Escuela Técnica Superior de Ingenieros Informáticos

Universidad Politécnica de Madrid

# ASIGNACIÓN DE CONTROLADORES A SECTORES AÉREOS MEDIANTE METAHEURÍSTICAS TRAYECTORIALES: VARIABLE NEIGHBOURHOOD SEARCH

## Trabajo Fin de Máster

Máster Universitario en Inteligencia Artificial

AUTOR: Pablo Lozano Santiuste
TUTOR/ES: Alfonso Mateos Caballero

2018

# AGRADECIMIENTOS

En primer lugar, me gustaría dar las gracias por la toda ayuda brindada a mi tutor Alfonso Mateos Caballero, que siempre me ha orientado cuando me he visto estancado en el desarrollo de este trabajo y me ha sabido dar las claves para continuar con él. Quería hacer también mención especial a Tino, por siempre haberme resuelto las dudas en el acto y haberme proporcionado material de soporte así como ideas durante el desarrollo de este TFM.

En segundo lugar, me gustaría dedicar este trabajo a mi familia, que siempre me ha apoyado y aguantado desde que tengo uso de razón.

En tercer lugar, me gustaría agradecer a Alba de Blas y a Sergio Liébana por el gran apoyo que me han otorgado en todo momento, y espero que se lo pueda devolver algún día.

En cuarto lugar, me gustaría agradecer a todos los compañeros del máster los buenos recuerdos que me llevo de aquí gracias a lo grandes personas que son, en especial a Francisco Laport, que se convirtió, en pocos meses, en un hermano que espero conservar durante mucho tiempo.

# RESUMEN

Este trabajo trata de dar solución a un problema de timetabling, propuesto por CRIDA, en el que se intenta asignar ciertos controladores aéreos a un determinado número de sectores abiertos en un tiempo determinado. Una de las principales complejidades del problema son las numerosas condiciones que deben cumplir las soluciones para ser factibles. Esta asignación se está realizando hasta ahora mediante un sistema de plantillas que tiene grandes limitaciones. Por esto, para la resolución del problema se ha propuesto el uso de la metaheurística *Variable Neighbourhood Search* (*VNS*).

La metodología que se propone para resolver el problema de asignación de controladores a sectores aéreos consta de tres fases. En la primera fase se formarán soluciones iniciales a partir de la plantilla optimizada de controlar un sector con tres controladores, y estas que servirán como punto de partida para generar soluciones factibles. En la segunda fase se aplicará la metaheurística VNS para la generación de soluciones factibles a partir de las soluciones iniciales, este será el principal desarrollo de este trabajo. En la tercera fase, la metaheurística también se aplicará para optimizar la solución factible de acuerdo a cuatro funciones objetivo.

La metaheurística logra importantes resultados en la segunda fase, donde es capaz de encontrar soluciones factibles prácticamente en todos los casos propuestos e incluso en dos de ellos consigue solventar la tarea con menos recursos de los disponibles. Además, en la tercera fase es capaz de encontrar soluciones mejores que las que se aplican actualmente en casos reales.

## SUMMARY

This work aim to solve a timetabling problem proposed by CRIDA where they need to build some air traffic controller schedules among some sector of the space filling a fixed amount of time. One of the main issues of the problem are the strong conditions that must be accomplished for a solution to be feasible. A *Variable Neighborhood Search (VNS)* approach is proposed to deal with this task.

The first step of the proposed solution is the construction of some templates that will be used as initial points for generating the feasible solutions. At the second step, VNS is applyed for building the feasible solution and this will be of main objective of this work. The metaheuristic will be also applied for the third phase, that it's aimed to optimize the feasible solution according to 4 objective solutions.

The metaheuristic achieves great results at the second step of the methodogy proposed finding feasible solutions in almost every studied case, even in one of them is cappable to find feasible solutions with fewer resources than expected. Also at the step 3 is finding better solutions that the nowadays implemented in real cases in CRIDA.

# Índice

# Índice de figuras

# Índice de tablas

# 1. Introducción

En la actualidad, unos de los medios de transporte más seguros, para media y larga distancia, es el avión, y ésto es en parte gracias a la labor que realizan los controladores aéreos de cada aeropuerto para gestionar el tránsito de los aviones.

El espacio aéreo esta dividido en distintos sectores, puesto que un controlador sólo no podría garantizar la seguridad del espacio aéreo. Los sectores van cambiando con el paso del tiempo, abriéndose, cerrándose o cambiando su configuración. En cada sector trabajan dos controladores distintos, uno que ejerce como ejecutivo, transmitiendo las intrucciones a los pilotos para solucionar conflictos y otro como planificador, cuya tarea es tratar de anticiparse a los conflictos y transmitir la información al ejecutivo para que los resuelva. De esta manera se minimiza la posibilidad de un error humano.

