UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INFORMÁTICOS

MÁSTER UNIVERSITARIO EN INTELIGENCIA ARTIFICIAL

ASIGNACIÓN DE CONTROLADORES A SECTORES AÉREOS MEDIANTE METAHEURÍSTICAS TRAYECTORIALES: BÚSQUEDA TABÚ

Trabajo Fin de Master

Adán Suárez Cuesta

Junio, 2017

TRABAJO FIN DE MASTER

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INFORMÁTICOS

MÁSTER UNIVERSITARIO EN INTELIGENCIA ARTIFICIAL

ASIGNACIÓN DE CONTROLADORES A SECTORES AÉREOS MEDIANTE METAHEURÍSTICAS TRAYECTORIALES: BÚSQUEDA TABÚ

Trabajo Fin de Master

Autor: Adán Suárez Cuesta

Director: Dr. Alfonso Mateos Caballero

Junio, 2017

Adán Suárez Cuesta

Madrid – Spain

E-mail: adan.suarezcu@alumnos.upm.es
E-mail: adansuarezcu@gmail.com

Resumen

En este documento abordamos la asignación de controladores a sectores aéreos en un problema de timetabling. En nuestro problema existen una serie de condiciones de obligado cumplimiento y otras condiciones deseables. Estos objetivos tienen que ver con los periodos de trabajo y descanso de los controladores así como con sus posiciones de trabajo, la estructura de la solución y el número de cambios de sala. Además se trata de buscar que la carga de trabajo entre los distintos controladores esté repartida uniformemente.

Se propone una metodología basada en tres fases. En la primera fase, se utiliza una heurística para obtener soluciones iniciales infactibles basadas en las plantillas proporcionadas. A continuación, en la segunda fase, se usa un algoritmo MIR (Multiple Independent Run) basado en recocido simulado para tratar de alcanzar soluciones factibles. Por último, en la tercera fase, usaremos la búsqueda tabú a partir de las soluciones factibles obtenidas en la segunda fase con el objetivo de optimizar las funciones objetivo. Para ello tenemos en cuenta información ordinal sobre estos, de esta manera transformamos el problema multiobjetivo de optimización en uno de un solo objetivo usando el método del centroide. Adicionalmente donde se pueda usamos expresiones regulares para comprobar las condiciones de trabajo de los controladores, permitiéndonos realizar ésto con rapidez.

Nuestro software, el cual implementa este enfoque, nos ha permitido obtener resultados reales que mejoran los proporcionados por una plantilla de referencia dada.

Abstract

In this document we address a work-shift scheduling problem regarding the assignment of air traffic controllers (ATC) to air sectors. In this problem there exists a series of constraints that must be fulfilled, in addition there are other conditions that are desirable. These objectives deal with the work and rest periods of the ATC and their positions, the structure of the solution and the number of control center changes. In addition we try to balance the workload of the different ATCs.

We propose a methodology based on three phases. In the first phase, a heuristic is used to obtain diferent feasible or unfeasible initial solutions based on some given templates. Next, in the second phase, we use a MIR (Multiple Independent Run) algorithm based on simulated annealing with the purpose of reaching feasible solutions. Last, in the third phase, we use tabu search on the feasible solutions given by the previous phase with the goal of optimizing them regarding the objetive functions. In order to do this, we have into account ordinal information about them and we can transform this multiobjetive optimization problem into a single optimization problem using the rank order centroid method. In addition we will use regular expressions, if possible, to check if the work conditions of the ATCs are met, allowing us to do this quite fast.

Our own software implementing this approach allowed us to obtain results in a real-world problem that improve the ones given by a reference template.

Agradecimientos

En primer lugar quiero agradecer a mi familia por todo su apoyo y su esfuerzo, ya que son los responsables de que sea la persona que soy.

También me gustaría agradecer a mis profesores y en especial a mi tutor Alfonso, cuya ayuda y dirección junto con la de Tino Tello me han sido muy útiles para lograr un buen resultado en este proyecto.

También quiero dar las gracias a mis amigos por todo su apoyo.

Adán

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Los controladores aéreos juegan un rol muy importante en la sociedad actual, su principal labor es facilitar el flujo del tráfico aéreo y el de movimientos dentro de los aeropuertos, a la vez que evitan colisiones entre las aeronaves.

Para un correcto desempeño de esta actividad la asignación de los controladores aéreos a su puesto de trabajo debe ser realizada correctamente, evitando así por ejemplo problemas como que exista un número deficiente de controladores. A esto además se añaden una serie de restricciones por ley, como por ejemplo las horas de descanso. Desde CRIDA se propone crear un software que organice a los controladores en su posición de trabajo, y que respete estas restricciones y se acerque a unas condiciones deseables. Para ello primero veamos algunos detalles sobre como se organiza el espacio aéreo.

El espacio aéreo está dividido en volúmenes (unidad elemental del espacio aéreo). Como la capacidad de trabajo de un controlador aéreo es limitada, el espacio aéreo se divide en sectores. Un sector puede estar formado por uno o varios volúmenes y son operados por dos controladores en la posición de ejecutivo y planificador.

Una sectorización es el conjunto de todos los sectores en un momento determinado, los cuales deben cubrir todos los volúmenes del espacio aéreo. La sectorización variará a lo largo del día dependiendo del volumen del tráfico aéreo. Así pues, si hay más tráfico habrá más sectores abiertos y por lo tanto se necesitarán más controladores.

Una vez realizada la sectorización surge el problema de determinar el mínimo número de controladores necesarios que cubran estos sectores y que a la vez cumplan una serie de restricciones (por ejemplo, periodos de trabajo y descanso).

A esto, además, se añaden una serie de condiciones deseables, las cuales nos obligan a optimizar las posiciones y horarios que ocupan dicho número de controladores. Por lo tanto, el problema que vamos a resolver, se reduce a un problema de timetabling.

Por definición, un problema de timetabling es aquel en el que se requiere la asignación de tiempo y recursos a eventos donde varios conjuntos de condiciones necesarias y deseables deben ser tenidos en cuenta. El tamaño y complejidad de estos problemas combinatorios los hacen difíciles e incluso imposibles de resolver por métodos exactos. A continuación,

hablaremos de los problemas de timetabling de forma detallada.

1.1 Problemas de timetabling

Esta clase de problemas de timetabling está muy extendida actualmente. Existen numerosas alternativas que tratan con este tipo de problema. Incluso existe una competición internacional, ITC (*International Timetabling Competition*) orientada a estimular la investigación en este área a la vez que anima a integrar los frutos del trabajo de investigación con la práctica ofreciendo instancias reales de este tipo de problemas para su solución. A partir de la competición de 2011 se motivó el desarrollo de varios enfoques para el problema XHSTT (*High School TimeTabling XML based Data Format*), que básicamente es un formato para el problema de timetabling en institutos cuyo principal objetivo es contener estos datasets a la vez que también puede albergar soluciones (Post et al., 2014).

En Babaei et al. (2015) se realiza un estudio y se analizan distintas metaheurísticas u otros métodos novedosos más inteligentes además de un enfoque distribuido basado en sistemas multiagente aplicado al timetabling en las universidades.

En Días et al. (2003) se proponen diferentes heurísticas basadas en algoritmos evolutivos y búsqueda local para resolver el problema de los horarios de los enfermeros en hospitales grandes.

En Ribeiro (2012) se proporciona un análisis de distintos tipos de problema en el ámbito de la planificación deportiva acompañado de un estudio de distintos métodos de optimización aplicados a estos problemas en distintas ligas deportivas.

En el contexto del tráfico aéreo, en Arving et al. (2006) se sugiere que la correcta o no distribución de los controladores tiene consecuencias tanto en la salud, como en la seguridad, la productividad y la eficiencia, además de implicaciones sociales. También se referencia a la normativa europea que regula el tiempo de trabajo para estos trabajadores.

En EUROCONTROL (2006) aparece un estudio sobre los turnos de trabajo, concluyendo que a pesar de que existan varias herramientas, son muy costosas y no se adaptan correctamente a las necesidades.

En Stojadinovic (2015) se presenta un híbrido que usa un enfoque de resolución SAT (*Boolean Satisfiability Problem*) y el método *hill climbing*. En esta propuesta primero se usa la resolución SAT para generar una solución factible. Después, hill climbing se usa para mejorar esta solución. Finalmente SAT se usa para mejorar la solución identificada arreglando partes de la solución. Se repite este proceso hasta lograr una solución óptima.

Una versión simplificada del problema que se va a abordar ya fue solucionada en Mateos et al. (2017).

En nuestro caso se buscará una solución que sea factible y a continuación, moviéndonos

por soluciones factibles usaremos la búsqueda tabú para optimizar la solución. Al ser la búsqueda tabú una metaheurística la siguiente sección está dedicada a ellas.

1.2 Metaheurísticas

Para resolver un problema, es común hacer uso de una heurística, la cual consiste en una técnica orientada a resolver un problema más rápido cuando los métodos clásicos son lentos, o que permite encontrar una solución aproximada cuando los clásicos no son capaces de encontrar una exacta. En ella se sacrifican variables como la calidad o la precisión a favor de la velocidad.

A un nivel superior, se puede hacer uso de las metaheurísticas, que sirven para controlar y ajustar los algoritmos heurísticos, normalmente haciendo uso de la memoria y el aprendizaje.

Una metaheurística es un procedimiento diseñado para encontrar, generar o seleccionar una heurística que puede proporcionar una solución suficientemente buena para un problema de optimización, especialmente con información que esté incompleta o imperfecta o donde exista una limitación de capacidad computacional.

Existen numerosos tipos de metaheurísticas, por un lado están las que están basados en poblaciones, donde existe una población inicial y en cada iteración se seleccionan el mejor o mejores individuos para después aplicar unos cambios sobre éstos con el objetivo de mejorar la solución, y repetir el proceso. Un ejemplo de este tipo son los algoritmos evolutivos y genéticos (Obit, 2010). Los algoritmos evolutivos se inspiran en los mecanismos de evolución natural e incluyen tres pasos:

1. *Selección*: Se seleccionan los individuos de mayor fitness como padres de la siguiente generación.
2. *Reproducción*: Se realizan operaciones de cruce y mutación sobre los padres seleccionados en la primera fase.
3. *Reemplazo*: Los individuos de la población original son reemplazados por los nuevos.

Otro tipo de algoritmos conocidos como algoritmos genéticos se basan en los siguientes pasos:

1. Generar la población inicial.
2. Evaluar la población generada.
3. Seleccionar unos padres para el cruce basándose en la información del paso anterior.
4. Aplicar los operadores de cruce para obtener una descendencia.
5. Aplicar un operador de mutación a los hijos.
6. Seleccionar padres e hijos que constituyan la nueva población.

7. Si se satisface la condición de parada el algoritmo se detiene, si no, se vuelve al Paso 2.

En Khonggamnerd and Innet (2009) y Alsmadi et al. (2011) se aplican estos algoritmos al problema de timetabling.

Los algoritmos de optimización basados en colonias de hormigas (Obit, 2010) están inspirados por el comportamiento que siguen las hormigas en las rutas cuando buscan comida. Estas se mueven aleatoriamente y van dejando un rastro de feromonas. Otras hormigas siguen esta ruta y si llegan a la comida, vuelven a casa dejando un rastro encima del camino usado.

Las feromonas se evaporan ligeramente y el camino va perdiendo interés para la siguiente hormiga. El objetivo de ésto consiste en encontrar el camino más corto.

En Socha et al. (2002) y Mayer et al. (2008) se aplican las colonias de hormigas a problemas de timetabling.

Otra posibiidad consiste en usar algoritmos meméticos (Obit, 2010), se considera un meme como la unidad de información que se genera a sí misma cuando las creencias de la gente cambian. Un meme es diferente de un gen porque cuando se transmite entre personas, cada persona lo adapta si parece bueno, mientras que el gen se transmite sin cambios. En este caso el espacio de posibles soluciones se reduce a la búsqueda local y se combinan los algoritmos de *hill climbing* y genéticos.

En Shengxiang and Jat (2011) se aplican los algoritmos meméticos para resolver problemas de timetabling.

Otro tipo de metaheurísticas son las que usan una sola solución para analizar el problema en lugar de usar una población inicial. Búsqueda tabú, recocido simulado, búsqueda local y VNS (*Variable Neighbourhood Search*) forman parte de este grupo.

La búsqueda tabú (Glover and Melián, 2003) parte de una solución inicial y a continuación se selecciona el mejor vecino. Si este vecino no está en la lista tabú el algoritmo se mueve hacia esa nueva solución. Si esta solución es la mejor encontrada, se puede realizar el movimiento igualmente. Si no se puede realizar el cambio se elige el siguiente mejor vecino. A continuación se actualiza la lista tabú, incluyendo al último movimiento para evitar dar un paso atrás y que aparezcan ciclos.

La busqueda tabú se aplicó en Álvarez et al. (2002) en relación con el timetabling por primera vez para asignar estudiantes a los grupos de clase buscando una alta calidad y balance en el número de estudiantes que se registran en un grupo, logrando buenos resultados.

El recocido simulado (Obit, 2010) se inspira en la física y en el calentamiento de los sólidos. Este enfoque evita quedar atrapado en óptimos locales y usa métodos de búsqueda local. Las soluciones obtenidas por búsqueda local reemplazan la solución actual frecuentemente.

Se crea al comienzo una solución inicial aleatoria, y se reemplaza por otra solución aleatoria que puede ser probabilísticamente la solución óptima. El proceso de búsqueda empieza con una alta temperatura que va decreciendo. Posee varios parámetros como la temperatura inicial y el valor de enfriamiento que varían según el problema.

En Aycan et al. (2008) se aplica recocido simulado para resolver problemas de *timetabling*.

La búsqueda local (Lewis, 2006) (Obit, 2010) busca encontrar una solución que maximiza un criterio entre un número de soluciones que continúan. Estos algoritmos se mueven de una solución a otra en un espacio de soluciones haciendo uso de cambios limitados hasta que una solución deseable es encontrada o pase un periodo de tiempo. El algoritmo de búsqueda local se mueve de una solución hacia vecinos definidos en la adyacencia del espacio de búsqueda y la selección de cada solución se realiza usando información acerca de las soluciones vecinas de manera que se maximize el criterio localmente.

En Joudaki et al. (2010) y Shengxiang and Jat (2011) se aplica búsqueda local en la resolución de problemas de *timetabling*.

Por último, VNS (algoritmo de búsqueda en vecindario variable), propuesto en Mladenovic and Hansen (1997) propone hacer uso de cambios en el vecindario de la búsqueda. Explora vecindarios más alejados de la solución y sólo salta a una nueva solución si existe una mejora.

1.3 Objetivo

El objetivo consiste en crear un software que resuelva este problema de asignaciones de controladores y que mejore una solución de referencia (hecha por un experto), proporcionada por CRIDA.

1.4 Descripción del problema

Existen dos tipos de posición de los controladores. En la posición ejecutiva, un controlador se encarga de hablar con las aeronaves y darles instrucciones a los pilotos para evitar situaciones de conflicto entre los aviones. En la posición de planificador, la principal misión es anticipar los posibles conflictos entre las aeronaves y comunicarlo al ejecutivo para que resuelva la situación antes de que ocurra.

