestra de datos, ejecutando el algoritmo varias veces y con los datos recogidos realizar un análisis a los movimientos que se van aplicando durante la ejecución del algoritmo.

Se hicieron 18 ejecuciones al algoritmo, con la instancia NL6 utilizando los mismos parámetros para todas. Los datos obtenidos fueron el movimiento aplicado cada vez que se aceptaba, si era factible o no y este se generaba a partir de un factible o infactible.

En la tabla 8.1 se muestra un resumen de los datos obtenidos sobre los movimientos, donde FF son los factibles generados a partir de factibles, FI son los factibles generados de infactibles, IF son los infactibles generados a partir de factibles, e II que son los infactibles generados de infactibles.

Se puede observar que el nuevo movimiento cuando es aplicado a una programación factible siempre genera una programación factible, de igual forma cuando es aplicado a una programación infactible siempre genera una programación infactible. Se puede señalar que el nuevo movimiento es de exploración.

Tabla 8.1. Resumen de datos de los movimientos.

Movimiento	Total	Factibles	Infactibles	FF	FI	IF	II
Swap Home	27682	2782	24900	71,28%	28,72%	2,89%	97,11%
Swap Team	15890	1482	14408	69,44%	30,56%	2,61%	97,39%
Swap Round	4945	318	4627	54,67%	45,33%	3,09%	96,91%
Partial Swap Rounds	8285	660	7625	66,73%	33,27%	3,12%	96,88%
Partial Swap Teams	49305	4122	45183	93,54%	6,46%	0,61%	99,39%
Nuevo	3853	404	3449	100%	0%	0%	100%

En la tabla 8.2 se muestra lo que se desea implementar en el algoritmo para mejorar la eficiencia de la ejecución de este. Se espera que los tiempos de ejecución disminuyan. Cuando se encuentra en una programación factible y se quiere ir a una factible se aplicarán el nuevo movimiento, el partial swap teams, el swap home y el swap team, cuando se encuentra en una infactible y se quiere ir a un factible se aplicarán swap round y partial swap round, si se está en un factible y se quiere ir a un infactible se aplicarán partial swap rounds y swap round y cuando se está en un infactible y se desea ir a un infactible se puede aplicar cualquiera de los 6 movimientos.

Tabla 8.2. Movimientos con relación a factibles e infactibles.

	Factible	Infactible
Factible	Nuevo, PST, SH, ST	PSR, SR
Infactible	SR, PSR	Todos

El algoritmo considera la parte factible como infactible y automáticamente cuando pasa mucho tiempo en la parte factible dirige la búsqueda hacia la parte infactible y de igual forma, cuando ha estado mucho tiempo en la parte infactible dirige las búsqueda hacia la parte factible, entonces se agregó una nueva parte al algoritmo manteniendo lo que ya hacía, agilizando el proceso de dirigir la búsqueda hacia la parte factible o infactible dependiendo de un rango.

A continuación se presentan los resultados asociados a la representación de Anagnostopoulos para las instancias NL6 y NL8.

En la tabla 8.3 se muestran los resultados para la instancia NL6 con la representación de Anagnostopoulos, con la ejecución del algoritmo en su versión original. Se observa que el mejor costo obtenido y que coincide con el mejor encontrado hasta la fecha (costo = 23916) tiene un tiempo de ejecución de 360 segundos y se obtuvo con 1 fase de recalentamiento y con una temperatura de 400.

Tabla 8.3. Solución con rep. Anagnostopoulos NL6, ejecución original.

NL	recalentamientos	temperatura	Tiempo (seg)	costo
NL6	1	400	301	24073
NL6	1	400	241	24085
NL6	1	400	203	24677
NL6	1	400	360	23916
NL6	3	400	468	23916
NL6	3	400	678	24032
NL6	3	400	1016	24254
NL6	3	400	629	24716

En la tabla 8.4 se muestran los resultados para la instancia NL6 con la representación de Anagnostopoulos, con la ejecución del algoritmo con el nuevo movimiento implementado. En este caso se logró obtener un costo de 23916 que es igual al mejor encontrado hasta la fecha, con 3 fases de recalentamiento, con 400 de temperatura y con un tiempo de ejecución de 744 segundos.

Tabla 8.4. Solución con rep. Anagnostopoulos NL6, ejecución con nuevo movimiento

NL	recalentamientos	temperatura	Tiempo (seg)	costo
NL6	1	400	262	24173
NL6	1	400	178	24461
NL6	1	400	267	24064
NL6	1	400	253	25044
NL6	3	400	425	24073
NL6	3	400	705	24467
NL6	3	400	830	24395
NL6	3	400	744	23916

En la tabla 8.5 se muestran los resultados para la instancia NL6 con la representación de Anagnostopoulos, con la ejecución del algoritmo con el nuevo movimiento y con las mejoras

propuestas implementados. Se observa que la mejor solución con el costo de 23916 se consiguió con 1 fase de recalentamiento, con una temperatura de 400 y en un tiempo de ejecución de 237.

Tabla 8.5. Solución con rep. Anagnostopoulos NL6, ejecución con mejoras.