Actualmente, estos controladores aéreos poseen diversas restricciones que limitan su actividad durante la jornada laboral con el objetivo de prevenir la fatiga y otros efectos que puedan tener repercusión en su actividad laboral. Esto complica la tarea de organizar los horarios de los controladores aéreos necesitando un número elevado de los mismos para poder satisfacer todas las restricciones y elaborar un horario factible. En la literatura, la resolución este tipo de problemas de secuenciación se denomina *Timetabling*.

## 1.1. Objetivo

Este trabajo tiene como objetivo resolver un problema real que tiene CRIDA, el cuál es un centro de investigación, desarrollo e innovación en el ámbito de la gestión del tráfico aéreo en España. Este problema surge de la complicada y habitual tarea de establecer los horarios de trabajo de los controladores aéreos de manera que cubran todos los sectores aéreos durante una jornada laboral, sin violar ninguna de las normas que actualmente rigen este tipo de puestos de trabajo.

Para resolver este problema, con anterioridad se ha propuesto un método basado en la metaheurística de *Recocido Simulado*, descrito en Tello (2015), obteniendo buenos resultados, pero con la idea de que podrían ser mejores usando alguna otra metaheurística. Por eso, mediante este trabajo se propone el uso de la metaheurística *Variable Neigbourhood Search* para la búsqueda y optimización de soluciones factibles. El método propuesto consta de tres fases:

1. En la primera fase se construirán soluciones iniciales haciendo uso de unas plantillas.

2. En la segunda fase se tratará de encontrar una solución factible que verifique todas las restricciones que se definen en los apartados posteriores.

3. En la tercera fase se tratará de optimizar la solución factible de manera que mejore cierta función objetivo que se compone de cuatro términos, los cuales cuantifican aspectos tales como:

- El tiempo de trabajo entre dos periodos de descanso debe acercarse a 90 minutos, el periodo de trabajo en la misma posición y sector de trabajo ha de ser cercano a 45 minutos y el porcentaje de tiempo que cada trabajador está en un rol de ejecutivo ha de estar entre el 40 % y 60 %.
- La solución debe ser similar a unos estadillos proporcionados por CRIDA de manera que sea de fácil comprensión para personal del centro de control y permitiera hacer cambios a mano.
- El número de cambios de sala deberá ser mínimo, para esto minimizaremos el número de periodos de descanso.
- La carga de trabajo deberá estar balanceada entre los trabajadores.

## 1.2. Estructura del documento

A continuación se resume el contenido de cada sección de este trabajo.

**Sección 2. Metaheurísticas**

En esta sección se realiza un rápido estudio acerca de las metaheurísticas enfocándose en el método *Variable Neighbourhood Search*.

**Sección 3. Descripción del problema**

En esta sección haremos una introducción a los problemas de *Timetabling* y definiremos distintos conceptos específicos de nuestro problema con el objetivo de proporcionar al lector una base de conocimiento que le servirá de ayuda en secciones posteriores.

**Sección 4. Método de resolución del problema**

En esta sección explicaremos detalladamente el método de resolución propuesto para este problema, definiendo las distintas características del algoritmo y explicando las decisiones que se han tomado en el transcurso de este trabajo.

**Sección 5. Análisis de resultados**

En esta sección se estudiará como se comporta el método propuesto en cinco casos reales proporcionados por CRIDA.

**Sección 6. Conclusión y trabajos futuros**

En esta sección se elaborará una conclusión final acerca de este TFM con el objetivo de evaluar el trabajo realizado y se propondrán nuevas vías de estudio relacionadas con el tema tratado.

## 2. Metaheurísticas

Las metaheurísticas surgen de la combinación de distintas estrategias de la Inteligencia Artificial tales como la evolución biológica con el objetivo de mejorar los métodos aproximados de optimización combinatoria. Se podrían definir como un marco que trata de dirigir distintas heurísticas a través del espacio de soluciones posibles marcando las pautas de exploratoriedad y explotación e incluso cuando cambiar la heurística utilizada.

A menudo, los problemas que poseen más de una función objetivo a optimizar (denominados problemas multiobjetivo) se encuentran dentro de la *Clase NP*. En esta clase se engloban los problemas que no poseen una complejidad polinómica, es decir, que el tiempo y los recursos a emplear para solucionar el problema crece de forma exponencial con el tamaño del problema.

Las heurísticas a menudo se desarrollaron para problemas específicos y no siempre funcionan correctamente en otros ámbitos. Unos de los principales problemas es el estancamiento en óptimos locales. Las metaheurísticas son algoritmos más generales que permiten ser aplicados a gran variedad de problemas.

Las metaheurísticas en la optimización multiobjetivo se clasifican en tres grandes grupos:

1. **De Búsqueda Global**, las cuales manejan una única solución en cada iteración. Entre ellas podemos encontrar el Recocido Simulado, Búsqueda Tabú, VNS...