Como se mencionó anteriormente, debido a la capacidad limitada de trabajo el espacio aéreo se divide en sectores operados cada uno por un ejecutivo y un planificador.

El conjunto de sectores abiertos en cualquier momento debe cubrir todos los volúmenes correspondientes al espacio aéreo y recibe el nombre de sectorización, la cual cambia según el tráfico que haya. A más tráfico, los sectores deben ser menores (estar constituidos por un número inferior de volúmenes) y por lo tanto su número aumenta, aumentando también el

número de controladores necesarios para controlarlos. Por lo tanto, los sectores son divididos y unidos dinámicamente a lo largo del tiempo dependiendo del tráfico aéreo, y el número de controladores necesario para cubrir los sectores abiertos también variará.

Si entendemos sectorización como el conjunto de sectores operados en cada periodo de tiempo a lo largo de un turno, ésta consiste en una entrada del problema, la cual nos indica la lista de los sectores abiertos y los intervalos de tiempo que estos permanecen abiertos en cada intervalo. Una sectorización también incluirá los sectores abiertos durante el turno largo con intersección del turno de noche que se vayan a operar en ese turno.

Por otro lado, un núcleo es un conjunto de sectores. Pueden existir sectores que pertenezcan a la vez a dos núcleos. Un controlador se habilita para operar en un determinado núcleo.

Dos sectores se denominan afines si poseen algún volumen en común. Esto es importante de saber ya que permite omitir alguna de las restricciones existentes, por ejemplo las relacionadas con la necesidad de descansar o no al realizar un cambio de sector por parte de un controlador. La afinidad entre sectores viene dada por la matriz de afinidad explícitamente, aunque implícitamente también se puede deducir de las listas de volúmenes que constituyen los sectores mediante la intersección de las correspondientes parejas de las listas.

Los sectores tienen asignadas posiciones de control (ejecutivo/planificador).

Cada controlador sólo puede controlar los sectores del núcleo para el que esté habilitado. Lo mismo que sólo puede controlar los tipos de sectores para los que tenga capacitación. Los sectores pueden ser de dos tipos: Ruta + Aproximación (PTD) o sólo ruta (CON).

Dependiendo de la unidad de control, si hay varios núcleos y éstos no comparten sectores la asignación de controladores puede resolverse por separado para cada núcleo.

Existen distintos turnos de trabajo, y cada uno de éstos tiene sus propias restricciones. Hay 3 tipos de turnos: Turno de mañana, turno de tarde y turno de noche. En los turnos de mañana o tarde cada controlador podrá tener asignado un turno de trabajo coincidente con el turno de control a resolver (turno corto), o un turno de trabajo extendido que cubre el turno de control a resolver más una parte del turno de noche (turno largo). Las horas de inicio y fin son variables y forman parte de la entrada del problema, al igual que el tipo de turno.

Los controladores son los encargados de operar en los distintos sectores. Cada controlador estará asignado a un núcleo de sectores y únicamente podrá ser asignado a los sectores pertenecientes al núcleo. Así pues, cada uno tendrá una capacitación que podrá ser «PTD» (ruta + Aproximación), «CON» (solo ruta). Es esta capacitación la que permite que un controlador esté asignado a un sector u otro.

En caso de que los turnos no se dividan en turno largo y corto (por ejemplo el nocturno), se considera que sólo existe el turno corto (Tc).

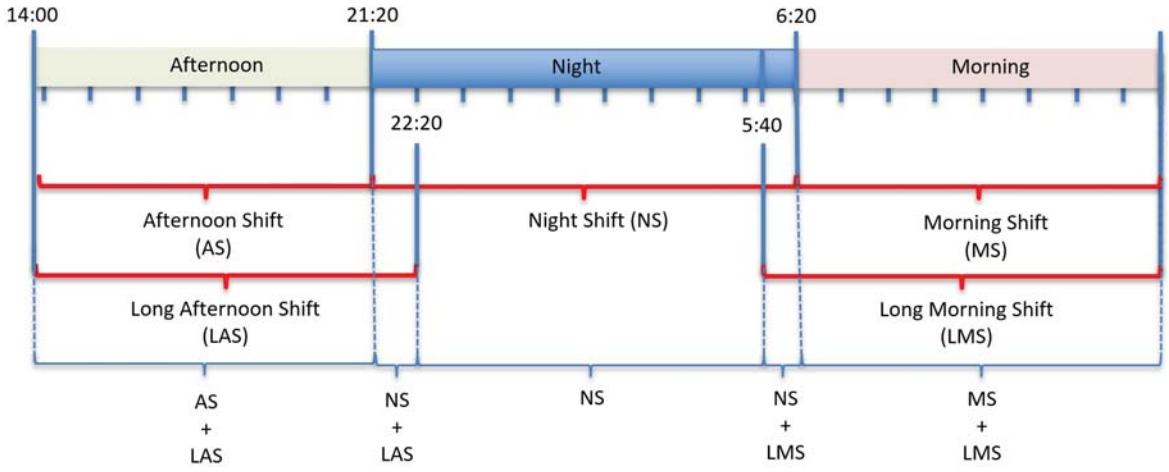

Figura 1.1: Diferentes turnos de trabajo

Para el algoritmo, en este caso, ya se considera un número fijo dado de controladores que debe cubrir cierta sectorización.

Nuestro problema tiene varias restricciones y condiciones deseables. Las restricciones que hay que cumplir son las siguientes:

1. Todas las posiciones de control deben estar cubiertas por controladores de manera exclusiva, exhaustiva y bajo las restricciones definidas.
2. Una determinada posición podrá ser asignada a un controlador si el controlador está habilitado en el núcleo al que pertenece el sector, si éste es un sector común, independientemente de la sectorización por la que el sector se encuentre abierto.
3. A un controlador tipo PTD podrá asignársele una posición cuyo sector sea Aproximación o ruta.
4. A un controlador tipo CON podrá asignársele una posición cuyo sector sea Ruta.
5. Porcentaje de tiempo de descanso mínimo en turno diurno (mañana y tarde), incluyendo turnos largos: 25 %.
6. Porcentaje de tiempo de descanso mínimo en el turno de noche: 33 %.
7. A los sectores abiertos durante todo el turno de noche se le asignarán cuatro controladores. Estos controladores no podrán operar otro sector distinto en ese turno.
8. En los turnos de mañana, tarde y noche, no es posible un periodo de trabajo continuo mayor de dos horas en los que el controlador no realice ningún periodo de descanso.
9. Un controlador solo puede operar en su turno correspondiente, si pertenece al turno largo, en el turno largo y si pertenece al turno corto en el turno corto.
10. En los turnos de mañana, tarde y noche no puede existir ningún periodo de dos horas y media en los que un controlador realice un periodo total de descanso menor de media hora. Es decir, dentro de una ventana de tiempo de dos horas y media un controlador

debe tener mínimo 30 minutos de descanso, sin ser necesario que estos se realicen de forma continua.

11. Un controlador no puede cambiar de posición de control de una posición ejecutiva de un determinado sector a una posición ejecutiva de otro sector diferente, sin que exista un descanso entre medias, a no ser que ambos sectores sean afines (cambio de configuración). Hay que tener en cuenta de que si no hay cambio de configuración de sectores no es posible que dos sectores diferentes posean volúmenes comunes.
12. A la misma línea de controlador todos los sectores asignados deben pertenecer al mismo núcleo. Porque si no, no se podría asignar ningún controlador a dicha línea.
13. El tiempo mínimo de trabajo continuado, es decir, el tiempo de trabajo de un controlador entre dos descansos debe ser de 15 minutos.
14. El tiempo mínimo de descanso de un controlador es de 15 minutos.
15. El tiempo mínimo en posición de un controlador es de 15 minutos.
16. Número máximo de sectores por los que rota un controlador: 3 (CRIDA dispone de un algoritmo que calcula el número de sectores por los que pasa un controlador, teniendo en cuenta las posibles afinidades).
17. Los sectores o agrupaciones de dos sectores que se indique en la entrada del problema, se deberán cubrir con 4 controladores en el turno de noche. Hay que tener en cuenta que puede existir más de un sector o agrupación de sectores que se deban cubrir con 4 controladores.
18. Un sector perteneciente a varios núcleos, puede estar cubierto por cualquier controlador con acreditación de uno de los núcleos.

Las condiciones deseables son las siguientes y vienen ordenadas por orden de importancia:

1. Condiciones de elección de soluciones:
   a) Tiempo óptimo de trabajo en posición. Tiempo en el que un controlador se encuentra sin cambio de sector y tipo de control: 45 minutos.
   b) Tiempo óptimo de trabajo entre descansos: 90 minutos.
   c) El porcentaje de tiempo que un controlador trabaja en posiciones ejecutivas debe estar comprendido entre el 40 y el 60 % del trabajo total realizado (sin contar descansos).
2. Mantener una estructura similar a los estadillos/plantillas, facilitando el entendimiento de la solución proporcionada.
3. Minimizar el número de cambios de sala:
   a) Minimizar el número de intervalos de descansos.

b) Maximizar el número de sectores que influyen en las acreditaciones.

4. Distribución homogénea de la carga de trabajo entre los distintos controladores.

1.5 Estructura del documento

Capítulo 2: Método de solución

En este capítulo se describirán las distintas fases que plantea nuestro enfoque, destacando los aspectos más relevantes de éste como pueden ser la modelización de la solución, los movimientos del entorno o como se calcula el fitness y se modelan las restricciones y condiciones deseables así como el método que seguimos para comprobar la factibilidad.

Capítulo 3: Resultados y ajuste paramétrico

En este capítulo se presentan los resultados obtenidos y a la vez como se ha hecho el ajuste paramétrico hasta lograr llegar a una configuración equilibrada tanto en el tiempo como en el coste y la calidad de la solución.

Capítulo 4: Conclusiones y trabajo futuro

En este capítulo figuran las conclusiones que hemos obtenido así como ciertas sugerencias para continuar con el proyecto en un futuro.

Capítulo 2

Método de solución

En este capítulo se describirán las distintas fases que plantea nuestro enfoque, destacando los aspectos más relevantes de éste como pueden ser la modelización de la solución, los movimientos del entorno o como se calcula el *fitness* y se modelan las restricciones y condiciones deseables así como el método que seguimos para comprobar la factibilidad.

2.1 Modelización de la solución

Una solución es representada por una matriz donde las filas son los controladores aéreos y las columnas son los intervalos de tiempo en los que se divide el turno.

Por lo tanto, las dimensiones de esta matriz vienen definidas por la entrada del problema, es decir, el número de columnas viene determinado por el tiempo que dura el turno que se va a resolver. El número de filas viene determinado por el número de controladores disponibles, que en nuestro caso ya conocemos porque es una de las entradas del problema.

El número de columnas se puede calcular una vez que sabemos la duración del turno que se va a resolver, por ejemplo, si este es de 8 horas, y lo dividimos entre la longitud del intervalo que son 5 minutos, tendríamos 96 slots/columnas.

Hay que tener en cuenta que los controladores deben ocupar todos los sectores abiertos en ambas posiciones (ejecutivo y planificador) para cada slot.

La solución se codificará de la siguiente manera:

1. Sectores: La codificación de los sectores se realizará con un conjunto de 3 letras: (aaa,aab,aac,aad,...,zzz).
2. Posiciones: Un sector tiene que estar cubierto en dos posiciones, así pues, la posición de Ejecutivo se representará con mayúsculas y la de Planificador con minúsculas.
3. Descansos: Cuando un controlador se encuentra descansando se representaría como 111.

En el siguiente cuadro se muestra un ejemplo de como se representaría una solución con la codificación descrita anteriormente:

	7:00				9:30			...	13:20	
	5				2			...	4	
	S1	S2	S3	S4	S5	S6	S7	...	Si-1	Si
Cntr1	aaa	aaa	aaa	aaa	aaa	aaa	aaa	...	aaa	aaa
Cntr2	AAA	AAA	AAA	AAA	AAA	AAA	111	...	AAA	AAA
Cntr3	111	111	111	111	111	111	AAA	...	111	111
Cntrr4	aab	aab	aab	aab	aab	aab	aab	...	aab	aab
Cntr5	AAB	AAB	AAB	AAB	AAB	AAB	111	...	AAB	AAB
Cntr6	111	111	111	111	111	111	AAB	...	111	111
...	...	...	...	...	...	...	...	...	...	...
CntrJ	111	111	111	111	111	111	XXX	...	111	111

Tabla 2.1: Ejemplo de como se representa una solución con nuestra codificación

2.2 Pasos del algoritmo

El algoritmo se compone de 3 fases:

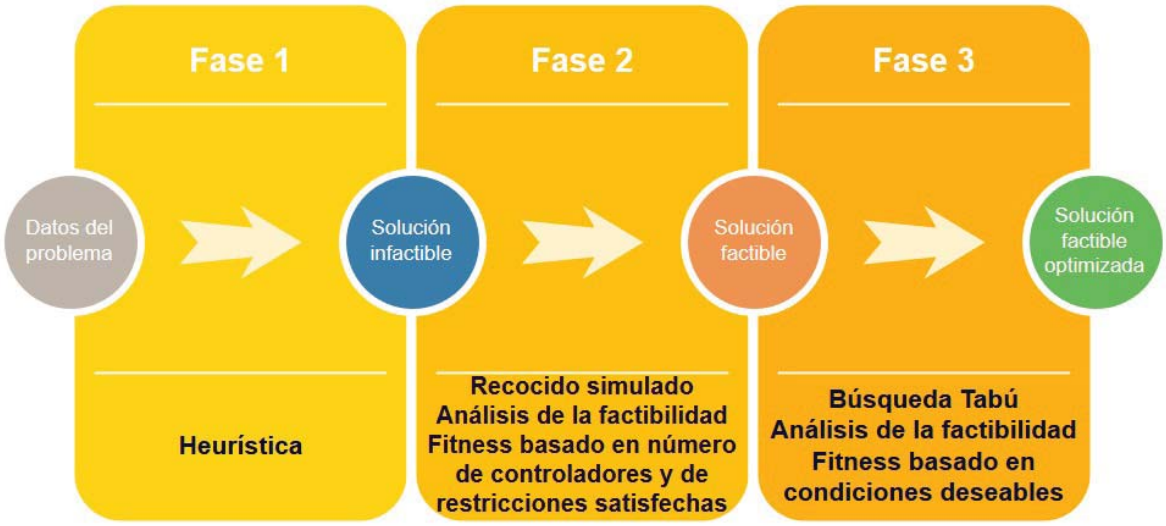

Figura 2.1: Diagrama que muestra las 3 fases llevadas a cabo

La metodología propuesta para resolver este problema comienza con una primera fase donde se propone una heurística para construir soluciones factibles/infactibles basado en el uso de una plantilla optimizada y la modificación de la longitud de los periodos de descanso.