NL	recalentamientos	temperatura	Tiempo (seg)	costo
NL6	1	400	581	24085
NL6	1	400	633	24073
NL6	1	400	237	23916
NL6	1	400	301	23916
NL6	3	400	441	24319
NL6	3	400	552	25145
NL6	3	400	529	24644
NL6	3	400	1180	24108

En la tabla 8.6 se muestran los resultados para la instancia NL8 con la representación de Anagnostopoulos, con la ejecución del algoritmo en su versión original. La mejor solución encontrada tiene un costo de 40707 con 1 fase de recalentamiento, una temperatura de 600 y su tiempo de ejecución fue de 1082 segundos.

Tabla 8.6. Solución con rep. Anagnostopoulos NL8, ejecución original.

NL	recalentamientos	temperatura	Tiempo (seg)	costo
NL8	1	600	1082	40707
NL8	1	600	434	41283
NL8	3	600	1533	41695
NL8	3	600	679	44473
NL8	3	600	1794	41630
NL8	5	600	1849	42367
NL8	5	600	1137	40852

En la tabla 8.7 se muestran los resultados para la instancia NL8 con la representación de Anagnostopoulos, con la ejecución del algoritmo con el nuevo movimiento implementado. El menor costo encontrado fue de 41254 con 3 fases de recalentamiento y un tiempo de ejecución de 2965 segundos.

Tabla 8.7. Solución con rep. Anagnostopoulos NL8, ejecución con nuevo movimiento

NL	recalentamientos	temperatura	Tiempo (seg)	costo
NL8	1	600	629	42497
NL8	1	600	654	41893
NL8	3	600	736	44257
NL8	3	600	2965	41254
NL8	5	600	1370	41843
NL8	5	600	998	42309

En la tabla 8.8 se muestran los resultados para la instancia NL8 con la representación de Anagnostopoulos, con la ejecución del algoritmo con el nuevo movimiento incluido y con las

mejoras propuestas implementadas. El costo menor que se obtuvo fue de 41478 con 5 fases de recalentamiento y en un tiempo de 2332 segundos.

Tabla 8.8. Solución con rep. Anagnostopoulos NL8, ejecución con mejoras.

NL	recalentamientos	temperatura	Tiempo (seg)	costo
NL8	1	600	470	43521
NL8	1	600	701	41626
NL8	3	600	847	41657
NL8	3	600	767	41633
NL8	5	600	2146	41967
NL8	5	600	2332	41478

A continuación se exponen los resultados asociados a la nueva representación para instancias NL6 y NL8.

En la tabla 8.9 se muestran los resultados para la instancia NL6 con la nueva representación, con la ejecución del algoritmo en su versión original. Se observa que el mejor costo obtenido y que coincide con el mejor encontrado hasta la fecha (costo = 23916) tiene un tiempo de ejecución de 328 segundos y se obtuvo con 1 fase de recalentamiento y con una temperatura de 400.

Tabla 8.9. Solución con nueva representación NL6, ejecución original.

NL	recalentamientos	temperatura	Tiempo (seg)	costo
NL6	1	400	422	24306
NL6	1	400	250	24085
NL6	1	400	328	23916
NL6	1	400	472	24108
NL6	3	400	623	24553
NL6	3	400	650	24467
NL6	3	400	1244	23916
NL6	3	400	536	24659

En la tabla 8.10 se muestran los resultados para la instancia NL6 con la nueva representación, con la ejecución del algoritmo con el nuevo movimiento implementado. En este caso se logró obtener el costo de 23916 que corresponde al mejor encontrado hasta la fecha, con 1 fase de recalentamiento con 400 de temperatura y con un tiempo de ejecución de 712 segundos.

Tabla 8.10. Solución con nueva representación NL6, ejecución con nuevo movimiento

NL	recalentamientos	temperatura	Tiempo (seg)	costo
NL6	1	400	226	24900
NL6	1	400	586	24698
NL6	1	400	712	23916
NL6	3	400	1176	24073
NL6	3	400	1097	24662

En la tabla 8.11 se muestran los resultados para la instancia NL6 con la nueva representación, con la ejecución del algoritmo con el nuevo movimiento y con las mejoras propuestas implementados. Se observa que la mejor solución con el costo de 23916 se consiguió con 3 fases de recalentamientos y con una temperatura de 400 en un tiempo de ejecución de 436.

Tabla 8.11. Solución con nueva representación NL6, ejecución con mejoras.

NL	recalentamientos	temperatura	Tiempo (seg)	costo
NL6	1	400	574	24480
NL6	1	400	954	24710
NL6	3	400	436	23916
NL6	3	400	586	24073

En la tabla 8.12 se muestran los resultados para la instancia NL8 con la nueva representación, con la ejecución del algoritmo en su versión original. El mejor costo encontrado fue de 40792 con 1 fase de recalentamiento, 600 de temperatura y el tiempo de ejecución fue de 680 segundos.

Tabla 8.12. Solución con nueva representación NL8, ejecución original.

NL	recalentamientos	temperatura	Tiempo (seg)	costo
NL8	1	600	680	40792
NL8	1	600	609	43081
NL8	3	600	1491	41130
NL8	3	600	1654	41612
NL8	5	600	1219	41740
NL8	5	600	1807	42028

En la tabla 8.13 se muestran los resultados para la instancia NL8 con la nueva representación, con la ejecución del algoritmo con el nuevo movimiento implementado. El mejor costo encontrado fue de 40723 con 3 fases de recalentamiento, 600 de temperatura y el tiempo de ejecución fue de 1565 segundos.