2. **Evolutivas**, las cuales manejan un conjunto de soluciones llamado población en cada iteración. Entre ellas podemos encontrar los Algoritmos Meméticos, Algoritmos Evolutivos, Algoritmos de Nubes de partículas...

3. **Constructivas**, las cuales parten de una solución vacía y van añadiendo componentes hasta construir una solución.

A continuación se detallan diversas metaheurísticas muy utilizados que son capaces de optimizar hasta funciones no convexas y problemas combinatorios *NP-Hard*:

- **Recocido simulado**. Descrito por primera vez en Aarts and Korst (1988), hacen uso de la idea de los estados de agitación de los átomos de la materia para definir el comportamiento del sistema. Este funciona de tal manera que en cada iteración se visita un vecino que aunque sea peor, se escoge con cierta probabilidad calculada a partir de una exponencial. Esta probabilidad disminuye con el tiempo, a menudo, de forma geométrica. Esto consigue que el algoritmo sea más exploratorio al principio y al ir disminuyendo su energía de agitación, más intensivo al final. Si la temperatura disminuyese infinitamente lenta, se garantizaría llegar al óptimo global del sistema, pero esto llevado a la práctica es imposible.

- **Búsqueda Tabú**. Descrita por primera vez en Glover (1989), surge como una alternativa derivada de la búsqueda local donde va a usar listas tabú para evitar

pasar por ciertas soluciones y así intentar no incurrir en óptimos locales. La forma más básica de este algoritmo incluye en la lista tabú a aquellas soluciones que han sido visitadas con anterioridad en un cierto periodo de iteraciones.

Otras opciones pudieran ser incluir en la lista tabú soluciones con cierto tipo de atributos indeseables. Realizar este último tipo de listas es, en algunos casos, algo arriesgado. Es necesario tener un gran conocimiento sobre el problema, puesto que pueden ser soluciones por las que hay que atravesar para llegar a la solución óptima y nuestro algoritmo nunca llegaría. Este algoritmo siempre se movería hacia el vecino más prometedor, excluyendo claramente aquellos que se encuentren dentro de la lista tabú.

- **Algoritmos evolutivos**. Fueron ideados por primera vez en Bremermann (1962), donde se empezó a plantear métodos como la evolución y recombinación para abordar problemas de optimización y forman parte de la llamada computación natural. La idea de estos algoritmos es codificar los atributos del problema en forma de vectores de forma que se puedan aplicar sobre ellos cierto tipo de operaciones al estilo que se llevan a cabo en la genética real. En este tipo de algoritmos existen individuos, que son posibles soluciones de un problema, y poblaciones, que son conjuntos de individuos.

  La idea principal de estos algoritmos es cruzar individuos con una buen valor de la función objetivo, para intentar lograr hijos que mejoren las soluciones existentes al problema. La mutación otorgará un grado de aleatoriedad al algoritmo, facilitando la exploración del espacio de búsqueda. Aunque los operadores genéticos de cruce y mutación son muy importantes, otras etapas del algoritmo también tendrán mucha influencia en los resultados, como son la inicialización y la selección.

## 2.1. Búsqueda en entorno variable

La búsqueda en entorno variable (VNS) es una metaheurística que busca resolver el problema del estancamiento de los algoritmos de búsqueda en óptimos locales mediante la alteración de la definición de vecindad. Como hemos podido estudiar en la literatura, otros algoritmos como la búsqueda tabú utilizan listas de soluciones ya visitadas para intentar evitar incurrir en óptimos locales basándose en métodos que alteran el tamaño de vecindad, en este caso, excluyendo algunos vecinos de las posibles siguientes soluciones, pero lo que se propone y define en Mladenović and Hansen (1997) es el cambio en la definición de vecindad, y el cómo ir jugando con las distintas definiciones para, de esta manera, rotar el subespacio en el cual está buscando mejores soluciones cuando se vaya quedando anclado en óptimos locales.

A continuación, y a modo de introducción se plantean tres preguntas básicas que nos ayudarán a definir nuestro algoritmo y que iremos respondiendo a lo largo del trabajo. Éstas son:

1. ¿Cuántos entornos vamos a escoger y cuántos son?
2. ¿En qué orden debemos buscar a través de ellos?
3. ¿Qué estrategia debemos usar para cambiar entre los distintos entornos?

Antes de definir el esquema general del método VNS, vamos a estudiar algunas aproximaciones más simples.

### 2.1.1. Búsqueda en entorno variable en descenso

También conocida como VND por sus siglas en inglés (*Variable Neighborhood Descent*), es descrita en diversos artículos durante 2001 logrando unos buenos resultados en diversos fines como el rutado de arcos, Hertz and Mittaz (2001). Esta variante tiene la particularidad de que va cambiando de entorno de una manera determinística. En Algorithm 1 podemos observar como en la línea 1 se determina el conjunto de vecindades y su orden, en la línea 2 seleccionamos una solución inicial, posteriormente entramos en un bucle hasta que hayamos recorrido todas las vecindades en el que cada iteración realizaremos una búsqueda para encontrar el óptimo local y evaluaremos si este óptimo local es mejor que la solución inicial (y mejor) que teníamos. Si es mejor, pasará a ser la solución a partir de la cual volveremos a realizar una búsqueda local, y si no, pasaremos al siguiente entorno a explorar.