Esta solución inicial puede incumplir condiciones de trabajo e incluso puede usar más controladores de los disponibles.

En la segunda fase, un algoritmo MIR (Multiple Independent Run) basado en recocido simulado es aplicado a las soluciones logradas en la primera fase con el objetivo de obtener soluciones factibles.

Para hacer esto, se aplica una función de fitness que intenta reducir el número de controladores usados hasta que llegue al número de controladores disponibles, a la vez que se

penaliza el número de veces que no se cumplen las condiciones laborales.

Es importante que estas dos fases puedan ser realizadas lo más rápido posible ya que es importante conocer tan pronto como sea posible si existe una solución factible con el número de controladores disponibles.

En la tercera fase, se aplica un algoritmo tabú que parte de las soluciones factibles obtenidas al finalizar la segunda fase.

El objetivo ahora es optimizar las condiciones deseables, teniendo en cuenta su importancia relativa ordinal proporcionada por los expertos de CRIDA. Este problema multiobjetivo, será transformado en un problema de optimización ponderada, donde los pesos son obtenidos por la función centroide, de esta manera se tiene en cuenta la información ordinal.

Durante la Fase 2 y 3 se usan expresiones regulares para comprobar la mayoría de las restricciones asociadas con las condiciones laborales. Esto nos aporta el beneficio de que son muy rápidas y además su modularidad nos permite realizar una implementación clara y mantenible del modelo de implementación.

2.2.1 Primera fase

En esta fase se propone una heurística para construir soluciones iniciales con distintos periodos de descanso.

Estas soluciones serán usadas después en la segunda fase para alcanzar soluciones factibles usando recocido simulado.

La heurística propuesta se basa en el uso de una plantilla optimizada como la que aparece en la Figura 2.2:

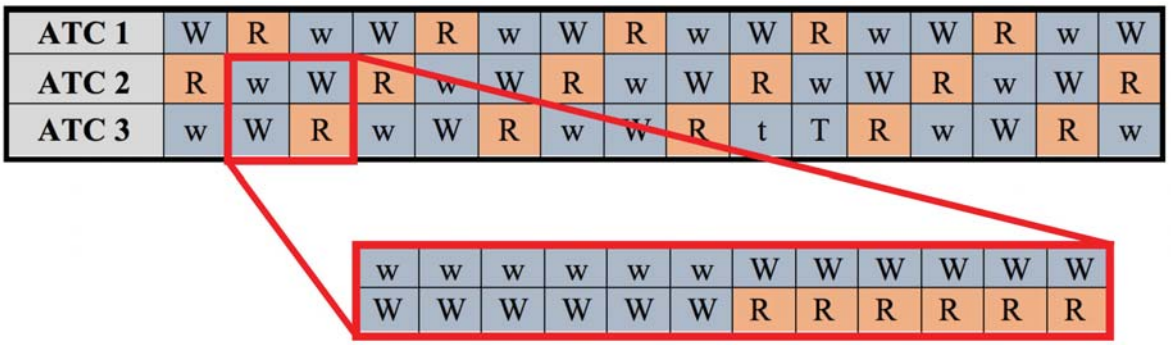

Figura 2.2: Plantilla

En la plantilla, la duración de los periodos de trabajo (w,W) es siempre la misma y es el doble de grande que los periodos de descanso (R). Además, si se usan distintas duraciones de cada periodo, la heurística construirá diferentes soluciones iniciales, aunque con una estructura similar.

Como las condiciones de trabajo, establecen que cada periodo de descanso debe durar 15

minutos (3 slots) y el máximo periodo de trabajo son 2 horas (24 slots), la duración de un periodo de descanso debe estar entre 3 y 12 slots. Por lo tanto, la heurística construye 10 soluciones iniciales.

La plantilla mostrada en la Figura 2.2 corresponde a un turno de 8 horas (96 slots) y con un periodo de descanso de 6 slots, donde R y W representan descanso y trabajo, respectivamente, y minúscula y mayúscula diferencian las posiciones de planificador y ejecutivo, respectivamente.

Denotemos con nCol el número de slots en el turno a optimizar, los pasos de la heurística son los siguientes:

- Paso 1. *Añadir plantillas para cubrir la sectorización aérea*. Por cada sector abierto $i\epsilon\{1, ..., s\}$:

  1. Insertar una plantilla vacía en la matriz de solución.
  2. Por cada slot $j\epsilon\{1, ..., nCol\}$:
     a) Si el sector *i* está abierto durante el slot de tiempo *j*, entonces poner el slot correspondiente a la plantilla en la posición *j* con el nombre del sector *j*.
     b) Si el sector *i* no está abierto durante el slot *j*, entonces introducir slots de descanso en la matriz solución en la posición *j*.

  Al finalizar el Paso 1, todos los sectores tienen asignados un par de controladores, para los que han sido introducidas las plantillas necesarias.

  Sin embargo, ciertas condiciones laborales pueden haber sido incumplidas.

  Denotemos con nRow el número de controladores considerados, en el Paso 2 se intenta mejorar la factibilidad.

- Paso 2. *Reparar la solución para mejorar la factibilidad*. Para cada fila $l\epsilon\{1, ..., nRows\}$, en la matriz solución que no cumple la longitud mínima de trabajo, intentamos asignar el periodo de trabajo cuya longitud no es mínima a otro controlador. Para hacer esto:

  1. Se intenta transferir un periodo de trabajo sin la longitud mínima de un controlador a otro:
     a) Se comprueba si hay otro controlador trabajando en el mismo sector que vaya a empezar a descansar cuando comienza el periodo de trabajo a tranferir (esto sería lo ideal porque el controlador seguiría trabajando en el mismo sector) y donde la longitud de ese periodo de descanso sea mayor que el periodo de trabajo que se intenta reasignar. Después, se comprueba si las siguientes restricciones son cumplidas antes de efectuar la transferencia:
        - Los controladores tienen que descansar 30 minutos por cada 2 horas de trabajo.
        - El máximo periodo de trabajo continuo es de 2 horas.

- El mínimo periodo de descanso continuo es de 15 minutos.
- El mínimo periodo de trabajo continuo es de 15 minutos.

b) Se comprueba si hay otro controlador que esté preparado para trabajar en ese mismo sector justo cuando el periodo de trabajo a ser transferido terminó y exista un periodo de descanso mayor que el fragmento que va a ser asignado. Al igual que en (a) también se comprueban las condiciones antes de efectuar la transferencia.

2. Se intenta ampliar un periodo de trabajo sin la mínima longitud usando un periodo de trabajo de otro controlador:
   a) Se comprueba si existe otro controlador trabajando en el mismo sector que haya empezado a la vez que el periodo de trabajo haya comenzado (este sería el caso ideal ya que el controlador estaría trabajando en el mismo sector) y que tenga suficientes slots para completar el tiempo de trabajo mínimo. Antes de hacer el cambio se comprueba que éste no implique un incremento en cuanto a la infactibilidad.
   b) Se comprueba si hay otro controlador que haya empezado a trabajar en el mismo sector justo después de que el periodo de trabajo que queramos expandir tenga suficientes slots para completar el mínimo tiempo de trabajo.
   c) Se comprueba la lista de restricciones que deben cumplir los controladores.

- Paso 3. *Asignación de recursos disponibles*. Por cada fila $l \epsilon \{1, ..., nRows\}$ en la matriz solución:
  1. Si la fila l contiene sectores de aproximación durante el turno largo, entonces buscamos un controlador PTD disponible asociado con el turno largo que pueda operar en ese núcleo. Este controlador es asignado a la fila l.
  2. Si la fila l contiene sectores de aproximación durante el turno corto, entonces se busca un controlador PTD disponible asociado con el turno corto que pueda operar en ese núcleo. Este controlador se asigna a la fila l. Si no, se busca un controlador PTD disponible asociado con el turno largo que pueda operar en ese núcleo y se asigna este controlador a la fila.
  3. Si la fila l contiene sectores de ruta durante el turno largo, entonces se busca un controlador CON disponible asociado con el turno largo que pueda operar en ese núcleo. Este controlador se asigna a la fila l. Si no, se busca un controlador PTD disponible asociado con el turno largo que pueda operar en ese núcleo y se asigna este controlador a la fila.
  4. Si la fila l contiene sectores de ruta durante el turno corto, entonces se busca un controlador CON disponible asociado con el turno corto que pueda operar en ese núcleo. Este controlador se asigna a la fila l. Si no, se busca un controlador CON disponible asociado con el turno largo, un controlador disponible asociado con el

turno corto, o un controlador disponible PTD asociado con el turno largo (en este orden) que pueda operar en ese núcleo y se asigna dicho controlador a esa fila.

En este punto, algunos controladores pueden no haber sido asignados ya que no cumplen los requisitos por fila. Estos controladores se asignan de la siguiente manera: Se comprueba por cada controlador $c\epsilon\{1, ..., nATC\}$ si está asignado a una fila en la matriz solución: Si el controlador *c* no ha sido asignado y es CON (PTD), entonces se busca una fila que incluya sectores en ruta (aproximación).

1. Si existe una fila disponible que incluya sectores en ruta (aproximación) entonces esta fila será asignada al controlador *c*.
2. Si no, una fila aleatoria será asignada al controlador *c*.

Una vez que todos los controladores han sido asignados puede que haya filas sin un controlador asignado. Si este es el caso, añadimos controladores artificiales, de esta manera la solución inicial podrá contener más controladores de los disponibles. Estos controladores artificiales serán eliminados en la siguiente fase mientras se buscan soluciones factibles.

Existe una excepción a esta heurística en los turnos nocturnos. Si existe un sector abierto durante todo el turno nocturno, entonces una plantilla que incluye 4 controladores con periodos de descanso de 9 slots (45 minutos) es usada para cubrir el sector. Estos controladores solamente trabajarán en ese sector. Por consiguiente, tendremos un problema de menor dimensionalidad (los sectores abiertos durante el turno nocturno ya están cubiertos) donde hay menos controladores disponibles (algunos de los originalmente disponibles ya han sido asignados a ese sector).

2.2.2 Segunda fase

El algoritmo MIR basado en recocido simulado comienza con las soluciones infactibles obtenidas de la primera fase con el objetivo de lograr soluciones factibles.

En primer lugar, vamos a describir la metaheurística del recocido simulado y como ésta se adapta al problema.

El recocido simulado (Cerny (1985), Kirkpatrick et al. (1983)) es una metaheurística basada en trayectorias que está nombrada e inspirada por el recocido en la metalurgia. La idea básica es la siguiente:

Se genera una solución factible x_0 aleatoriamente.

Después en cada iteración *i*, una nueva solución (y) se genera aleatoriamente en el vecindario $N(x_i)$, de la solución considerada en esa iteración x_i.

Si la nueva solución es mejor que la solución actual, entonces el algoritmo se mueve a esa solución. Si no, existe cierta probabilidad de moverse a una solución peor. El hecho de aceptar soluciones peores permite ampliar la búsqueda de una solución óptima y evita caer

en óptimos locales en iteraciones tempranas.

La búsqueda inicialmente está muy diversificada ya que prácticamente todos los movimientos están permitidos. A medida que la temperatura baja, la probabilidad de aceptar un movimiento peor decrece y solo movimientos mejores serán aceptados cuando es 0. Esto hace que que el recocido simulado funcione como *hill climbing*.

Un pseudocódigo tipo para un problema de minimización es el siguiente:

Inicialización: $x^* = x_0, f^* = f(x_0), i = 0$. Se selecciona como temperatura inicial $t_0 = T$(t_i es la temperatura en el paso *i*).

Se repite hasta que el criterio de parada se satisfaga:

Generar aleatoriamente $y_i \epsilon N(x_i)$

Si $f(y_i) - f(x_i) \leq 0$, entonces

$x_{i+1} = y_i$

Si $f(x^*) > f(y_i)$, entonces $x^* = y_i, f^* = f(y_i)$

Si no

$p \sim U(0,1)$

Si $p \leq e^{-(f(y_i)-f(x_i))/t_i}$, entonces $x_{i+1} = y_i$

Si no $x_{i+1} = x_i$

Actualizar temperatura, $i = i + 1$

Generalmente, la temperatura inicial se fija de manera que el ratio de aceptación de movimientos peores sea igual a cierto valor especificado (Ben-Ameur, 2004).

La programación del enfriamiento define la manera en que la temperatura disminuye tras las iteraciones. Una programación típica de la temperatura es aquella en la que esta se mantiene constante durante un número de iteraciones (L) y luego se decrementa de acuerdo a alguna fórmula: $T_k = \alpha^k T_0$, siendo el valor típico para α 0.95 (Hajek, 1988).

El criterio de parada más común consiste en detenerse cuando la mejora en la función fitness es menor que un porcentaje dado durante un número fijo de iteraciones.

El algoritmo de recocido simulado se adapta para tratar la búsqueda de soluciones factibles de esta manera: las soluciones iniciales que recibe este algoritmo, son las soluciones infactibles de la primera parte. Sin embargo, éstas se reorganizan antes de comenzar la búsqueda. El controlador con la menor carga de trabajo se sitúa en la primera fila de la solución codificada, el segundo con la carga más pequeña se situará en la segunda fila y así, hasta que los controladores con la mayor carga se sitúen en las últimas filas, es decir, la matriz solución es cambiada a través de permutaciones.

En relación a los movimientos que se realizan para moverse a una nueva solución, el

proceso llevado a cabo está detallado en la Sección 2.2.3.

Este proceso de permutación también es realizado en cada una de las soluciones visitadas en el proceso de búsqueda, ya que tiene su influencia en el fitness, y viene desarrollado en la Sección 2.2.4.

El objetivo de la función fitness en esta fase es reducir el número de controladores usados hasta que se alcance el número de controladores disponibles a la vez que se penalizan las condiciones de trabajo incumplidas, el detalle de como se calcula la función fitness se desarrolla también en la Sección 2.2.4.

2.2.3 Movimientos del entorno

Dada una posible solución, la siguiente solución (o siguiente vecino) será obtenido realizando un movimiento que llamaremos movimiento de entorno.

El movimiento de entorno que usaremos, al que llamamos *movimientoIntervalosPseudo-Aleatorios* consta de los siguientes pasos:

1. Se escoge un controlador que cederá un periodo de trabajo a otro controlador.
2. Obtenemos los periodos de trabajo continuado de este primer controlador y escogemos uno aleatoriamente. Ahora toca elegir la longitud de este periodo, para ello elegimos aleatoriamente entre 3, 6, 9 o 12 slots de trabajo. Si la longitud del periodo es menor que la elegida, esta no varía. Si la longitud del periodo es mayor que la elegida, se eligen únicamente los primeros 3, 6, 9 o 12 slots del periodo de trabajo.
3. Por último se elige un segundo controlador aleatoriamente, que acepte la carga de trabajo del otro controlador, si es posible. Esto habría que comprobarlo.

Además de estos pasos hay ciertas circunstancias especiales que hay que tener en cuenta.