Tabla 8.13. Solución con nueva representación NL8, ejecución con nuevo movimiento

NL	recalentamientos	temperatura	Tiempo (seg)	costo
NL8	1	600	477	42804
NL8	1	600	650	42235
NL8	3	600	1565	40723
NL8	3	600	1145	41328
NL8	5	600	1761	41077
NL8	5	600	1999	41237

En la tabla 8.14 se muestran los resultados para la instancia NL8 con la nueva representación, con la ejecución del algoritmo con el nuevo movimiento incluido y con las mejoras propuestas implementadas. El costo menor que se obtuvo fue de 40800 con 3 fases de recalentamiento y en un tiempo de 2764 segundos.

Tabla 8.14. Solución con nueva representación NL8, ejecución con mejoras.

NL	recalentamientos	temperatura	Tiempo (seg)	costo
NL8	1	600	468	42331
NL8	1	600	742	42928
NL8	3	600	1024	43827
NL8	3	600	2764	40800
NL8	5	600	1606	41604
NL8	5	600	1400	41537

A continuación, en la tabla 8.15, se presenta un resumen de resultados para la instancia NL6 para las dos representaciones. Se muestran las mejores soluciones encontradas con ambas representaciones y para los diversos formatos de ejecución se logró obtener el costo del mejor hasta la fecha, costo de 23916. El que menos se demoró en encontrar la solución fue para la representación de Anagnostopoulos con las mejoras implementadas en un tiempo de 237 segundos.

Tabla 8.15. Resumen de soluciones para NL6

Representación		recalentamientos	temperatura	Tiempo (seg)	costo
Anagnostopoulos	SA original	1	400	360	23916
Anagnostopoulos	SA + movimiento nuevo	3	400	744	23916
Anagnostopoulos	SA + movimiento nuevo + mejoras	1	400	237	23916
Nueva Representación	SA original	1	400	328	23916
Nueva Representación	SA + movimiento nuevo	1	400	712	23916
Nueva Representación	SA + movimiento nuevo + mejoras	3	400	436	23916

A continuación, en la tabla 8.16, se presenta un resumen de resultados para la instancia NL8 para las dos representaciones. La mejor solución encontrada es para la ejecución del algoritmo en su forma original con la representación de Anagnostopoulos, se obtuvo el costo de 40707 en 1 fase de recalentamiento y su tiempo de ejecución fue de 1082 segundos. Luego le sigue el costo de 40723 con la representación nueva con el nuevo movimiento.

Tabla 8.16. Resumen de soluciones para NL8

Representación		recalentamientos	temperatura	Tiempo (seg)	costo
Anagnostopoulos	SA original	1	600	1082	40707
Anagnostopoulos	SA + movimiento nuevo	3	600	2965	41254
Anagnostopoulos	SA + movimiento nuevo + mejoras	5	600	2332	41478
Nueva Representación	SA original	1	600	680	40792
Nueva Representación	SA + movimiento nuevo	3	600	1565	40723
Nueva Representación	SA + movimiento nuevo + mejoras	3	600	2764	40800

A continuación se presentan resultados para las instancias circulares.

Los resultados para la representación Anagnostopoulos se muestran en la tabla 8.17.

Tabla 8.17. Resultados instancia Circular para la representación Anagnostopoulos.

Instancia		Costo	Tiempo (seg)	temperatura	w	fases	iteración	óptimo
Circ4	Original	20	72	400	300	50	300	20
Circ4	Nuevo mov	20	257	400	300	50	300	
Circ4	mejoras	20	84	400	300	50	300	
Circ6	Original	64	4969	400	350	100	500	64
Circ6	Nuevo mov	64	14181	400	350	100	500	
Circ6	mejoras	64	2197	400	350	100	500	
Circ8	Original	146	7329	600	400	100	300	132
Circ8	Nuevo mov	150	12532	600	400	100	300	
Circ8	mejoras	148	3524	600	400	100	300	
Circ10	Original	298	4281	600	250	100	300	242
Circ10	Nuevo mov	298	13517	600	250	100	300	
Circ10	mejoras	298	4910	600	250	100	300	
Circ12	Original	496	8786	600	400	50	300	404
Circ12	Nuevo mov	512	8735	600	400	50	300	
Circ12	mejoras	496	10201	600	400	50	300	

Los resultados para la nueva representación se muestran a continuación en la tabla 8.18.

Tabla 8.18. Resultados instancia Circular para la nueva representación.

Instancia		costo	Tiempo (seg)	temperatura	w	fases	iteración	óptimo
Circ4	Original	20	92	400	300	50	300	20
Circ4	Nuevo mov	20	35	400	300	50	300	
Circ4	mejoras	20	98	400	300	50	300	
Circ6	Original	64	17250	400	350	100	500	64
Circ6	Nuevo mov	64	13264	400	350	100	500	
Circ6	mejoras	64	1487	400	350	100	500	
Circ8	Original	146	21490	600	400	100	300	132
Circ8	Nuevo mov	146	15136	600	400	100	300	
Circ8	mejoras	148	3361	600	400	100	300	
Circ10	Original	280	14197	600	250	100	300	242
Circ10	Nuevo mov	298	16200	600	250	100	300	
Circ10	mejoras	282	6464	600	250	100	300	
Circ12	Original	520	5486	600	400	50	300	404
Circ12	Nuevo mov	514	8354	600	400	50	300	
Circ12	mejoras	508	12136	600	400	50	300	

9 Conclusiones

Uno de los problemas más estudiados en el área de la optimización ha sido el Traveling Tournament Problem el cual ha atraído la atención de un sin número de investigadores alrededor del mundo debido a su dificultad de obtener resultados óptimos para la programación de torneos deportivos de más de 8 equipos, generando un gran reto a resolver. Por lo que en los últimos años se ha puesto especial interés en la utilización de técnicas usando algoritmos y heurísticas para tratar de resolver este problema.