**Algorithm 1** VNDS

```
1: Determinar el conjunto de vecindades N_k, k = 1, ..., k_max
2: Seleccionar solución inicial x.
3: Elegir k = 1
4: repeat
5:    Realizar una búsqueda para encontrar el óptimo local partiendo de x como
      solución inicial.
6:    if x' es mejor solución que x then
7:       x = x'
8:       CONTINUE
9:    else
10:      k = k + 1
11:   end if
12: until K = Kmax
```

Usualmente se desarrolla un algoritmo de búsquedas anidadas unas a otras, esto quiere decir que si tienes 4 tipos de vecindad, puedes determinar que una vez que encuentres una solución mejor que la actual en el tercer o cuarto tipo de vecindad según tu orden determinado, puedes, partiendo de ese punto, empezar a buscar usando el primer tipo de vecindad ($k = 1$).

Este método pudiera ser el más adecuado en aquellos casos donde se tiene un gran conocimiento sobre el problema, teniendo a un experto que dirija y ordene los distintos entornos consiguiendo una gran explotación.

### 2.1.2. Búsqueda reducida en entorno variable

Llamada RVNS por sus siglas en inglés *Reduced Variable Neighboshood Search*, descrito en Hansen et al. (2001) propone una aproximación de búsqueda en entorno variable en la que se simplifica el paso de la búsqueda local, que a menudo es el paso más costoso, eligiendo un punto $x'$ al azar y comparándolo con el $x$ actual. Podemos observar su esquema en Algorithm 2.

**Algorithm 2** RVNS

```
1: Seleccionar el conjunto de vecindades N_k, k = 1, ..., k_max
2: Seleccionar solución inicial x
3: Elegir k = 1
4: repeat
5:    Generar un punto x' aleatorio de entre las posibles soluciones contenidas en
      el k − enesima vecindad de x
6:    if x' es mejor solución que x then
7:       x = x'
8:       CONTINUE
9:    else
10:      k = k + 1
11:   end if
12: until K = Kmax
```

En Algortihm 2 se puede observar cómo se determinan los distintos entornos en la línea 1, posteriormente se selecciona una solución inicial y comenzamos un bucle que finalizará cuando hayamos recorrido todos los entornos definidos. En cada iteración podemos observar cómo se selecciona un punto aleatorio en la vecindad de la mejor solución encontrada hasta el momento y se comprueba si es mejor solución que ésta. Si lo es, esta nueva solución pasará a ser la mejor solución y a partir de la cuál elegiremos otro punto aleatorio de la vecindad de esta última. Si no es mejor, se procederá de la misma manera con la siguiente vecindad.

La metaheurística RVNS es ampliamente utilizada para encontrar soluciones iniciales en problemas donde no se tiene un gran dominio y ha de prevalecer la exploratoriedad.

El Algorithm 3 hace referencia a una implementación simple de VNS completo descrito en Mladenović and Hansen (1997). Como podemos observar, la versión básica de este algoritmo destaca por su sencillez al no conllevar complejos cálculos y se podría definir como la unión de los métodos VND y RVNS

**Algorithm 3** VNS simple

1: Seleccionar el conjunto de vecindades $N_k, k = 1, ..., k_{max}$
2: Seleccionar solución inicial $x$
3: Elegir $k = 1$
4: **repeat**
5: Generar un punto $x'$ aleatorio de entre las posibles soluciones contenidas en el $k - enesima$ vecindad de $x$
6: Aplicar un método de búsqueda local partiendo de $x'$ como solución inicial
7: Definir como $x''$ al óptimo local
8: **if** $x''$ es mejor solución que $x$ **then**
9: $x = x''$
10: CONTINUE
11: **else**
12: $k = k + 1$
13: **end if**
14: **until** $K = Kmax$

anteriormente mencionados, incluyendo en el mismo algoritmo los pasos de exploración (Paso 5) mediante el cual intentará no caer en ciclos de operaciones repetitivas, y el de explotación (Paso 6) a través del cual pretenderemos explotar las soluciones cercanas a $x'$ y excoger la mejor de ellas. Para este último paso de búsqueda local, quizás el más elaborado de todos, existe multitud de bibliografía. Algunos buenos puntos de partida para su estudio pueden ser los indicados en Korupolu et al. (2000) y Lourenço et al. (2003).

Los criterios de parada habituales son el número máximo de iteraciones, tiempo máximo de CPU permitido, o máximo número de iteraciones sin ninguna mejora.