- Si se ha conseguido obtener un controlador que puede asumir la carga de trabajo extra (para ello tendría que estar en posición de descanso en el periodo de trabajo que vaya a asumir) sin que se haya roto la factibilidad, el movimiento habrá finalizado y tendremos una nueva solución factible.
- Si no puede asumir la carga de trabajo por estar trabajando previamente o porque se genere una solución infactible, se buscará aleatoriamente entre el resto de controladores hasta encontrar uno que pueda asumir la carga manteniendo la factibilidad.Una vez encontrado, el movimiento finaliza.
- Si aún no se ha podido encontrar ni un controlador que asuma la carga de trabajo que se va a intercambiar se cambia la longitud de ésta. Para ello, se vuelve a elegir aleatoriamente entre 3, 6, 9 y 12, excluyendo el valor elegido con anterioridad, para evitar que vuelva a ocurrir lo mismo. Volvemos al punto anterior y repetimos, si hay suerte y se puede completar el movimiento, finalizamos. Si no hubo suerte, volvemos a elegir

el tamaño, excluyendo los elegidos anteriormente. Si una vez que hemos probado los 4 posibles tamaños no logramos alcanzar una solución factible, modificamos el primer controlador y el periodo de trabajo.

- Para esto, elegimos otro controlador (evitando repetir) y elegimos otro periodo de trabajo como se hizo en el Paso 1.

Además de este movimiento, existen otros dos que son una pequeña variación del anterior:

- En el movimiento denominado *movimientoIntervalosAleatorios*, la única diferencia aparece a la hora de elegir los intervalos de trabajo. Una vez que se tenía el intervalo de trabajo, se podía reducir el tamaño a 3, 6, 9 o 12. Con este movimiento el tamaño se escoge aleatoriamente entre 2 y 12.
- En el movimiento denominado *movimientoAleatorio*, la diferencia es que se empieza con una longitud de intervalo de 12 y si no se logra alcanzar una solución factible, se va reduciendo de 1 en 1 el número de slots del intervalo, hasta llegar a un tamaño mínimo de 2.

Estos diferentes tipos de movimiento se pueden alternar para usar el que más nos interese.

2.2.4 Fitness de la segunda fase

En la segunda fase, es decir, en la de búsqueda de una solución factible, se usará una función fitness que consta de dos términos, los cuales serán normalizados para ser combinados en un modelo aditivo ponderado. En el primero se intenta reducir el número de controladores en la solución, en el segundo se intenta reducir la infactibilidad.

- Primer término: La reducción del número de controladores se va a intentar conseguir tratando de maximizar la siguiente fórmula:

$$p = \sum_{k=0}^{nATC-1} \left(\frac{(k+1) * h_k}{nATC^2} \right)$$

siendo, nATC el número de controladores y h_k el número de slots de trabajo del controlador *k* (es decir, en la fila *k*-ésima de la codificación de la solución).

Hay que tener en cuenta que en la solución que se va analizando, los controladores van ordenados de menor a mayor carga de trabajo.

En la fórmula anterior, en el numerador, el producto hace que el valor de *p* sea mayor a mayor carga de trabajo que tengan los controladores de las últimas filas. Así, en el proceso de búsqueda se tiende a pasar carga de trabajo de las primeras filas a las

últimas para así poder tener los controladores de las primeras filas sin carga y poder eliminarlos.

El denominador está elevado al cuadrado para que merezca la pena eliminar un controlador.

A continuación se normaliza:

$$f_1 = 1 - ((p_{max} - p) \ / (p_{max} - p_{min})),$$

siendo p_{max} y p_{min} el valor máximo y mínimo de *p* con el objetivo de normalizar la función.

p_{max} y p_{min} hacen uso de un término denominado $nATC_{completos}$, para calcular éste, se calcula la carga de trabajo total a partir de la sectorización. Lo cual nos proporciona un número de slots de trabajo que se deben repartir entre todos los controladores, al que denominamos C_{total}.

Para obtener $nATC_{completos}$ se usa la siguiente fórmula:

$$nATC_{completos} = \frac{C_{total}}{nSlot}.$$

Es decir, tenemos el número de controladores que tienen que trabajar todo el turno sin descanso para poder cubrir la carga de trabajo (C_{total}). Una vez que conocemos este valor podemos calcular p_{max}:

$$p_{max} = \sum_{k=nATC-(nATC_{completos}+1)}^{nATC} \left(\frac{k \times nSlot}{nATC^2}\right) ,$$

donde se asigna la máxima carga de trabajo a los controladores que se encuentran en las últimas filas, mientras que a las primeras no se les asigna la carga de trabajo.

El valor de p_{min} se calcula con la siguiente fórmula:

$$p_{min} = \sum_{k=1}^{nATC_{completos}-1} \left(\frac{k \times nSlot}{nATC^2}\right) ,$$

donde en este caso, no asignamos carga de trabajo a los controladores de las últimas filas pero sí se les asigna la máxima carga de trabajo a los de las primeras.

- Segundo término: Para medir la reducción de la infactibilidad, se contabilizan el número de condiciones de entorno que se están violando, el cual llamaremos R_i y queremos minimizar.

Para pasarlo a forma de maximización usaremos la siguiente fórmula:

$$f_2 = \ (MaxR_i - R_i) \ / \ MaxR_i \ ,$$

siendo $MaxR_i$ el valor máximo que se puede alcanzar incumpliendo todas las restricciones.

A continuación hablamos de que existen 14 restricciones, cuando en la introducción habíamos mencionado 18 (las proporcionadas por CRIDA), esto sucede porque debido a los movimientos del entorno que se realizan, estas restricciones pueden formularse en otras 14, las cuales vienen detalladas en la Sección 2.2.5.

Hay que tener en cuenta dos casos:

- Turno de noche: $MaxR_i$ se calcula de la siguiente forma. Hay 8 restricciones que pueden incumplirse nATC veces (R3, R5, R6, R9, R10, R11, R12, R14). 2 restricciones (R4 y R13) pueden incumplirse 2 x nATC veces. Y hay 4 restricciones (R1, R2, R7 y R8) que se pueden incumplir nATC, sólo que esta vez, se considera el grado de incumplimiento y se multiplica cada vez que no se cumple en cada slot del turno por 0.01. Por lo tanto la fórmula resultante sería la siguiente:

  $$MaxR_i = (8 \ \times \ nATC) + (2 \ \times \ 2nATC) + \\ (4 \ \times \ nATC + nSlot \ \times \ nATC \ \times \ 0{,}01) \rightarrow \\ MaxR_i = 16 \ \times \ nATC + \ 4 \ \times (nATC \ \times \ nSlot \ \times \ 0{,}01) \, .$$

- Turnos diurnos (mañana y tarde): En este caso la restricción R4 no se aplica así que la fórmula resultante es ésta:

  $$MaxR_i = (8 \ \times \ nATC) + (1 \ \times \ 2nATC) + \\ (4 \ \times \ nATC + nSlot \ \times \ nATC \ \times \ 0{,}01) \rightarrow \\ MaxR_i = 14 \ \times \ nATC + \ 4 \ \times (nATC \ \times \ nSlot \ \times \ 0{,}01) \, .$$

La función fitness final queda así:

$$f = \begin{cases} f_1 * \mu_1 \ + \ f_2 * \mu_2 & si \ nATC > nATCdisp \\ f_2 & si \ nATC \leq nATCdisp \end{cases}$$

donde μ_1 y μ_2 son los pesos que expresan la importancia relativa de los términos. Por defecto se han puesto a 0.2 y 0.8.

Cuando el número de controladores disponibles (nATCdisp) sea menor que el número de controladores usados en la solución se usarán los dos términos, si no, solo se usa f_2 para buscar una solución con ese número de controladores, pero que sea factible.

2.2.5 Análisis de la factibilidad

De las restricciones proporcionadas por CRIDA, debido a como se realizan los movimientos del entorno, éstas, se pueden reformular en 14, las cuales se explican en este apartado.

Para comprobarlas, en algunas de ellas usaremos expresiones regulares.

Una expresión regular o regexp es una cadena de texto especial que se usa para describir un patrón de búsqueda (Friedl (1997)). Estas se componen de metacaracteres (),[] y ; clases de caracteres como:

\A(comienzo de una cadena).

\s (espacio en blanco).

\d (dígito).

Y cuantificadores * (0 o más), + (1 o más) y ? (0 o 1).

Su principal aplicación en programación es automatizar la búsqueda y los procesos de análisis. Existen muchas herramientas donde se usan expresiones regulares, en este caso como usamos Java como lenguaje de programación usaremos RegEx que pertenece a java.util.regex.

Los beneficios de usar expresiones regulares es su rápida velocidad de verificación y su modularidad, lo que permite realizar una clara y mantenible implementación del modelo de optimizacion.

Entre sus aplicaciones figuran la identificación de patrones en secuencias de ADN, la reconstrucción de hebras de ADN o el reconocimiento de patrones.

Las 14 condiciones que verificamos así como las expresiones regulares que usamos para comprobarlas (cuando se pueda) son las siguientes:

1. *Una determinada posición podrá ser asignada a un controlador si el controlador está habilitado en el núcleo al que pertenece el sector que le corresponde, o bien en uno de los núcleos a los que pertenece el sector, si este es un sector común, independientemente de la sectorización por la que el sector se encuentre abierto.*

   Esta restricción no la comprobamos con expresiones regulares. Comprobamos que cada fila asignada a un controlador con una acreditación para un núcleo solo tenga sectores pertenecientes a ese núcleo. Si en una fila encontramos algún sector no perteneciente al núcleo suma 1 al valor de penalización. También se suma 0.01 por cada vez que se encuentre en una celda de la fila un sector que no pertenezca al núcleo.

2. *A un controlador tipo CON podrá asignársele una posición cuyo sector sea Ruta.*

   Esta restricción no la comprobamos con expresiones regulares. Se comprueba que cada fila asignada a un controlador CON sólo contenga sectores Ruta. Si encontramos algún sector que no sea Ruta sumamos 1 a la penalización, más 0.01 cada vez que encontremos una celda de la fila que contenga algún sector que no sea tipo Ruta.

3. *Porcentaje de tiempo de descanso mínimo en turno diurno (mañana y tarde), incluyendo turnos largos es de 25 %. Mientras que el porcentaje de tiempo de descanso mínimo en el turno de noche es de 33 %.*

   Esta restricción se comprueba tanto con expresiones regulares como con código. Contamos cuantas celdas hay de descanso por cada fila mediante la expresión regular: Pattern.compile("(111)") Si estamos en un turno diurno, si la fila no tiene un 25 % de descanso sobre el total del turno, la restricción es incumplida y sumamos 1 a la penalización, si el turno es nocturno, el porcentaje de descanso se eleva al 33 %.

4. *Los sectores o agrupaciones de dos sectores que se indique en la entrada del problema, se deberán cubir con 4 controladores en el turno de noche. Nota: puede existir más un sector o agrupación de sectores que se deban cubrir con 4 controladores. Esta restricción solo se aplica a los turnos de noche.*

   Esta restricción se comprueba con expresiones regulares a la vez que con código. Para ello se comprueba con expresiones regulares que los controladores que tienen asignado algún «sector nocturno» (Denominación de las agrupaciones de sectores que deben cubrirse por 4 controladores), sólo trabajen en sectores nocturnos de la misma agrupación. La regexp usada es la siguiente:

   ```
   Pattern.compile("^("+ sctNc +"111){"+numSlots+"}\$", Pattern.CASE_INSENSITIVE)
   ```

   donde sctNc es una cadena de texto que contiene la lista de sectores nocturnos de la misma agrupación. Por ejemplo aaa|abb|aad. Si esta condición no se cumple se suma 1 al valor de penalización. Hecho esto, en código comprobamos que por cada agrupación de sectores nocturnos haya 4 controladores que cubran estos sectores. Si no se cumple, se suma 1 a la penalización.

5. *En los turnos de mañana, tarde y noche, no es posible un periodo de trabajo continuo mayor de 2 horas en los que el controlador no realice ningún periodo de descanso.*

   Se comprueba con expresiones regulares. Utilizando 12 regexp con esta estructura:

   ```
   Pattern.compile("\^(111)*[a-zA-Z]{3,"+tMax+"}
   (111)+[a-zA-Z]{3,"+tMax+"} (111)*\$")
   ```

Si cumple una sola de las 12, cumple la restricción. El valor de tMax es 72 (3*120/5=72). Aplicamos el conjunto de regexp a cada línea de la matriz y sumamos 1 a la penalización por cada línea que no lo cumpla.

6. *Un controlador solo puede operar en su turno correspondiente, si pertenece al turno largo, en el turno largo y si pertenece al turno corto en el turno corto*

   Esta restricción se comprueba con expresiones regulares. El turno largo es el total del turno, por lo que no es posible trabajar fuera del mismo y no hace falta comprobarlo. En el turno corto se usará una de las siguientes expresiones regulares:

   ```
   Pattern.compile("\^.*(111){"+resto+"}\$")
   ```

   ```
   Pattern.compile("\^(111){"+resto+"}.*\$")
   ```

   El valor de resto es la diferencia entre el turno largo y corto, es decir, lo que los controladores pertenecientes al turno corto tienen que descansar. Si el turno corto empieza después del largo usamos la primera, si el turno corto finaliza antes que el largo se usa la segunda. Por cada fila con un controlador de turno corto y que no cumpla la restricción se suma 1 a la penalización.

7. *En los turnos de mañana, tarde y noche, no puede existir ningún periodo de dos horas y media en los que un controlador realice un periodo total de descanso menor de media hora. Es decir, dentro de una ventana de tiempo de dos horas y media un controlador debe tener mínimo 30 minutos de descanso, sin ser necesario que estos se realicen de forma continua.*

   En este caso no se usan expresiones regulares. Se comprueba cada fila para que en una ventana de 2 horas y 30 minutos solo pueda trabajar durante dos horas y tenga que descansar media obligatoriamente. Si una fila no cumple la restricción se sumará 1 al valor de penalización, además se sumará 0.01 por cada intervalo de 2:30 horas que no cumpla la restricción.

8. *Un controlador no puede cambiar de posición de control de una posición ejecutiva de un determinado sector a una posición ejecutiva de otro sector diferente, sin que exista un descanso entre medias, a no ser que ambos sectores sean afines (cambio de configuración). Nota: si no hay cambio de configuración de sectores no es posible que dos sectores diferentes posean volúmenes comunes.*

   En este caso no se usan expresiones regulares. Para comprobar esta restricción se utiliza la matriz de afinidad, que es una matriz en la que están todos los sectores asociados entre ellos y nos da la información cuando los sectores son afines. Por lo tanto, cuando un controlador pasa en posición ejecutiva a otro sector sin descanso se debe comprobar si estos son afines. Si en la fila se encuentran dos sectores no afines en posición

ejecutiva consecutivos sumamos 1 al valor de penalización, además sumaremos 0.01 por cada vez que esto suceda en esa misma fila.