El uso de tecnologías modernas pueden ser efectivas también en el campo del deporte para hacer campeonatos más atractivos para el público, así como también, más rentables y justos para los clubes. Es por eso que el trabajo expuesto en este proyecto entrega conocimientos sobre el algoritmo Simulated Annealing con el fin de ayudar en la programación de los torneos deportivos.

A pesar de que Simulated Annealing es un algoritmo que nació basado en la termodinámica, ha sido utilizado para diversos problemas de optimización, es por eso que se ha planteado el uso de él para resolver el TTP.

Se implementó el algoritmo en el lenguaje Scilab, permitiendo obtener resultados para distintas instancias del TTP. Se obtuvo soluciones cercanas a las óptimas lo que permite concluir que el algoritmo Simulated Annealing es una buena alternativa para resolver el problema del TTP.

El algoritmo SA permite explorar tanto las regiones factibles como las infactibles utilizando un vecindario para poder moverse en el espacio de soluciones, además incluye una técnica de oscilación estratégica y de recalentamiento que permite balancear la exploración de las regiones factibles e infactibles y escapar de los mínimos locales a bajas temperaturas.

El uso de este algoritmo debe ser tratado con cuidado con respecto a las decisiones de los parámetros y variables puesto que influyen considerablemente en la calidad de las soluciones, esto se puede ver claramente en los resultados expuestos, comparados con los propuestos en la literatura.

Los valores de los parámetros deben ser escogidos según la cantidad de equipos que se involucren, con instancias pequeñas como NL4 y NL6 la temperatura inicial utilizada fue de 400 con pocas fases de recalentamiento y de enfriamiento, sin embargo, para instancias mayores se deben ir aumentando el valor de la temperatura inicial como también las fases.

La nueva representación reduce el espacio de búsqueda y después de realizar el análisis de los movimientos más comunes con la representación de Anagnostopoulos y la representación que se propuso se concluye que con la nueva representación se requiere ejecutar la mitad de las operaciones para construir una nueva solución.

Con la nueva representación, se esperaba que se encontrasen mejores soluciones, sin embargo, los resultados no fueron mejores que con la otra representación. Por otra parte, la nueva representación permite realizar cálculos con mayor rapidez ya que la cantidad de operaciones para realizar los movimientos es menor.

Si bien no se obtuvo mejores soluciones, se esperaba que la nueva representación se demorase menor tiempo en entregar una solución, no fue en todos los casos menor tiempo, puesto que, al ser las iteraciones y las fases variables estas se reinicializan cada vez que se encuentra una mejor solución, de haber sido las fases e iteraciones fijas, el tiempo transcurrido desde que se inicia hasta que finaliza hubiera sido menor.

Se realizó la implementación de un nuevo movimiento en el vecindario, el cual es de exploración y que cuando se está en una programación factible siempre genera una solución factible mientras que cuando se está en una solución infactible genera soluciones infactibles. Los resultados obtenidos no mostraron gran diferencia con los encontrados anteriormente, es decir, incorporar este nuevo movimiento no genera mejores soluciones ni tampoco empeora los resultados.

Además, se realizó un estudio a los movimientos que se van generando y aceptando durante la ejecución del algoritmo, con estos datos se agregó al algoritmo una mejora la cual consiste en agilizar el proceso de dirigir la búsqueda hacia la parte factible o infactible dependiendo de un rango.

Referencias

[1] Anagnostopoulos, A., Michel, L., Van Hentenryck, P. and Vergados, Y. "A Simulated Annealing Aproach to the Traveling Tournament Problem", Proceedings CPAIOR'03, 2003.

[2] Cardemil A. "Optimización de fixtures deportivos: estado del arte y un algoritmo Tabu search para el Traveling Tournament Problem", tesis, 2002.

[3] Celso C. Ribeiro, Sebastian Urrutia, "*Heuristics for the Mirrored Traveling Tournament Problem"*, PATAT 2004 - Practice and Theory of Automated Timetabling, 323-342, 2004.

[4] Cerny, V. "A thermodynamic approach to the traveling salesman problem: An efficient simulation", J. Optim. Theory Appl. 45 41-51. 1985.

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

[6] Dowsland, K. & Díaz, B. "Heuristic design and fundamentals of Simulated Annealing", Revista Iberoamericana de Inteligencia Artificial nº 19, pp 93-102, 2003.

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

[8] Kirkpatrick, Gelett y Vecci. "Optimization by simulated annealing" Science 220 621-630, 1983.

[9] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A.H., Teller, E.: Equation of state calculation by fast computing machines. Journal of Chemistry Physics, 21: 1087-1091, 1953.

[10] Morales, E., "Búsqueda, Optimización y Aprendizaje" Capítulo 4, 2004.