### 2.1.3. Variantes de la búsqueda en entorno variable

#### Búsqueda en entorno variable con descomposición

Esta variante del VNS propone descomponer el problema en dos fases usando como método de búsqueda local otra heurística una vez escogido el punto $x'$ aleatorio. Este método se describe en Hansen et al. (2001) obteniendo unos grandes resultados aplicados al problema P-Mediano implementando un VNS de dos niveles, donde el algoritmo de búsqueda local es otro VNS. Podemos ver el esquema genérico en el Algorithm 4, donde en la línea 1 determinamos las vecindades por las que se moverá nuestro algoritmo, posteriormente elegiremos una solución inicial y entraremos en un bucle que parará cuando hayamos recorrido todas las vecindades disponibles. En cada iteración el algoritmo seleccionará un punto aleatorio de la vecindad del mejor punto encontrado hasta el momento (o punto inicial si todavía no se ha encontrado uno mejor) y a partir de éste se empezará a buscar a través de los vecinos de la mejor solución un

óptimo local mediante una heurística. Si el óptimo local es mejor solución que la mejor solución hasta el momento, este punto pasará a ser la mejor solución, si no, procederemos a buscar en el siguiente vecindario.

**Algorithm 4** VNDS

1: Seleccionar el conjunto de vecindades $N_k, k = 1, ..., k_{max}$
2: Seleccionar solución inicial $x$
3: Elegir $k = 1$
4: **repeat**
5: Generar un punto $x'$ aleatorio de entre las posibles soluciones contenidas en el k-enésima vecindad de $x$
6: Aplicar una heurística para encontrar el óptimo local partiendo de $x'$ como solución inicial
7: Definir como $x''$ al óptimo local
8: **if** $x''$ es mejor solución que $x$ **then**
9: $x = x''$
10: CONTINUE
11: **else**
12: $k = k + 1$
13: **end if**
14: **until** $K = Kmax$

### Búsqueda en entorno variable sesgada

Una vez encontrada una solución buena tras una buena explotación, generalmente hará faltar moverse lejos de este mejor vecino para intentar buscar una solución mejor. La modelización de esta nueva búsqueda, lejos del óptimo local suele ser muy parecida a las heurísticas multicomienzo, es decir, en las cuales se escogen varios puntos iniciales para arrancar la heurística, pero este método suele ser demasiado costoso en cuanto a eficiencia se refiere, por eso, este algoritmo descrito en Hansen et al. (2001) elegirá moverse hacia el mejor vecino dependiendo de una función de distancia $\rho$ entre $x'$ y $x''$ multiplicada por un valor parametrizable $\alpha$. De esta manera, si no nos movemos, lo que haremos será rotar de vecindad. Con esta medida se pretende fomentar la exploración cuando la solución encontrada no sea mucho mejor que la actual. En el Algorithm 5 podemos observar como en la primera línea se determina el conjunto de las vecindades, en la segunda se determina la solución inicial y se establece como la óptima hasta el momento. También se establece el valor $f(x)$ como óptimo, y se determina el parámetro $\alpha$. Posteriormente entraremos en un bucle hasta que hayamos recorrido todos los entornos en el que en cada iteración elegiremos un vecino aleatorio del $x$ actual, el cual nos servirá como punto de partida para la búsqueda local. Si el óptimo local encontrado es mejor que la mejor solución que teníamos, se sustituye la óptima por la actual, si la solución

no decrementa en una cantidad $\alpha$ por la función de distancia $\rho$ entre los dos puntos, entonces también la tomamos como buena y se actualiza la solución actual a la encontrada en la búsqueda local. Si ninguno de estos dos casos se cumple, se pasa a al siguiente vecindad.

**Algorithm 5** SVNS

1: Seleccionar el conjunto de vecindades $N_k, k = 1, ..., k_{max}$.
2: Seleccionar solución inicial $x$. Determinar $x$ como $x_{opt}$ y $f(x)$ como $f_{opt}$. Elegir un parámetro $\alpha$.
3: Elegir $k = 1$
4: **repeat**
5: Generar un punto $x'$ aleatorio de entre las posibles soluciones contenidas en el k-enésima vecindad de $x$
6: Aplicar una algoritmo de búsqueda local para encontrar el óptimo local partiendo de $x'$ como solución inicial
7: Definir como $x''$ al óptimo local
8: **if** $f(x'')$ es mejor solución que $f_{opt}$ **then**
9: $x_{opt} = x''$
10: **end if**
11: **if** $f(x'') - \alpha\rho(x.x')$ es peor solución que $f(x)$ **then**
12: $x = x''$
13: $k = 1$
14: **else**
15: $k = k + 1$
16: **end if**
17: **until** $K = Kmax$

### Búsquedas por Entornos Variables Paralelas

En inglés llamadas *Parallel Variable Neighbourhood Search*, *PVNS*, descrita en Moreno and Mladenović, es una variante que pretende paralelizar la etapa de búsqueda local de VNS obteniendo finalmente varias soluciones resultantes de empezar en distintos puntos iniciales la búsqueda y comparándolas al final. Otra variante incluiría el hecho de que cada búsqueda local va actualizando una variable que almacena la mejor solución y es común a todas las ramas, por lo que cada rama compararía con la mejor global.