9. *El tiempo mínimo de trabajo continuado. Es decir, el tiempo de trabajo de un controlador entre dos descansos es de 15 minutos.*

   En este caso usamos expresiones regulares, 12 con la siguiente estructura

```
Pattern.compile("\^(111)*[a-zA-Z]{"+tMin+",}(111)+[a-zA-Z]{"+tMin+",}(111)*\$")
```

   Solo es necesario que cumpla 1 de las 12 utilizadas, con eso ya sabremos que cumple la restricción. El valor de tMin es de 9 (3 * 15/5 = 9). Este conjunto de expresiones lo aplicamos a cada línea de la matriz de turnos y sumaremos 1 al valor de penalización por cada línea que no lo cumpla.

10. *El tiempo mínimo de descanso de un controlador es de 15 minutos.*

    Usaremos 12 expresiones regulares con la siguiente estructura:

```
Pattern.compile ("\^(111)*[a-zA-Z]{3,}(111){"+dMin+",}[a-zA-Z]{3,}((111)*\$)")
```

    Solo es necesario que cumpla 1 de las 12 utilizadas, con eso ya sabremos que cumple la restricción. El valor de dMin es de 9 (3 * 15/5 = 9). Este conjunto de expresiones lo aplicamos a cada línea de la matriz de turnos y sumaremos 1 al valor de penalización por cada línea que no lo cumpla.

11. *El tiempo mínimo en posición de un controlador es de 15 minutos.*

    Lo comprobaremos con 12 expresiones regulares con la siguiente estructura:

```
Pattern.compile ("\^(111)*(([A-Z]{"+pMin+",}|([A-Z]{"+pMin+",}[a-z]
 {"+pMin+",})|([A-Z]{"+pMin+",}[a-z]{"+pMin+",}[A-Z]{"+pMin+",})|([A-Z]
 {"+pMin+",}[a-z]{"+pMin+",}[A-Z]{"+pMin+",}[a-z]{"+pMin+",}))|(([a-z]
 {"+pMin+",})|([a-z]{"+pMin+",}[A-Z]{"+pMin+",})|([a-z]{"+pMin +",}[A-Z]
 {"+pMin+",}[a-z]{"+pMin+",})|([a-z]{"+pMin+",}[A-Z]
 {"+pMin +",}[a-z] {"+pMin+",}[A-Z]{"+pMin+",}))))(111)*\$")
```

    Solo es necesario que cumpla 1 de las 12 utilizadas, con eso ya sabremos que cumple la restricción. El valor de pMin es de 9 (3 * 15/5 = 9). Este conjunto de expresiones lo aplicamos a cada línea de la matriz de turnos y sumaremos 1 al valor de penalización por cada línea que no lo cumpla.

12. *Número máximo de sectores por los que rota un controlador, teniendo en cuenta las posibles afinidades: 3*

    Comprobamos que los controladores no pasen por más de 3 sectores teniendo en cuenta cuando comparten volúmenes. Si un controlador pasa por más de 3 sectores penalizamos con 1 cada vez que esto no se cumpla.

13. *Todos los controladores disponibles deben tener una línea de la matriz asignada y a su vez todas las líneas de la matriz deben estar asociadas a un controlador.*

    Comprobamos que cada línea tenga un controlador asociado, si alguna no lo tiene penalizamos con 1.

    También debemos realizar la comprobación al revés, es decir, que cada controlador esté asignado a una línea de la matriz de turnos. En el caso de que no lo esté penalizamos con 1.

14. *Todos los controladores deberán tener al menos 3 slots de trabajo, no podrán permanecer descansando toda la jornada.*

    En este caso usaremos expresiones regulares. Usaremos el patrón siguiente:

    ```
    Pattern.compile("\^(111){"+nSlot+"}\$")
    ```

    nSlot es el número total de slots del turno. El patrón comprueba si la línea entera esta formada por 111, indicando ésto, que en ningún momento trabaja, por lo tanto, si cumple el patrón incumple la restricción. Entonces sumamos 1 al valor de penalización cada vez que una línea incumpla la restricción.

2.2.6 Tercera fase

En la tercera fase usaremos búsqueda tabú para optimizar las soluciones factibles procedentes de la Fase 2. El objetivo ahora consiste en optimizar cuatro funciones objetivo teniendo en cuenta la información ordinal acerca de su importancia relativa que nos ha sido dada por CRIDA. Los movimientos del entorno siguen siendo los mismos que en la Fase 2.

A continuación vamos a describir la búsqueda tabú y como la hemos adaptado a nuestro problema.

La búsqueda tabú (Glover and Melián, 2003) es una metaheurística que usa búsqueda local con el propósito de optimizar una solución.

Usa una búsqueda local para moverse de una solución potencial x a una solución mejor x' en el vecindario de x hasta que un criterio de parada sea satisfecho.

El problema de esta búsqueda local es que puede quedarse atascado en zonas donde hay soluciones malas, mientras que en otras zonas puede haber soluciones mejores y que queden sin explorar. Es decir, corre el riesgo de caer en mínimos locales.

Para evitar esto, las soluciones admitidas para el nuevo vecindario $N(x)$ vienen determinadas por el uso de estructuras con memoria.

Estas estructuras es lo que se conoce como lista tabú. Que consisten en un conjunto de reglas y soluciones prohibidas que deciden si una solución puede ser admitida en el vecindario de $N(x)$.

En nuestro caso usaremos dos tipos de memoria, memoria a corto plazo y memoria a largo plazo. Y lo que almacenarán será el movimiento, por ejemplo, en estas se almacenará que el Controlador1 le intercambió los slots de trabajo $[15 - 18)$ al Controlador3.

La memoria a corto plazo (intensificación) consiste en prohibir un movimiento durante un cierto número de iteraciones, esto se usa para evitar caer en ciclos.

La memoria a largo plazo (diversifiación) consiste en penalizar los movimientos más comunes, con la intención de moverse a otras zonas y escapar de mínimos locales.

La búsqueda tabú en cada iteración va a elegir una solución que incluso puede que sea peor que la solución actual.

Además de esto, hemos implementado un criterio de aspiración que consiste en que si una solución es la mejor vista hasta ahora, podemos seleccionarla aunque esté tabú.

Debido a la gran dimensionalidad del problema, nos es imposible calcular el fitness de todos los vecinos de una solución, por eso debemos elegir una cifra razonable de ellos, 100, por ejemplo.

La alternancia entre la diversificación y la intensificación es otro de los parámetros que podemos ajustar, junto con los valores de las penalizaciones de éstos.

Por último, como criterio de parada, podemos hacerlo después de un cierto número de iteraciones, después de un límite de tiempo o después de superar un cierto valor del fitness. Además, existe la posibilidad de realizar un cierto número de iteraciones extra, para ver si la solución va a seguir evolucionando.

Las funciones objetivo que se van a optimizar se detallan a continuación.

En primer lugar, los periodos de trabajo y descanso y las posiciones de los controladores deben estar cercanos a unos valores fijos. Concretamente, el tiempo que un controlador debe permanecer en el mismo sector y posición de trabajo debe acercarse a 45 minutos, el tiempo de trabajo entre dos periodos de descanso debe acercarse a 90 minutos y el porcentaje de tiempo que los controladores trabajan en posiciones ejecutivas debe estar entre el 40 y el 60 % del tiempo total de trabajo (excluyendo descansos).

En segundo lugar, la estructura de la solución debe ser similar a unos estadillos proporcionados por CRIDA. Con esto se pretende que la solución sea fácilmente entendible por el personal del centro de control y facilitaría el poder hacer cambios a mano.

En tercer lugar, el número de cambios de sala del centro de control debe ser mínimo. Para esto el número de periodos de descanso debe ser mínimo.

Por último, la distribución de la carga de trabajo entre los distintos controladores debe estar balanceada.

Para medir el grado de cumplimiento de estas 4 condiciones deseables, usaremos una medida de fitness que viene detallada en la Sección 2.2.7.

2.2.7 Modelización de las condiciones deseables

Las condiciones deseables que nos solicita CRIDA son las que se van a tener en cuenta para calcular el fitness durante la tercera fase. Constan de 4 objetivos, a los que se les asignan pesos mediante el método del centroide para obtener una medida de fitness global.

En este caso, como queremos medir el grado de cumplimiento de cada una de ellas, no podemos usar expresiones regulares.

La modelización de cada una de estas condiciones y el ajuste de los pesos de estas 4 para tener un solo objetivo global a optimizar se detalla a continuación:

Objetivo 1: Condiciones de elección de solución

El primer objetivo está formado por 3 condiciones deseables:

1. Primera condición deseable: Tiempo óptimo de trabajo en posición. Se corresponde con el tiempo en el que un controlador se encuentra sin cambio de sector y tipo de control. El tiempo óptimo es de 45 minutos.

   Para comprobar si esta condición se cumple y poder medir su grado de incumplimiento realizamos lo siguiente. Por cada fila de la matriz se calcula el número de minutos en los que el controlador está en el mismo sector cubriendo la misma posición de forma continuada, y anotamos la diferencia entre este valor y el óptimo.

   A continuación, sumamos todas las diferencias de todos los intervalos. Como esto se realiza por filas, sumamos el total de la diferencia de todas las filas y luego dividimos entre el número de controladores (número de filas) para obtener la media. La fórmula es la siguiente:

   $$v1 = \left(\sum_{k=0}^{nATC} \sum_{i=0}^{nIntervalos} |posOpt - l_i| \right) / nATC,$$

   donde posOpt es el tiempo óptimo de trabajo (45 minutos) y l_i la longitud en minutos del intervalo *i*. Para normalizar este valor, lo realizaremos con la siguiente fórmula:

   $$vn1 = ((v1_{max} - v1) / v1_{max}),$$

donde $v1_{max}$ es el valor máximo de v1 calculado mediante la siguiente expresión:

$$v1_{max} = |posOpt - posMin| \ \times \ 8 \ \times \left(\frac{nSlot}{30}\right),$$

donde posMin es el tiempo mínimo que debe permanecer en una posición un controlador para no incumplir ninguna restricción. Al dividir entre 30 slots, se calcula el número de veces que se puede dar un periodo de trabajo de dos horas (24 slots) continuado de un periodo de 30 minutos de descanso (6 slots). Se multiplica por 8 ya que en este periodo se pueden encontrar 8 intervalos de trabajo de distintas posiciones sin incumplir ninguna restricción.

2. Segunda condición deseable: Tiempo óptimo de trabajo entre descansos (90 minutos).

   Para comprobar si se cumple y medir su grado de incumplimiento realizamos lo siguiente. Por cada fila calculamos la longitud de cada intervalo de trabajo (en minutos) y anotamos la diferencia entre ésta y el óptimo de trabajo (trabOpt) y sumamos al final las diferencias de todos los intervalos. Como realizamos esto por filas, las sumamos todas y realizamos la media. La formula quedaría tal que así:

   $$v2 = \left(\sum_{k=0}^{nATC} \sum_{i=0}^{nIntervalos} |trabOpt - l_i| \right) / nATC,$$

   y para normalizar este valor, el valor máximo de estas diferencias lo obtenemos así:

   $$v2_{max} = |trabOpt - trabMin| \ \times \ \left(\frac{nSlot}{6}\right),$$

   donde trabMin es el trabajo mínimo posible sin incumplir ninguna restrricción (15 minutos). En este caso se divide entre 6 el número de slots porque el trabajo mínimo son 3 slots y el descanso mínimo son 3 también. Por lo tanto esta restricción se podría incumplir por cada 6. Para normalizar usamos la siguiente expresión:

   $$vn2 = \ ((v2_{max} - \ v2) \ / v2_{max}).$$

3. Tercera condición deseable: El porcentaje de tiempo que un controlador trabaja en posiciones ejecutivas debe estar comprendido entre el 40 y el 60 % del trabajo total realizado (sin contar descansos).

   Para comprobar si se cumple y medir su grado de incumplimiento, realizamos lo siguiente. Por cada fila *k* de la matriz, dividimos el tiempo que se encuentra un contro-

lador trabajando en la posición de ejecutivo entre el tiempo de trabajo total. De esta manera obtenemos el porcentaje de trabajo en posición ejecutiva en esa fila $pEje_k$. Si este es menor que 40 % o mayor que 60 % realizamos la diferencia al valor más próximo y dividimos por la diferencia máxima, es decir por 0.4 para normalizar este valor. Después sumamos las diferencias entre todas las filas y dividimos entre el número de filas para mantener normalizado el valor. La fórmula queda expresada de la siguiente manera:

$$vn3 = \left(\frac{1}{0{,}4}\right) \times \sum_{k=0}^{nATC} \begin{cases} \frac{|pEje_k - 0{,}4|}{nATC}, & si\ pEje_k < 0{,}4 \\ \frac{|pEje_k - 0{,}6|}{nATC}, & si\ pEje_k > 0{,}6 \end{cases} .$$

Una vez que tenemos los valores de cada una de las condiciones deseables, las ponderamos equitativamente y obtenemos un fitness normalizado entre 0 y 1 usando la siguiente fórmula:

$$f_1 = vn1 \ \times \ \mu_1 + \ vn2 \ \times \ \mu_2 + \ vn3 \ \times \ \mu_3,$$

donde los pesos μ_1,μ_2,μ_3 tienen el valor de 0.33.

Objetivo 2: Mantener una estructura similar a los estadillos

El segundo objetivo consiste en mantener una estructura similar a los estadillos/plantillas que se nos proporcionan. Un ejemplo de un estadillo aparece en la Figura 2.3.

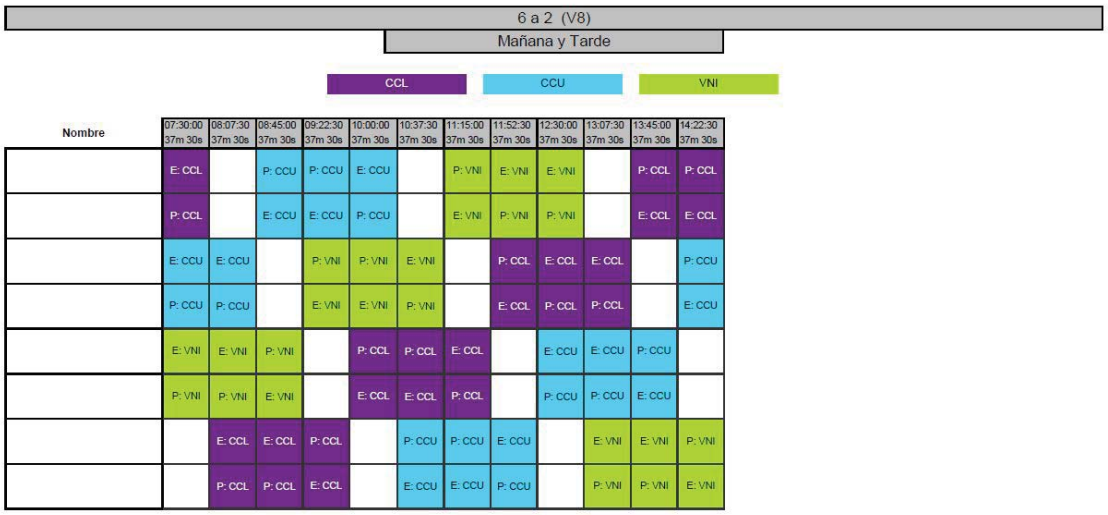

Figura 2.3: Estadillos/plantillas

Básicamente el objetivo de esta condición consiste en que el resultado final pueda ser comprendido y entendido fácilmente con un simple vistazo.