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

[12] Schaerf, A. & Di Gaspero, L. "A composite-neighborhood tabu search approach to the Traveling Tournament Problem", 2006.

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

Anexos

A Código Fuente

A.1 Funciones de los Movimientos para la Representación Anagnostopoulos

A.1.1 Función SwapHomes

```
function S2 = SwapHomes(S,Ti,Tj)
   //Buscar el equipo Tj de local en la fila Ti
   pos1 = find(S(Ti,:) ==  Tj);
   //Buscar el equipo Tj de visita en la fila Ti
   pos2 = find(S(Ti,:) == -Tj);
   //Intercambiar visita local de los equipos Ti y Tj
   S([Ti Tj],[pos1 pos2]) = S([Ti Tj],[pos2 pos1]);
   S2  = S;
endfunction
```

A.1.2 Función SwapRounds

```
function S2 = SwapRounds(S,Rk,Rl)
   //Intercambiar los rounds Rk y Rl (columnas)
   S(:,[Rk Rl]) = S(:,[Rl Rk]);
   S2 = S;
endfunction
```

A.1.3 Función SwapTeams

```
function S2 = SwapTeams(S,Ti,Tj)
   //Buscar las 2 posiciones que no variarán
   //Buscar el equipo Tj de local en la fila Ti
   pos1 = find(S(Ti,:) ==  Tj);
   //Buscar el equipo Tj de visita en la fila Ti
   pos2 = find(S(Ti,:) == -Tj);
   //intercambiar las 2 filas completas
   S([Ti  Tj],:) = S([Tj Ti],:);
   S([Ti Tj],[pos1 pos2]) = S([Tj Ti],[pos1 pos2]);
   //Intercambiar cuando se enfrentan
   ncolumnas = size(S,"c");
   for col = 1 : ncolumnas
```

```
      if( col <> pos1 & col <> pos2 ) then
         fila1 = abs(S(Ti,col));
         fila2 = abs(S(Tj,col));
         S = SwapAux(S,fila1,fila2,col);
      end
   end
   S2 = S;
endfunction
```

A.1.4 Función PartialSwapRounds

```
function S2 = PartialSwapRounds(S,Ti,Rk,Rl)
   PorCambiar = Ti;
   Cambiados = [];
   cont = 1;
   while ( cont <= size(PorCambiar,"c")) then
      tamano = size(PorCambiar,"c");
      visitado = PorCambiar(cont);
      //si no ha sido cambiado
      if(size(find(Cambiados == visitado )) == 0) then
         PorCambiar(:,$+1) = abs(S(visitado,abs(Rk)));
         PorCambiar(:,$+1) = abs(S(visitado,abs(Rl)));
         S(visitado,[Rk Rl]) = S(visitado,[Rl Rk]); //swap
         Cambiados(:,$+1) = abs(visitado);
      end
      cont = cont + 1;
   end
   S2 = S;
endfunction
```

A.1.5 Función PartialSwapTeams

```
function S2 = PartialSwapTeams(S,Ti,Tj,Rk)
   //se agrega el primer elemento a la listaSwap
   PorCambiar = Rk;
   cont = 1;
   while(cont <= size(PorCambiar,"c")) then
      ronda = PorCambiar(cont);
      f_1 = abs(S(Ti,ronda)); // Equipo oponente afectado
      f_2 = abs(S(Tj,ronda)); // Equipo oponente afectado
```

```
            S([Ti Tj],ronda) = S([Tj Ti],ronda); // 1° swap
            //Segundo swap (de equipos afectados)
            S = SwapAux(S,f_1,f_2,ronda);
            lista_rondas = find(S(Ti,:) == S(Ti,ronda));
            if(size(lista_rondas) <> 0) then
               //revisar si no está en la lista PorCambiar,
               //si está no se agrega.
               ronda_a_agregar = lista_rondas(find(lista_rondas <> ronda));
               encontrado = find(PorCambiar == ronda_a_agregar);
                  if(size(ronda_a_agregar) <> 0) then
                     if(size(encontrado) == 0 ) then
                        PorCambiar(:,$+1) = ronda_a_agregar;
                     end
                  end
            end
            lista_rondas = find(S(Tj,:) == S(Tj,ronda));
            if(size(lista_rondas) <> 0) then
               //revisar si no está en la lista PorCambiar,
               //si está no se agrega.
               ronda_a_agregar = lista_rondas(find(lista_rondas <>
                                 ronda));
               encontrado = find(PorCambiar == ronda_a_agregar);
               if(size(ronda_a_agregar) <> 0) then
                  if(size(encontrado) == 0 ) then
                     PorCambiar(:,$+1) = ronda_a_agregar;
                  end
               end
            end
            cont = cont + 1;
         end
         S2 = S;
      endfunction
```

A.1.6 Función Swap Auxiliar

```
      //Función que permite hacer el swap entre 2 equipos sin cambiar las
      //localías
      function S2 = SwapAux(S,f1,f2,c)
         val_f1 = S(f1,c);
```

```
   val_f2 = S(f2,c);
   if( (val_f1 > 0 & val_f2 < 0 ) | (val_f2 > 0 & val_f1 < 0 )) then
      S(f1,c) = (val_f1*val_f2)/-val_f1;
      S(f2,c) = (val_f2*val_f1)/-val_f2;
   else
      S([f1 f2],c) = S([f2 f1],c);
   end
   S2 = S;
Endfunction
```