# 3. Descripción del problema

En este capítulo se realizará una breve introducción a los problemas de asignación de tareas y entraremos en detalle con el caso real a resolver en este TFM. Se definirán conceptos usados durante las siguientes secciones así como las distintas restricciones del problema.

## 3.1. Problemas de asignación de tareas

El problema clásico de asignación de tareas o *timetabling*, tal y como podemos encontrar en Hertz (1991), modelaba la generación de los horarios de clase de un colegio donde se debían cumplir ciertas restricciones que incluían tanto a profesores como a alumnos. Este tipo de restricciones podían ser globales, como que dos profesores no pudieran impartir clases en el mismo aula a la misma hora, restricciones de emplazamiento (aulas específicas para ciertas asignaturas o aulas genéricas) o restricciones específicas de los alumnos y profesores. Existen competiciones que se basan en la generación de horarios dado un problema, unas restricciones y unos datos de entrada concretos como la *International Timetabling Competition* of Twente (2011).

Además, los problemas de asignación de tareas han sido aplicados a diferentes campos tales como secuenciación de tareas en fábricas, planificación de robots, planificación logística... Estos tipos de problema tienen en común el método de modelización. En todos ellos existen unas tareas a realizar, que a menudo incluyen distintas subtareas, que se han de distribuir a lo largo de un conjunto finito de máquinas cumpliendo con ciertas restricciones. Una de las primeras referencias que podemos encontrar en la bibliografía acerca de este tema es Muth and Thompson (1963). Para la mayoría de estos problemas, no existe un algoritmo conocido que los resuelva en tiempo polinómico, y es aquí cuando recurrimos a las heurísticas y metaheurísticas.

En Colorni et al. (1998) se estudia el uso de tres metaheurísticas (Recocido Simulado, Búsqueda Tabú y Algoritmos Genéticos) aplicadas al problema clásico de resolución de horarios en un instituto. Además, en Lewis (2008) podemos encontrar un estudio del funcionamiento de distintas metaheurísticas para problemas de *timetabling*.

Como veremos durante los capítulos siguientes de este trabajo, en nuestro problema sólo habrá tres tipos de tareas, trabajar como controlador ejecutivo, trabajar como controlador planificador, o descansar, y estas tareas se distribuirán entre los distintos controladores disponibles, de manera que se optimice cierta función objetivo.

## 3.2. Descripción general del problema

El objetivo de este trabajo es la asignación de un conjunto de controladores aéreos con distintas cualificaciones y habilitaciones, tanto a posiciones de ejecutivo como de planificador, a lo largo de una serie de sectores aéreos abiertos durante un periodo determinado de tiempo, que habitualmente será un turno de trabajo. La dificultad de este objetivo se basa en la complejidad y severidad de las restricciones que acotan en gran medida, la forma que ha de tener una solución factible. Una vez encontrada la primera solución factible, intentaremos maximizar una función multiobjetivo que consta de cuatro funciones objetivo, que podrán ser ponderadas como se deseen. Estas funciones se podrían resumir en:

1. Las condiciones de elección de soluciones.
   (a) Tiempo óptimo de trabajo en la misma posición y sector.
   (b) Tiempo óptimo de trabajo entre descansos.
   (c) Porcentaje de tiempo que un controlador trabaja en posiciones ejecutivas.
2. Mantener una estructura similar a los estadillos/plantillas que actualmente se están utilizando, facilitando el entendimiento de la solución proporcionada.
3. Minimizar el número de cambios en 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.

### 3.2.1. Definición de conceptos generales del problema

**Turno de trabajo:**

Existen tres tipos de turnos de trabajo: Turnos de mañana, de tarde y de noche. Cuando un controlador deba cubrir solamente uno de los turnos mencionado, diremos que es un turno corto, mientras que si un controlador debe cubrir uno de los turnos de mañana o de tarde y además una parte del turno de noche, diremos que está realizando un turno largo. Cada uno de éstos tiene su propias restricciones. Las horas de inicio de los turnos son variables y vienen definidas en la entrada del problema. En la Figura 1 se puede observar las distintas configuraciones que puede tener un turno. Por ejemplo, el turno de mañana corto abarca desde las 06:20 a.m. hasta las 14:00 p.m., mientras que el turno de mañana largo comienza a las 05:40 a.m. y termina a las 14:00 p.m. por lo que se ampliaría en 40 minutos. El turno de tarde corto abarcará

desde las 14:00 p.m. hasta las 21:20 P.M mientras que el turno de tarde largo empezará a la misma hora y finalizará a las 22:20 p.m. El turno de noche será único y comenzará a las 21:20 p.m. y finalizará a las 06:20 a.m.