A falta de una medida proporcionada por un experto, se entiende que cuanto más agrupados estén los sectores iguales así como los descansos, mejor será la medida/solución. Para cumplir este objetivo empleamos la siguiente fórmula:

$$v = \sum_{k=1}^{nATC-1} \sum_{j=1}^{nSlot-1} \begin{cases} 1, & si \;\; slot_{kj} = \; slot_{k+1j} \\ 1, & si \;\; slot_{kj} = \; slot_{kj+1} \end{cases}.$$

En esta función lo que se hace es, por cada $slot_{kj}$, si el slot de su derecha ($slot_{kj+1}$) tiene el mismo contenido y si es así sumamos 1, también realizamos lo mismo con el slot de debajo $slot_{k+1j}$), es decir si este tiene el mismo contenido sumamos también 1. El máximo valor teórico viene dado por la siguiente expresión:

$$v_{max} = (nSlot - 1) \times (nATC - 1) \times 2.$$

Este es un valor máximo teórico, esto quiere decir que ha sido obtenido sin tener en cuenta las restricciones. Una vez tenemos este valor ya podemos normalizar el objetivo con la siguiente expresión:

$$f_2 = 1 - \; ((v_{max} - \; v) \; / v_{max}).$$

Objetivo 3: Minimizar el número de cambios de sala

El tercer objetivo está compuesto de dos subobjetivos, sin embargo uno de ellos (Maximizar el número de sectores que influyen en las acreditaciones) no nos ha sido proporcionado por CRIDA por lo que ha sido descartado de la solución final. En cuanto al otro subobjetivo, este consiste en minimizar el número de intervalos de descanso. Para calcular el fitness, por cada fila contamos el número de intervalos de descanso. El valor mínimo de la función es el número de filas $v_{min} = nATC$, ya que en todas debe haber un descanso como mínimo. El valor máximo viene dado por la siguiente fórmula:

$$v_{max} = \left(\tfrac{nSlot}{6}\right) \times \; nATC,$$

en este caso, dividimos entre 6 porque es la suma del mínimo de slots entre los intervalos de descanso y trabajo mínimos, de esta manera obtenemos el número máximo de intervalos por cada fila de la matriz. Una vez tenemos los valores mínimos y máximos normalizamos con la siguiente función:

$$f_{3,1} = (v_{max} - \ v) \ / \ (v_{max} - v_{min}) \ .$$

Objetivo 4: Distribución homogénea de la carga de trabajo

El cuarto objetivo busca que haya una distribución de carga de trabajo equilibrada entre los controladores. Para medirlo, usaremos la desviación estándar sobre los minutos de trabajo.

El valor máximo de la desviación estándar, obviando las restricciones del problema, es igual a la media. Así pues con una carga de trabajo de 200 slots y 4 controladores que trabajan como máximo 100 slots la media es de 50 slots de trabajo. Para maximizar la varianza se reparte 100 slots a 2 controladores y 0 a los otros 2. Esto nos da una varianza de 50^2 y al realizar la desviación típica un valor de 50, es decir, la media.

Para calcular el fitness usaremos la siguiente expresión:

$$f_4 = (ds_{max} - \ ds) \ / \ (ds_{max}) \ .$$

Objetivo global

Una vez que tenemos los 4 objetivos usamos la siguiente expresión para ponderarlos:

$$f_t = f_1 \ \times \ \mu_1 + f_2 \ \times \ \mu_2 + \ f_3 \ \times \ \mu_3 + \ f_4 \ \times \ \mu_4 \ .$$

De esta manera el problema multiobjetivo se transforma en un problema de optimización de un solo objetivo, en el cual los pesos se obtienen del método Rank Order Centroid (ROC) (Butler et al. (1999)). Este método nos permite tener en cuenta la información ordinal proporcionada por los expertos de CRIDA, en ROC se aplica la siguiente fórmula:

$$w_i = \frac{\sum_{j=i}^{n} 1/j}{n}, i = 1, ..., n.$$

Como hay 4 pesos tenemos que aplicarla cuatro veces:

$$w_1 = \frac{1+1/2+1/3+1/4}{4} = 0{,}52,$$

$$w_2 = \frac{1/2+1/3+1/4}{4} = 0{,}27,$$

$$w_3 = \frac{1/3+1/4}{4} = 0{,}15,$$

$w_4 = \frac{1/4}{4} = 0{,}06$.

Y así damos los valores 0.52 , 0.27 , 0.15 y 0.06 a los pesos μ_1, μ_2, μ_3 y μ_4, respectivamente.

Capítulo 3

Resultados y ajuste paramétrico

ESTE capítulo presenta el análisis hecho de los resultados y como se han ido ajustando los parámetros para lograr conseguir una solución eficiente en términos de velocidad de cálculo y calidad de ésta.

3.1 Ejemplo de prueba

Los ejemplos se han realizado con el turno de noche de Canarias, donde existen 4 controladores con capacitación CON y 9 con capacitación PTD (ruta + Aproximación).

Existen 114 slots de trabajo de 5 minutos, es decir, el turno dura 9 horas y media.

En este caso siempre están abiertos 3 sectores (no siempre los mismos) y el número de sectores que intervendrán en este turno es 5.

La sectorización de dicho turno se muestra en la Figura 3.1:

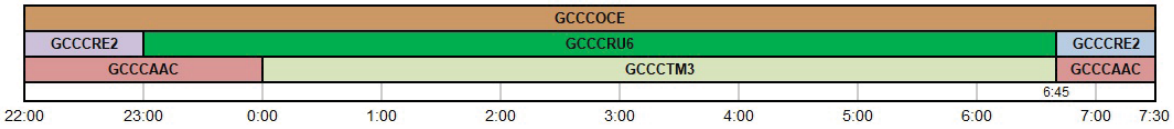

Figura 3.1: Sectorización del turno nocturno de Canarias

En este centro de control existen dos núcleos. Los sectores de aproximación GCCCTM3 y GCCCACC pertenecen al núcleo 1, mientras que los sectores GCCCOCE, GCCCRE2 y GCCCRU6 son sectores en ruta y pertenecen a ambos núcleos.

En la Figura 3.1 se muestra la sectorización donde el sector GCCCOCE está abierto durante el turno, los sectores GCCCAAC y GCCCRE2 están abiertos al principio y al final del turno y los sectores GCCCTM3 y GCCCRU6 están abiertos en el centro del turno.

GCCCTM3 y GCCCACC, y GCCCRE2 y GCCCRU6 son sectores afines.

Los 4 controladores CON pueden trabajar en el núcleo 1 y los 9 controladores PTD en el núcleo 2.

3.2 Solución de referencia

Desde CRIDA se nos ha proporcionado una solución referencia que han usado ellos. Para poder medir nuestra solución frente a la de referencia, calculamos el fitness con nuestra función de fitness de esa solución.

Fitness Total: 0.8811179608786343

Fitness del objetivo 1: 0.927429149797571

Fitness del objetivo 2: 0.7986725663716814

Fitness del objetivo 3: 0.9102564102564102

Fitness del objetivo 4: 0.7779124754180322

La representación gráfica de la solución referencia es mostrada en la Figura 3.2:

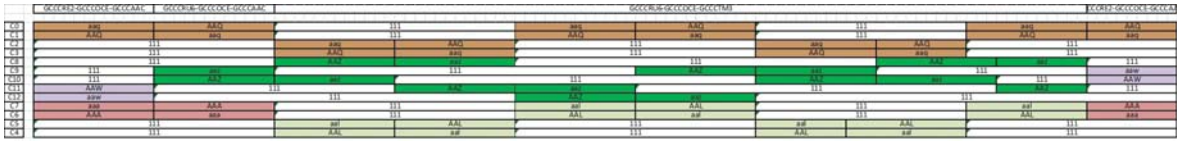

Figura 3.2: Solución referencia proporcionada por CRIDA

3.3 Solución infactible

Una vez tenemos todos los datos del problema usando la heurística de la primera fase obtenemos varias soluciones infactibles, una solución tipo generada es la siguiente:

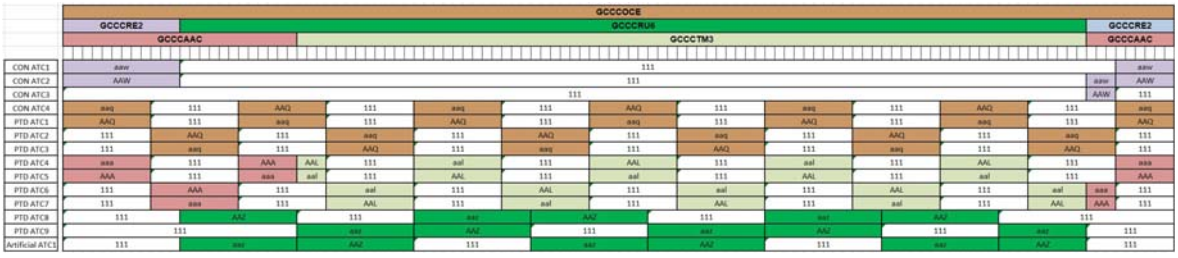

Figura 3.3: Solución infactible generada con la heurística de la primera fase

Como el sector GCCCOCE está abierto durante toda la noche, este sector se cubre con una plantilla especial con 4 controladores (CON4, PTD1, PTD2 y PTD3) con periodos de descanso de 9 slots. Lo mismo se aplica para los sectores GCCCACC y GCCCTM3 que pueden ser vistos como un solo sector abierto toda la noche y ser cubiertos por cuatro controladores (PTD4, PTD5, PTD6 y PTD7). Periodos de descanso de 12 slots son considerados para el resto de controladores.

En este caso, la única condición no cumplida es que hay 14 controladores, cuando en realidad solo disponemos de 13.

3.4 Solución factible

Una vez tenemos las soluciones infactibles de la primera fase, en la segunda fase buscábamos lograr tener una solución factible.

El MIR de recocido simulado se ha usado con estos parámetros: se ejecuta el algoritmo propuesto en Ben-Ameur (2004) con diferentes temperaturas iniciales para ajustar la temperatura inicial, usando un ratio de aceptación de 0.95, llegando a la conclusión de que T_0 debe ser 0.125.

El número de iteraciones en las que la temperatura no cambia lo fijamos a 1200 y ponemos la α a 0.9.

La búsqueda finaliza cuando el fitness no mejora al menos un 0.005 % durante 30000 iteraciones.

En la Figura 3.4 se muestra la evolución del fitness durante el proceso de búsqueda, mostrando claramente donde el número de controladores disminuye en las primeras iteraciones y como las condiciones laborales no cumplidas empiezan a disminuir a medida que transcurren las iteraciones hasta que una solución factible (fitness = 1) es alcanzada.

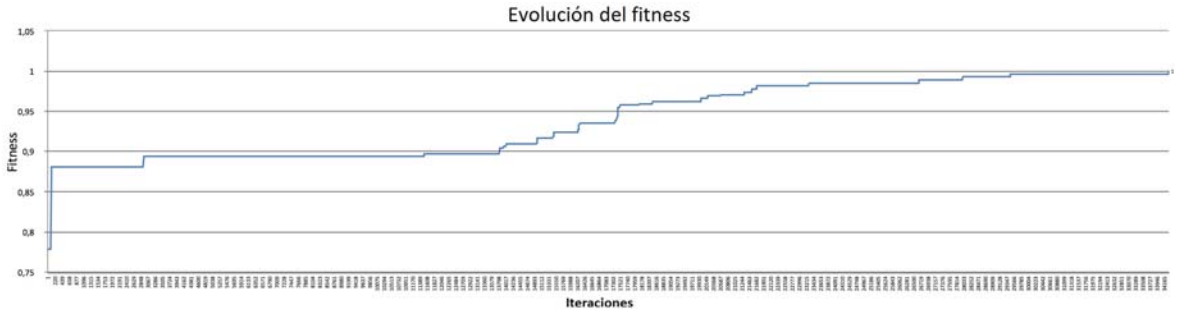

Figura 3.4: Evolución del fitness en la fase 2 hasta alcanzar una solución factible

Un ejemplo de solución factible, obtenida al final de la esta fase es el siguiente:

Fitness Total: 0.7046085345000965

Fitness del objetivo 1: 0.7373008130533435

Fitness del objetivo 2: 0.6146755162241888

Fitness del objetivo 3: 0.7264957264957265

Fitness del objetivo 4: 0.7712560559578018

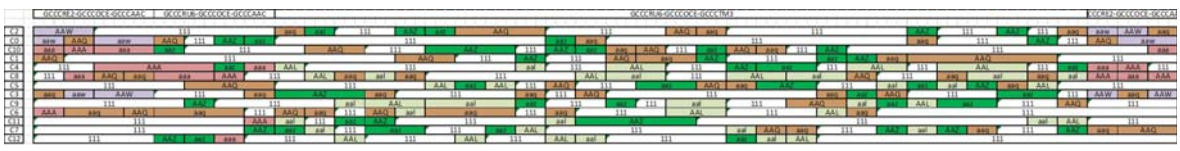

Figura 3.5: Solución factible obtenida al final de la primera fase

Como se puede observar, el fitness no es bueno comparado con la solución referencia, y además a la vista está muy desordenada de cara al objetivo 2. Por lo tanto, esta solución será optimizada usando búsqueda tabú. En las siguientes secciones se muestra como se ha llevado a cabo la optimización.

3.5 Primeros resultados experimentales

El primer experimento lo vamos a realizar con los siguientes parámetros:

numeroGeneracionesPorIteracion=100 o 1000

tabuBan = 3

penalizacionLT = 0.01

iteracionesExtra = 0

it1 = 50

it2 = 50

valorAMejorar = 0.999

limiteTiempo = 0

limiteIteraciones = 100

Es decir, realizaremos 100 iteraciones que consistirán en 2 fases, 50 iteraciones de intensificación, 50 de diversificación.

El tiempo tabú será de 3 iteraciones durante la fase de intensificación y penalizaremos con 0.01 cada repetición de movimiento durante la fase de diversificación.

Por cada iteración calcularemos el fitness de 100 soluciones del entorno de soluciones, esto lo repetimos varias veces. Haremos lo mismo, pero esta vez generando 1000 soluciones por iteración, repetimos esto varias veces también y comparamos los resultados en una gráfica:

Las trazas 1 y 2 corresponden a experimentos en los que se generan 100 soluciones por iteración mientras que en la 3 y la 4 se generan 1000.