A.2 Funciones de los Movimientos para la Nueva Representación

A.2.1 Función SwapHomes

```
function S2 = SwapHomes(S,Ti,Tj)
     aux = S(Ti,Tj);
     S(Ti,Tj) = S(Tj,Ti);
     S(Tj,Ti) = aux;
     S2 = S;
endfunction
```

A.2.2 Función SwapRounds

```
function S2 = SwapRounds(S,Rk,Rl)
     nfilas = size(S,"r");
     for fila = 1 : nfilas
         pos = find(S(fila,:) == Rk);
         pos2 = find(S(fila,:) == Rl);
         if (size(pos,"c") == 1) then
             S(fila,pos) = Rl
         end
         if (size(pos2,"c") == 1) then
             S(fila,pos2) = Rk;
         end
     end
     S2 = S;
endfunction
```

A.2.3 Función SwapTeams

```
function S2 = SwapTeams(S,Ti,Tj)

     val_aux1 = S(Ti,Tj);

     val_aux2 = S(Tj,Ti);

     S([Ti Tj],:) = S([Tj Ti],:);

     S(:,[Ti Tj]) = S(:,[Tj Ti]);

     S(Ti,Ti) = 0;

     S(Tj,Tj) = 0;

     S(Ti,Tj) = val_aux1;

     S(Tj,Ti) = val_aux2;

     S2 = S;

endfunction
```

A.2.4 Función PartialSwapRounds

```
function S2 = PartialSwapRounds(S,Ti,Rk,Rl)

     ListaVisitados = [];

     Lista = Ti;

     cont = 1;

       while(cont <= size(Lista,"c")) then

            equipo = Lista(cont);

            if(size(find(ListaVisitados == equipo),"c") == 0 ) then

                 equipo_Rk = ObtenerEquipoNoVisitado(find(S(equipo,:)
                 == Rk),ListaVisitados);

                 equipo_Rl = ObtenerEquipoNoVisitado(find(S(equipo,:)
                 == Rl),ListaVisitados);

                 if(~isempty(equipo_Rk))  then

                       S(equipo,equipo_Rk) = Rl;

                       Lista(:,$+1) = equipo_Rk;

                 end

                 if(~isempty(equipo_Rl)) then

                       S(equipo,equipo_Rl) = Rk;
```

```
                        Lista(:,$+1) = equipo_Rl;
                    end
                    equipo_Rk = ObtenerEquipoNoVisitado(find(S(:,equipo)
== Rk),ListaVisitados);
                    equipo_Rl = ObtenerEquipoNoVisitado(find(S(:,equipo)
                    == Rl),ListaVisitados);
                    if(~isempty(equipo_Rk)) then
                        S(equipo_Rk,equipo) = Rl;
                        Lista(:,$+1) = equipo_Rk;
                    end
                    if(~isempty(equipo_Rl)) then
                        S(equipo_Rl,equipo) = Rk;
                        Lista(:,$+1) = equipo_Rl;
                    end
                end
                ListaVisitados(:,$+1) = equipo;
                cont = cont + 1;
            end
            S2 = S;
    endfunction
```

A.2.5 Función PartialSwapTeams

```
    function S2 = PartialSwapTeams(S,Ti,Tj,Rk)
       Lista = Rk;
       Cambiados = [];
       cont = 1;
           while ( cont <= size(Lista,"c")) then
                FilaTi = 0;
                FilaTj = 0;
                ColTi = 0;
                ColTj = 0;
                ronda = Lista(cont);
```

```
//Revisar fila del equipo Ti
        pos = find(S(Ti,:) == ronda)
        if(~isempty(pos)) then //pos != vacio
              pos = ObtenerNoCambiado(S,pos,Tj,Cambiados,%T);
              if(~isempty(pos)) then
                    ColTi = pos;
              end
        end
//Revisar columna del equipo Ti
        pos = find(S(:,Ti) == ronda)
        if(~isempty(pos)) then //pos != vacio
              pos = ObtenerNoCambiado(S,pos,Tj,Cambiados,%F);
              if(~isempty(pos)) then
                    FilaTi = pos;
              end
        end
//Revisar fila del equipo Tj
        pos = find(S(Tj,:) == ronda)
        if(~isempty(pos)) then //pos != vacio
              pos = ObtenerNoCambiado(S,pos,Ti,Cambiados,%T);
              if(~isempty(pos)) then
                    ColTj = pos;
              end
        end
//Revisar columna del equipo Tj
        pos = find(S(:,Tj) == ronda)
        if(~isempty(pos)) then //pos != vacio
              pos = ObtenerNoCambiado(S,pos,Ti,Cambiados,%F);
              if(~isempty(pos)) then
                    FilaTj = pos;
```

```
            end
        end
        if(ColTi <> 0 & S(Tj,ColTi) <> 0) then
            Lista(:,$+1) = S(Tj,ColTi);
            aux = S(Ti,ColTi);
            S(Ti,ColTi) = S(Tj,ColTi);
            S(Tj,ColTi) = aux;
        end
        if(ColTj <> 0 & S(Ti,ColTj)) then
            Lista(:,$+1) = S(Ti,ColTj);
            aux = S(Ti,ColTj);
            S(Ti,ColTj) = S(Tj,ColTj);
            S(Tj,ColTj) = aux;
        end
        if(FilaTi <> 0 & S(FilaTi,Tj)) then
            Lista(:,$+1) = S(FilaTi,Tj);
            aux = S(FilaTi,Ti);
            S(FilaTi,Ti) = S(FilaTi,Tj);
            S(FilaTi,Tj) = aux;
        end
        if(FilaTj <> 0 & S(FilaTj,Ti)) then
            Lista(:,$+1) = S(FilaTj,Ti);
            aux = S(FilaTj,Ti);
            S(FilaTj,Ti) = S(FilaTj,Tj);
            S(FilaTj,Tj) = aux;
        end
        cont = cont + 1;
        Cambiados(:,$+1) = ronda;
    end
    S2 = S;
```

```
endfunction
```