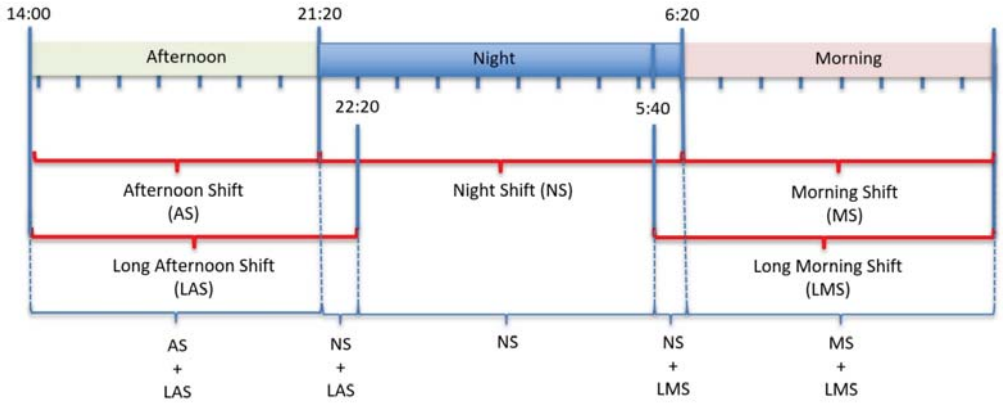



**Fig. 1:** Turnos de trabajo

**Espacio aéreo:**

El espacio aéreo se compone de diversos sectores que a su vez, están compuestos por volúmenes, los cuales podemos imaginar como el componente básico del espacio aéreo. Además, existe el concepto de núcleo, lo cual no es más que un conjunto de sectores en los cuales puede operar un determinado controlador.

**Afinidad:**

Dos sectores serán afines si existe algún volumen que esté presente en ambos sectores, es decir, que dos sectores compartan al menos un volumen. La afinidad entre sectores es un parámetro de entrada importante puesto que podrá relajar algunas restricciones. La matriz de afinidad es uno de los datos de entrada al problema.

**Sectorizacion:**

Definiremos como sectorización a la configuración de los sectores y en que periodo está abierto cada uno. Los sectores pueden ser de dos tipos, sectores de ruta y aproximación (PTD) y sectores de sólo ruta (CON). Estos datos también son una variable de entrada al problema. En la Tabla 1 podemos observar un ejemplo de sectorización.

**Tab. 1:** Ejemplo de sectorización

| Sectorización | | | |
|---|---|---|---|
| **Configuración** | **Volumenes** | **Inicio** | **Fin** |
| 3A | GTL, LVL, LVU, LAG... | 5:20:00 | 6:05:00 |
| 4A | VVS, MMS, CCL, CCU... | 6:05:00 | 7:50:00 |
| 5A | VVS, MMS, GTL, LAG... | 12:05:00 | 13:05:00 |
| ... | ... | ... | ... |

**Controladores:**

Los controladores se encargan de operar en los distintos sectores, y tal y como hemos descrito antes, el núcleo define en qué sectores podrá operar cierto controlador. Además, un controlador tampoco podrá operar en los sectores que excedan su capacitación. Los dos tipos de capacitación posible para los controladores coincide con los tipos de sectores, es decir, pueden tener capacitación "PTD" o "CON".

### 3.2.2. Restricciones del problema

Las restricciones del problema en cuanto a la asignación de controladores a sectores se refiere, vienen descritas a continuación precedida de un identificador, ya utilizado en Tello et al. (2018), a través del cual nos referiremos a ellas en secciones posteriores:

1. LC1: Una determinada posición podrá ser asignada a un controlador si el controlador está habilitado en el núcleo al que pertenece, 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.
2. LC2: A un controlador tipo CON podrá asignársele una posición cuyo sector sea Ruta.
3. LC3: Porcentaje de tiempo de descanso mínimo en turno diurno (mañana y tarde), incluyendo turnos largos es del 25 % mientras que el porcentaje de tiempo de descanso mínimo en el turno de noche será del 33 %
4. LC4: 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. Nota: puede existir más de un sector o agrupación de sectores que se deban cubrir con 4 controladores.
5. LC5: 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.
6. LC6: 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.

7. LC7: 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.
8. LC8: 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.
9. LC9: El tiempo mínimo de trabajo continuado, es decir, el tiempo de trabajo de un controlador entre dos descansos es de 15 minutos.
10. LC10: El tiempo mínimo de descanso de un controlador es de 15 minutos.
11. LC11: El tiempo mínimo de un controlador en la misma posición es de 15 minutos.
12. LC12: Número máximo de sectores por los que rota un controlador en un mismo turno: 3 (CRIDA dispone de un algoritmo que calcula el número de sectores por los que pasa un controlador, teniendo en cuenta las posibles afinidades).
13. LC13: Cada turno de trabajo debe tener al menos un controlador asignado y cada controlador debe ser responsable de un turno.
14. LC14: Cada controlador debe trabajar al menos 15 minutos, no puede descansar todo el turno.