Con respecto a la gráfica mostrada las trazas mostradas han tardado lo siguiente:

Traza1: 178682 ms

Traza2: 164291 ms

Traza3: 1623132 ms

Traza4: 1832417 ms

Como se puede ver, el tiempo incrementa demasiado, y no provoca ningún cambio significativo en el fitness, así que para el futuro, dejaremos el valor de generar X número de

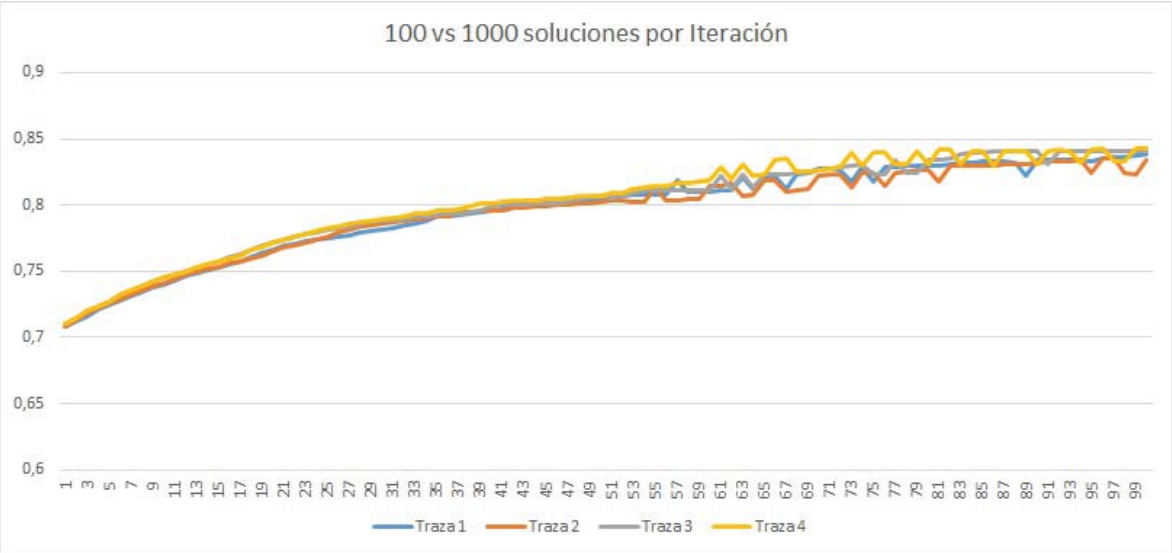

Figura 3.6: La traza 1 y 2 muestra la evolución del fitness generando 100 soluciones por iteración, en las trazas 3 y 4 se generan 1000.

soluciones por iteración con un valor de 100.

3.6 Análisis de la evolución de cada objetivo

Ahora vamos a fijarnos en la evolución de cada objetivo, para ello realizamos varias trazas del algoritmo. Y observamos el resultado en la Tabla 3.1.

En la primera columna se muestra el número de iteración seguido por el fitness global ponderado y el fitness de los 4 objetivos.

Como podemos observar el objetivo que tiene el valor más bajo es el objetivo 2 (columna 4), es decir, el de los estadillos. Esto puede deberse, en parte, a que el máximo valor teórico sea demasiado superior al real.

Sin embargo, las características de este objetivo permiten una pequeña mejora.

Este objetivo puede ser mejorado sin que se afecte a los demás, es decir tan solo intercambiando filas/controladores es posible mejorar éste sin alterar los demás.

Este proceso puede hacerse simplemente al final, una vez que tenemos la solución intercambiar todos los controladores entre sí y volver a calcular el fitness de cada una de las nuevas soluciones y elegir el mejor.

Sin embargo, es un proceso muy rápido, es decir, hay muy pocos intercambios de controladores posible a partir de una sola solución y computacionalmente no es costoso. El número de permutaciones ascendería (siendo *n* el número de controladores) a:

$permutaciones = \frac{n \times (n-1)}{2}$.

Este proceso es tan computacionalmente poco costoso que en vez de realizar al final, podemos realizarlo en cada iteración.

1	0,708445947	0,743002746	0,615044248	0,730769231	0,77345313
2	0,713254037	0,751357789	0,613938053	0,739316239	0,764787935
3	0,71699664	0,757267347	0,613569322	0,743589744	0,766924031
4	0,7224622	0,762060363	0,616887906	0,752136752	0,780176069
5	0,725368387	0,763663602	0,621681416	0,756410256	0,782463215
6	0,728276043	0,76687008	0,622787611	0,764957265	0,776789287
7	0,732516084	0,771663096	0,624262537	0,773504274	0,777912475
8	0,734833651	0,776378504	0,624262537	0,773504274	0,775671722
9	0,737735917	0,779451379	0,627212389	0,777777778	0,77345313
10	0,740026181	0,78336231	0,627949853	0,782051282	0,76372713
...	...	...	...	...	...
...	...	...	...	...	...
90	0,813862669	0,846300439	0,699115044	0,854700855	0,947004184
91	0,813582673	0,845953441	0,698746313	0,854700855	0,947004184
92	0,813290141	0,845248313	0,700958702	0,854700855	0,938284003
93	0,812647559	0,846834852	0,698746313	0,854700855	0,923780056
94	0,812348886	0,846834852	0,697640118	0,854700855	0,923780056
95	0,813103831	0,846300439	0,699115044	0,854700855	0,934356888
96	0,813407453	0,847005567	0,69800885	0,854700855	0,938284003
97	0,813402504	0,846300439	0,700221239	0,854700855	0,934356888
98	0,813085585	0,844179487	0,701327434	0,854700855	0,942478609
99	0,812538184	0,843457659	0,700958702	0,85042735	0,951954162
100	0,812941425	0,846283738	0,699115044	0,85042735	0,942478609
...	...	...	...	...	...
...	...	...	...	...	...
380	0,856589071	0,909277665	0,735619469	0,884615385	0,874252007
381	0,855998618	0,908208839	0,731932153	0,88034188	0,900950967
382	0,856025185	0,90768556	0,733038348	0,88034188	0,900950967
383	0,856305209	0,908385121	0,736725664	0,876068376	0,893646003
384	0,856215248	0,911574899	0,730825959	0,88034188	0,880366827
385	0,857128043	0,908214406	0,736725664	0,876068376	0,908839431
386	0,857447493	0,907679993	0,738938053	0,876068376	0,908839431
387	0,857526742	0,90714558	0,738569322	0,871794872	0,927134878
388	0,856581407	0,906093455	0,737094395	0,871794872	0,927134878
389	0,856569439	0,903972503	0,735619469	0,871794872	0,951954162
390	0,855422094	0,902914811	0,73340708	0,871794872	0,951954162

Tabla 3.1: Evolución del fitness. Se puede observar el número de iteración, el fitness global y el fitness de los 4 objetivos

Por lo tanto, por cada solución elegida como la mejor de la actual iteración, la permutamos, y elegimos la mejor permutación. Además, esto no rompería la factibilidad y sólo mejoraría el objetivo 2.

En principio parece que no es un cambio muy grande y que si ayuda a elevar un poco el fitness sin afectar en gran medida al tiempo de computación, aunque tras unos experimentos mostrados en las gráficas de la Figura 3.7 observamos que esto no es así.

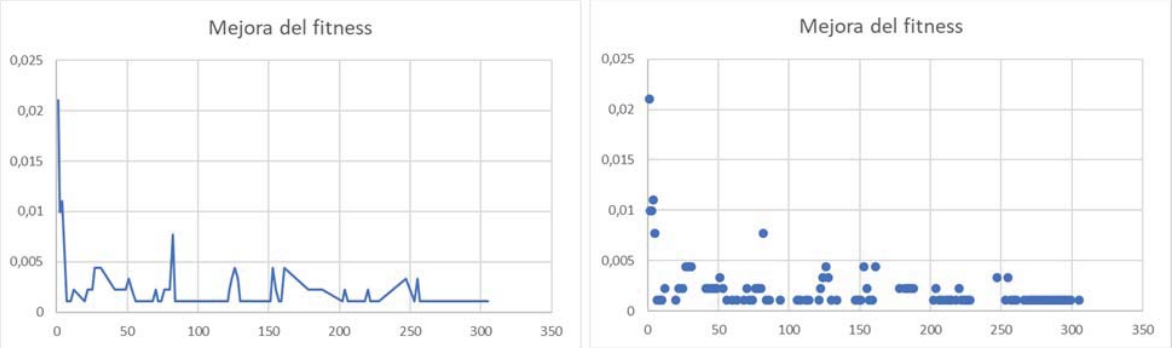

Figura 3.7: Evolución de la mejora en el fitness del objetivo 2 con la mejora introducida

En la gráfica se muestra el número de iteración junto con la mejora producida en el fitness del objetivo 2, los saltos en el número de iteración indican que el proceso no ha servido para mejorar el fitness.

Podemos observar que si hay una mejora considerable en las primeras iteraciones, sin embargo el proceso de mejora no obtiene demasiados buenos resultados a medida que avanza la ejecución.

Por lo tanto, dada la poca mejora que se produce y el coste computacional que ello conlleva (aunque sea pequeño no compensa), decidimos descartar esta mejora en cada iteración y realizarla únicamente en la solución final.

Para ello, lo que haremos será permutar las filas a pares hasta que la permutación no arroje ninguna mejora en el fitness, sería una pequeña optimización de la solución final y muy probablemente caemos en un mínimo local.

Hacerlo correctamente implicaría extender el software con una cuarta fase, que implemente una metaheurística para resolver este problema, aunque creemos que la mejora en el fitness no sea muy alta, sí que podría ser interesante tenerlo en cuenta como posible trabajo futuro.

3.7 Ajustando parámetros

Tras observar los resultados de los valores del fitness y resultados adicionales de la trazas del software utilizado nos damos cuenta de que las variaciones de fitness entre una solución y la siguiente giran en torno a la unidad de las milésimas, por lo tanto penalizar a las centésimas

por cada movimiento repetido en la fase de diversificación es demasiado, así que a partir de ahora penalizaremos con 0.001.

En el siguiente experimento mostramos 6 ejecuciones, en las ejecuciones 1-3 fijamos el valor de penalización tabú a 3 y el de penalización por movimiento repetido a 0.001.

En las ejecuciones 4-6 los fijamos en 10 y 0.01, respectivamente.

La configuración paramétrica es la siguiente:

numeroGeneracionesPorIteracion=100

tabuBan = 3 o 10

penalizacionLT = 0.001 o 0.01

iteracionesExtra = 0

it1 = 100

it2 = 100

valorAMejorar = 0.882

limiteTiempo = 0

limiteIteraciones = 0

Los resultados se muestran en la Figura 3.8:

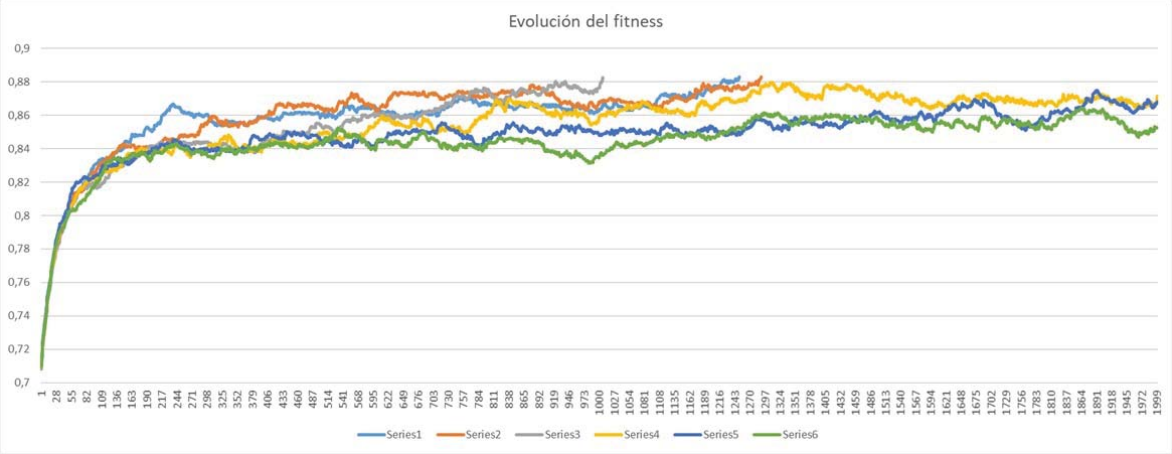

Figura 3.8: Evolución del fitness

Como se puede ver, las más favorecidas son las trazas de la 1, 2 y 3. En concreto, si ponemos como criterio de parada el valor del fitness de referencia, aproximadamente 0.882, las trazas se detienen en las iteraciones 1251, 1291 y 1007 respectivamente.

Las trazas 4, 5 y 6 superan las 2000 iteraciones sin llegar a detenerse, (la traza 6 lo hace en la iteración 2006).

Así que a partir de ahora trabajaremos con un valor tabú de 3 y un valor de penalización por repetición de movimiento de 0.001. Tras inspeccionar en detalle éstas y otras trazas se

ha llegado a la conclusión de que este último valor si es relevante, en cambio el valor tabú no influye mucho.

Ahora, procedemos a comparar el hecho de que alternar entre diversificación e intensificación cada 50 o 100 iteraciones sea mejor o peor.

Para 100 iteraciones usaremos los resultados de las trazas 1, 2 y 3. Para 50 iteraciones conducimos un nuevo experimento, y así obtenemos las trazas 7, 8 y 9 que se muestran en la Figura 3.9.

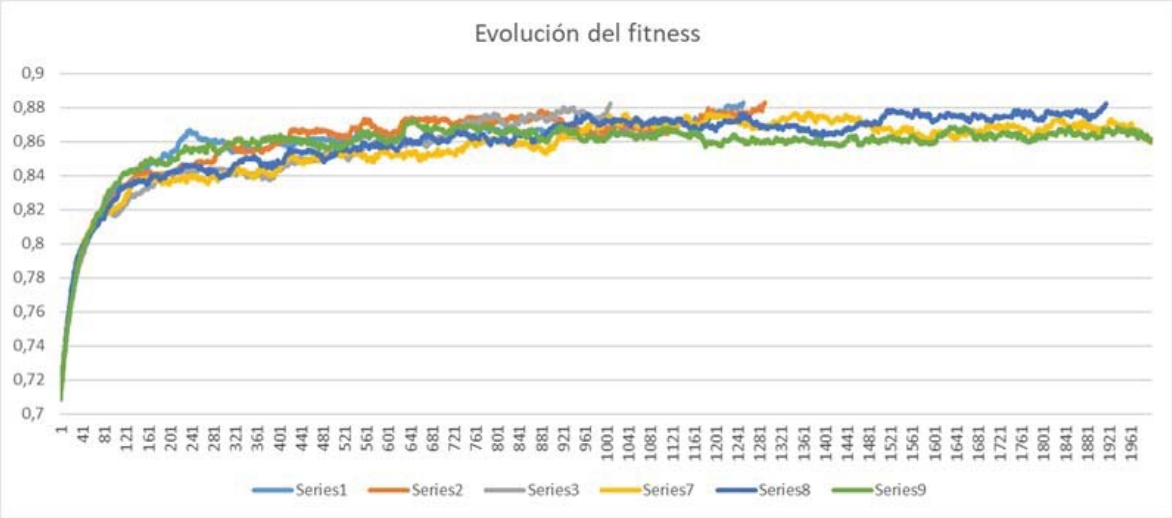

Figura 3.9: Evolución del fitness

Según los resultados, cambiar de modo cada 50 iteraciones ofrece resultados peores que hacerlo cada 100. Experimentos adicionales indican que hacerlo cada 200 ofrece peor rendimiento que estos dos.