A.3 Funciones de Cálculo de Costo

A.3.1 Función Cálculo de Costo de una Programación Factible Representación Anagnostopoulos

```
function costoTotal = CalcularCosto(S,C)
   costoTotal = 0;
   contFila = 1;
   //recorrer por filas
   while( contFila <= size(S,"r")) then
      home = contFila;
      costoEquipo = 0;
      posActual = home;
      contColumna = 1;
      //recorrer por columnas
      while (contColumna <= size(S,"c")) then
         rival = S(home,contColumna);
         if(rival < 0) then // visita (ir a donde el rival)
            costo = C(posActual,abs(rival));
            posActual = abs(rival);
         else //local (volver a casa)
            costo = C(posActual,home);
            posActual = home;
         end
            costoEquipo = costoEquipo + costo;
            contColumna = contColumna + 1;
      end
      //suma volver a casa
      costoEquipo = costoEquipo + C(posActual,home);
      costoTotal = costoTotal + costoEquipo;
      contFila = contFila + 1;
   end
endfunction
```

A.3.2 Función Cálculo de Costo de una Programación Factible Nueva Representación

```
function costoTotal = CalcularCosto(S,C)
    n_equipo = size(S,"c");
```

```
    n_round = 2*n_equipo-2;
    costoTotal=0;
    for i = 1 : n_equipo
        anterior = i;
        costo_eq = 0;
        for j = 1 : n_round
           if (size(find(S(i,:) == j)) <> 0)
              actual = i;
           else
              eq = find(S(:,i) == j);
              actual = eq;
           end
           costo_eq = costo_eq + C(anterior,actual);
           anterior = actual;
        end
        costo_eq = costo_eq + C(actual,i);
        costoTotal = costoTotal + costo_eq;
     end
endfunction
```

A.3.3 Función Cálculo de la Función Sublineal

```
function R = F(nbv)
   R = 1 + sqrt(nbv)*log(nbv)/2;
endfunction
```

A.3.4 Función Cálculo de Costo de una Programación Infactible

```
function C2 = CalcularCostoInfactible(S,C,w)
   CostoS = CalcularCosto(S,C);
   num_violaciones = nbv(S);
   C2 = sqrt(CostoS**2 + (w*F(num_violaciones))**2);
Endfunction
```

A.4 Funciones de Cálculo de Violaciones

A.4.1 Función que Cuenta el Número de Violaciones a-lo-mas Representación Anagnostopoulos

```
function nviolaciones = CuentaALoMas(S)
   nviolaciones = 0;
   //total de columnas - 3 (para no sobrepasar el indice)
   //ncolumnas + 3 = totalcolumnas
   nColumnas = size(S,"c") - 3;
   nFilas = size(S,"r");
   contFilas = 1;
   while (contFilas <= nFilas ) then
      contColumnas = 1;
      while (contColumnas <= nColumnas) then
         cantNegativos =
 size(find(S(contFilas,[contColumnas:contColumnas+3]) < 0),"c");
             cantPositivos =
 size(find(S(contFilas,[contColumnas:contColumnas+3]) > 0),"c");
         if (cantNegativos > 3) then
            nviolaciones = nviolaciones + 1;
         end
         if (cantPositivos > 3 ) then
            nviolaciones = nviolaciones + 1;
         end
         contColumnas = contColumnas + 1;
      end
      contFilas = contFilas + 1;
   end
endfunction
```

A.4.2 Función que Cuenta el Número de Violaciones sin-repetición

```
function nviolaciones = CuentaSinRepeticion(S)
   nviolaciones = 0;
   //total de columnas - 1 (para no sobrepasar el índice)
   //ncolumnas + 1 = totalcolumnas
   nColumnas = size(S,"c") - 1;
   nFilas = size(S,"r");
   contFilas = 1;
   while (contFilas <= nFilas ) then
```

```
      contColumnas = 1;
      while (contColumnas <= nColumnas) then
             cant = size( find( abs(
                    S(contFilas,[contColumnas:contColumnas+1])) ==
                    abs(S(contFilas,contColumnas))),"c");
             if (cant == 2) then
                nviolaciones = nviolaciones + 1;
             end
             contColumnas = contColumnas + 1;
      end
      contFilas = contFilas + 1;
   end
endfunction
```