3.8 Objetivo: Superar el fitness de CRIDA

Una vez tenemos los parámetros ajustados el siguiente objetivo es superar el fitness de la solución referencia proporcionada por CRIDA. Por lo tanto fijaremos como criterio de parada 0.882, recordemos que el fitness referencia estaba en 0.8811179608786343. El ajuste paramétrico llevado a cabo es el siguiente:

numeroGeneracionesPorIteracion=100

tabuBan = 3

penalizacionLT = 0.001

iteracionesExtra = 0

it1 = 100

it2 = 100

valorAMejorar = 0.882

limiteTiempo = 0

limiteIteraciones = 0

Tras la ejecución hemos logrado superar el fitness referencia en 1007 iteraciones. Y ha tardado 1727422 ms, aproximadamente 29 minutos en alcanzar este nivel de optimización.

La evolución del fitness se representa en la Figura 3.10:

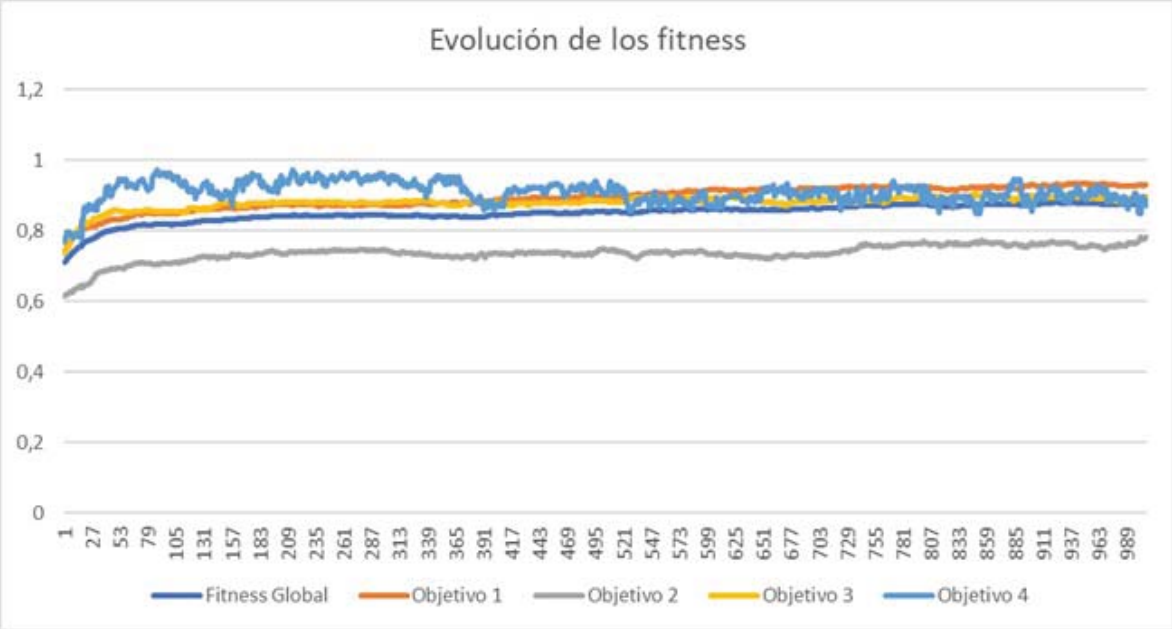

Figura 3.10: Evolución del fitness de cada objetivo

La representación gráfica de la solución aparece en la Figura 3.11:

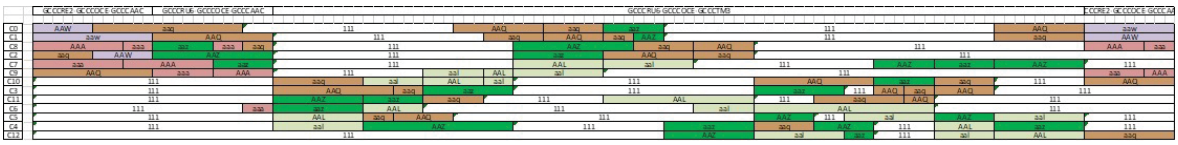

Figura 3.11: Representación de la solución obtenida

Tras permutar las filas de los controladores para mejorar el objetivo 2 del fitness la solución obtenida tiene el fitness siguiente:

Fitness Total: 0.8830869880555445 (Antes de la mejora 0.8824896429228011)

Fitness del objetivo 1: 0.9311422064777328

Fitness del objetivo 2: 0.7853982300884956 (Antes de la mejora 0.7831858407079646)

Fitness del objetivo 3: 0.8974358974358975

Fitness del objetivo 4: 0.8703355657974168

Como se puede observar el fitness mejora ligeramente al aplicar la optimización del objetivo 2.

Si comparamos estos resultados con los de la solución referencia podemos ver que nuestra solución mejora en gran medida la distribución homogénea de la carga de trabajo entre los distintos controladores. El resto de objetivos se mantienen prácticamente en los mismos valores.

La solución final se puede ver en la Figura 3.12:

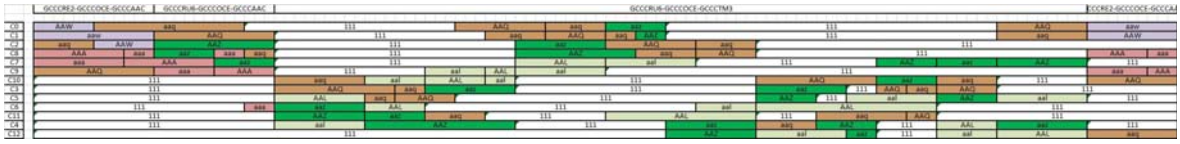

Figura 3.12: Representación de la solución obtenida una vez mejorada

3.9 Mejor resultado

Una vez hemos superado el fitness proporcionado por CRIDA de la solución de referencia, hemos dejado el algoritmo durante más tiempo, aunque a estas alturas cada vez es más difícil mejorar el fitness, es importante destacar la mejor solución obtenida.

La mejor solución que hemos encontrado es la siguiente:

Fitness Total: 0.8943669811671088

Fitness del objetivo 1: 0.9341501349527666

Fitness del objetivo 2: 0.8101032448377581

Fitness del objetivo 3: 0.9102564102564102

Fitness del objetivo 4: 0.889042889116899

Como se puede apreciar el fitness es considerablemente mayor que el de la solución referencia (0.8811).

La representación gráfica de la solución es la siguiente:

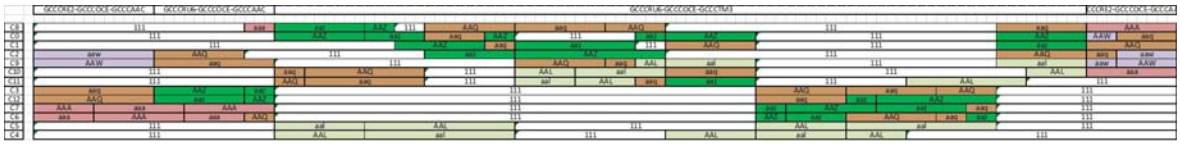

Figura 3.13: Representación de la mejor solución obtenida

Capítulo 4

Conclusiones y trabajo futuro

HEMOS propuesto un enfoque que resuelva el problema de la asignación de controladores a sectores aéreos. Dado un número fijo de controladores y una sectorización del espacio aéreo, optimizamos varios objetivos que tienen que ver con los periodos de trabajo y descanso de los controladores así como con sus posiciones de trabajo, la estructura de la solución y el número de cambios de sala. Además, se trata de buscar que la carga de trabajo entre los distintos controladores esté repartida uniformemente.

Proponemos una metodología basada en tres fases. En la primera fase, utilizamos una heurística para obtener soluciones iniciales infactibles basadas en unas plantillas proporcionadas. A continuación, en la segunda fase, usamos un algoritmo MIR (Multiple Independent Run) basado en recocido simulado para tratar de alcanzar soluciones factibles. Por último, en la tercera fase, usamos la búsqueda tabú a partir de las soluciones factibles obtenidas en la segunda fase con el objetivo de optimizar las funciones objetivo. Para ello tenemos en cuenta información ordinal sobre estos, de esta manera transformamos el problema multiobjetivo de optimización en uno de un solo objetivo usando el método del centroide. Adicionalmente donde se pueda usamos expresiones regulares para comprobar las condiciones de trabajo de los controladores, permitiéndonos realizar esto con rapidez.

Hemos modelado el problema aplicando los algoritmos usados al contexto de este problema en un software propio.

Por último, hemos ejecutado este software con los datos de un ejemplo real, y hemos conseguido mejorar una solución de referencia que nos ha sido proporcionada por expertos en el dominio, además uno de los objetivos planteados, que consistía en realizar una distribución homogénea de la carga de trabajo entre los distintos controladores ha sido mejorado notablemente.

Los resultados son muy satisfactorios, ya que podemos demostrar que es posible mejorar una solución hecha a mano con unas plantillas a través del uso de técnicas de inteligencia artificial y un uso adecuado de los recursos disponibles. Además, la solución obtenida, aparte de mejorar la proporcionada por los expertos sigue siendo fácilmente comprensible por estos (gracias a que hemos tratado de buscar que esta tenga cierto parecido a los estadillos que estos usan actualmente), por lo que es posible que ellos puedan realizar modificaciones sobre ésta

sin que exista una dificultad añadida.

Por último, queremos destacar ciertas aspectos que pueden ser tenidos en cuenta para tratar de mejorar los resultados obtenidos con nuestro enfoque.

4.1 Trabajo futuro

Aparte de la metaheurística que ha sido usada para la fase de optimización, sería interesante poder comparar estos resultados con los que podrían ofrecer otras.

Quizá el uso de otras metaheurísticas pueda ofrecer mejores resultados, bien sea, que obtengan una solución buena (por buena entiéndase, que mejore el fitness de la solución de los expertos) en menor tiempo o que lleguen a obtener una solución con un fitness mejor en un tiempo razonable.

Para finalizar sería interesante, una vez obtenida una solución optimizada, ampliar nuestro enfoque con una cuarta fase, que se reduzca a un problema de optimización en el que se permuten las filas de la solución con el objetivo de mejorar el fitness del objetivo 2, es decir, el que busca que la solución sea similar a los estadillos.

Referencias

Alsmadi, O. MK., Abo-Hammour, Z. S., Abu-Al-Nadi, D. I., & Algsoon, A. (2011). A novel genetic algorithm technique for solving university course timetabling problems. IEEE.

Alvarez, R., Crespo, E., & Tamarit, J. M. (2002). Design and implementation of a course scheduling system using Tabu search. European Journal of Operational Research, 137, 512–523.

Arnving, M., Beermann B.., Koper B., Maziul M., Mellett U., Niesing C., Vogt J. (2006). Managing shiftwork in European ATM, Literature Review. European Organization for the Safety of Air Navigation.

Aycan, E., & Ayav, T. (2008). Solving the course scheduling problem using simulated annealing. IEEE.

Babaei, H., Karimpour J., Hadidi A. (2015). A survey of approaches for university course timetabling problem. Computers & Industrial Engineering 86, 43-59.

Ben-Ameur, W. (2004). Computing the initial temperature of simulated annealing. Computational Optimization and Applications 29, 369-385.

Butler, J., Olson, D.L., 1999. Comparison of centroid and simulation approaches for selecting sensitivity analysis. Journal of Multi-Criteria Decision Analysis 8, 146-161.

Cerny, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal of Optimization Theory and Applications 45, 41-51.

Días, T.M., Feber, D.F., DeSouza, C.C., Moura, A.V. (2003). Constructing nurse schedules at large hospitals. International Transactions in Operational Research 10(3), 245-265.

EUROCONTROL (2006). Shiftwork practices study - ATM and related industries, DAP/SAF-2006/56 Brussels: EUROCONTROL.

Friedl, J.E.F. (1997). Mastering Regular Expressions, O'Reilly and Associates.

Fred Glover, Belén Melián BÚSQUEDA TABÚ Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, año/vol. 7, número 019,pp. 29-48.

Hajek, B. (1988). Cooling schedules for optimal annealing. Mathematics of Operations Research 13 (2), 311-329.

Joudaki, M., Imani, M., & Mazhari, N. (2010). Using improved Memetic algorithm and local search to solve University Course Timetabling Problem (UCTTP). Doroud, Iran: Islamic Azad University.

Khonggamnerd, P., & Innet, S. (2009). On improvement of effectiveness in automatic university timetabling arrangement with applied genetic algorithm. IEEE.

Kirkpatrick, S., Gelatt., C.D., Vecchi, M. P. (1983). Optimization by simulated annealing. Science 220, 671- 680.

Lewis, M. R. R. (2006). Metaheuristics for university course timetabling. Ph.D. Thesis, Napier University.

Mateos, A., Tello, F., Jiménez-Martín, A. and Fernández del Pozo, J.A. The ATC work shift scheduling problem based on multistart simulated annealing and regular expressions. Journal of Air Transport Management, under review.

Mayer, A., Nothegger, C., Chwatal, A., & Raidl, G. (2008). Solving the post enrolment course timetabling problem by ant colony optimization. In Proceedings of the 7th international conference on the practice and theory of automated timetabling.

N. Mladenović and P. Hansen. 1997. Variable neighborhood search. Comput. Oper. Res. 24, 11 (November 1997), 1097-1100.

Obit, J. H. (2010). Developing novel meta-heuristic, hyper-heuristic and cooperative search for course timetabling problems. Ph.D. Thesis, School of Computer Science University of Nottingham.

Post, G., Kingston, J., Ahmadi, S., Daskalaki, S., Gogos, C., Kyngas, J., et al. (2014). XHSTT: an XML archive for high school timetabling problems in different countries. Annals Operations Research, 218(1), 295-301.

Ribeiro, C.S. (2012). Sports scheduling: Problems and applications. International Transactions in Operational Research 19 (1-2), 201-226.

Shengxiang, Y., & Jat, N. S. (2011). Genetic algorithms with guided and local search strategies for university course timetabling. IEEE Transactions on Systems, MAN, and Cybernetics-PART C: Applications and Reviews, 41, 1.

Socha, K., Knowles, J., & Samples, M. (2002). A max-min ant system for the university course timetabling problem. In Proceedings of the 3rd international workshop on ant algorithms (ANTS 2002). Lecturer notes in computer science (Vol. 2463, pp. 1–13). Springer-Verlag.

Stojadinovic, M. (2015). Hybrid of hill climbing and SAT solving for air traffic controller shift scheduling, Journal of Information Technology and Applications 2, 81-87.

Este documento fue editado y tipografiado con LaTeX empleando la clase **etsii-tfm** (versión 0.20170712) que se puede encontrar en:
`https://bitbucket.org/Gonlos/etsii-tfm`