A.4.3 Función que Cuenta el Número de Violaciones a-lo-mas Nueva Representación

```
function nviolaciones = CuentaALoMas(S)
    nviolaciones=0;
    n_equipos = size(S,"c");
    n=2*n_equipos-2;
         for j=1 : n_equipos
             for i=1 : n-3
                   tam=find(S(j,:)>=i & S(j,:)<= (i+3))
                   if (size(tam,"c")==4)
                         nviolaciones=nviolaciones+1;
                   end
                   tam2=find(S(:,j)>=i & S(:,j)<= (i+3))
                   if (size(tam2,"c")==4)
                         nviolaciones=nviolaciones+1;
                   end
             end
         end
endfunction
```

A.4.4 Función que Cuenta el Número de Violaciones sin-repetición Nueva Representación

```
function nviolaciones = CuentaSinRepeticion(S)
    nviolaciones=0;
    nequipos=size(S,"c");
    for i=1: nequipos-1
             for j=1:nequipos
             if (abs(S(i,j) - S(j,i))==1)
                   nviolaciones=nviolaciones+1;
             end
       end
    end
endfunction
```

A.4.5 Función Cálculo Total de Violaciones

```
function nviolaciones = nbv(S)
   nviol_alomas = CuentaALoMas(S);
   nviol_norepeat = CuentaSinRepeticion(S);
   nviolaciones = nviol_alomas + nviol_norepeat;
endfunction
```

A.5 Funciones Generales

A.5.1 Función Aplicar Movimiento Aleatorio

```
function S2 = AplicarMovAleatorio(S, nequipos, nrounds)
   random = int(rand()*5) + 1; //numero aleatorio entre 1 y 5
   // equipo 1 aleatorio
   e1rand = int(rand()*nequipos) + 1;
   // equipo 2 aleatorio
   e2rand = int(rand()*nequipos) + 1;
   while(e2rand == e1rand) do
      e2rand = int(rand()*nequipos) + 1;
   end
      r1rand = int(rand()*nrounds) + 1; // round 1 aleatorio
      r2rand = int(rand()*nrounds) + 1; // round 2 aleatorio
   while( r2rand == r1rand) do
      r2rand = int(rand()*nrounds) + 1;
```

```
   end
   select random
      case 1 then S2 = SwapHomes(S,e1rand,e2rand);  break;
      case 2 then S2 = SwapTeams(S,e1rand,e2rand);  break;
      case 3 then S2 = SwapRounds(S,r1rand,r2rand); break;
      case 4 then S2 = PartialSwapRounds(S,e1rand,r1rand,r2rand);
       break;
   else S2 = PartialSwapTeams(S,e1rand,e2rand,r1rand); break;
   end
endfunction
```

A.5.2 Función de Probabilidad

```
function P = Probabilidad(CostoS,CostoS2,Temperatura)
   P = exp(-(CostoS2-CostoS)/Temperatura);
Endfunction
```

A.5.3 Programa Principal

```
function ProgramaPrincipal()
   //Inicialización de variables
   nequipos = 6;
   nrounds = nequipos*2-2;
   mejorFactible = %inf;
   nbf = %inf;
   mejorInfactible = %inf;
   nbi = %inf;
   maxRecalentamiento = 10;
   maxFase = 10;
   maxContador = 30;
   temperatura = 400;
   mejorTemperatura = temperatura;
   w = 4000;
   alfa = 1.04;
   gama = 1.04;
   betta = 0.99;
   printf("Costo S Inicial: %d\n",CalcularCosto(S,C));
   recalentamiento = 0;
   while recalentamiento <= maxRecalentamiento then
      fase = 0;
```

```
    while fase <= maxFase then
       contador = 0;
       while contador <= maxContador then
          S2 = AplicarMovAleatorio(S,nequipos,nrounds);
          if(nbv(S2) > 0) then
             costoS2 = CalcularCostoInfactible(S2,C,w);
          else
          costoS2 = CalcularCosto(S2,C);
          end
          if(nbv(S) > 0) then
             costoS = CalcularCostoInfactible(S,C,w);
          else
             costoS = CalcularCosto(S,C);
          end
          nbvS2 = nbv(S2);
          if((costoS2 < costoS) | (nbvS2 == 0 & costoS2 <
mejorFactible) | (nbvS2 > 0 & costoS2 < mejorInfactible)) then
             aceptar = %T;
          else
             pro = Probabilidad(costoS,costoS2,temperatura);
             if(pro > 0.5) then
                aceptar = %T;
             else
             aceptar = %F;
             end
          end
          if aceptar then
             S = S2;
             nbvS = nbvS2;
             costoS = costoS2;
             if(nbvS == 0) then
                nbf = min(costoS,mejorFactible);
             else
                nbi = min(costoS,mejorInfactible);
             end
             if(nbf < mejorFactible | nbi < mejorInfactible) then
                recalentamiento = 0;
                contador = 0;
```

```
                    fase = 0;
                    mejorTemperatura = temperatura;
                    mejorFactible = nbf;
                    mejorInfactible = nbi;
                    if(nbvS == 0) then
                       w = w/alfa;
                    else
                       w = w*gama;
                    end
                 end
              else
                 contador = contador + 1;
              end
           end
           fase = fase + 1;
           temperatura = temperatura*betta;
        end
        temperatura = 2*mejorTemperatura;
        recalentamiento = recalentamiento + 1;
     end
     printf("Mejor Factible: %d\n",mejorFactible);
     printf("Mejor Infactible: %d\n",mejorInfactible);
endfunction
```