Estas restricciones son estrictas, y se han de cumplir todas y cada una de ellas para que una solución sea factible. A éstas se le añaden las siguientes restricciones más flexibles que tienen como objetivo el ayudar a que el algoritmo converja a una solución más adecuada.

- Tiempo óptimo de trabajo en posición (ejecutivo o planificador). Tiempo en el que un controlador se encuentra sin cambio de sector y tipo de control (Ej. Planificador: 45 minutos).
- Tiempo óptimo de trabajo entre descansos: 90 minutos.
- 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).

Todas estas restricciones son en realidad los sub-objetivos que componen el primer objetivo de la Fase 3.

# 4. Método de resolución del problema

Este trabajo se enfocará en resolver la Fase 2 y la Fase 3 de la metodología descrita posteriormente en la Sección 4.2, generando un algoritmo basado en VNS de manera que pueda ser aplicable para las dos fases. La primera tarea será especificar como se modelizarán y evaluarán las distintas soluciones con las que se vaya trabajando.

## 4.1. Modelización de las soluciones

Las soluciones se modelarán de una manera bastante intuitiva mediante una matriz, donde las columnas nos reflejarán las distintas capturas de la situación de los controladores en cierto periodo de tiempo. Cada columna equivaldrá a un periodo de tiempo mínimo que a lo largo del trabajo denominaremos "*slot*" y vendrá definido por los parámetros de entrada al problema. Las filas de la matriz equivaldrán a la distribución de carga de trabajo de cada controlador. De forma más concreta podemos decir que que en cada fila habrá un controlador, y las columnas marcarán el horario. En la Figura 2 podemos observar un fragmento de una solución.

El código representado en cada celda hace referencia a la sectorización que esta cubriendo cierto controlador en ese slot de tiempo concreto.

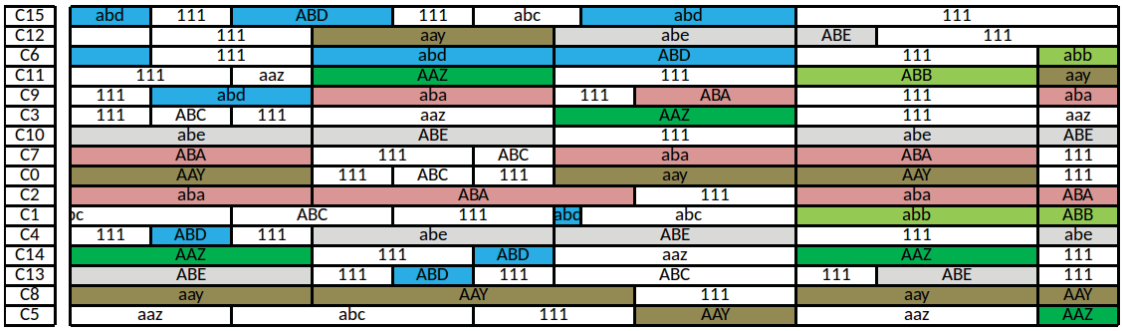



**Fig. 2:** Ejemplo de una solución.

Además, se poseerá tantas estructuras de tipo controlador como filas tenga la matriz solución. Dicha estructura poseerá la siguiente información:

- **ID**: Identificador de cada controlador.
- **Turno**: Turno de trabajo en el que el controlador realiza su jornada (Mañana Largo, Tarde Corto, Noche...).
- **Núcleo**: Núcleo que el controlador tiene asignado para realizar su trabajo.
- **PTD**: Acreditación del controlador para controlar sectores de ruta y aproximación.

- **CON**: Acreditación del controlador para controlar solo sectores de ruta.
- **Turno Asignado**: Posición de la fila en la matriz de turnos que el controlador tiene asignada para realizar su jornada.
- **Turno Noche**: Si el controlador esta cubriendo un sector en un turno de noche, es posible que solo pueda cubrir este único sector, para indicar si esto se produce (lo cual depende del sector) se utiliza esta variable.
- **Imaginario**: Parámetro para marcar cuando se utilizan más controladores de los disponibles en las fases de pruebas.

## 4.2. Metodología

En la metodología propuesta se decidió afrontar el problema dividiéndolo en 3 fases, cada una con unos objetivos distintos, los cuales que se detallan a continuación.

### 4.2.1. Fase 1

En esta fase se pretende, mediante el uso de plantillas y a través de la modificación de los tiempos de descanso de los controladores, generar una serie de soluciones iniciales. La heurística propuesta para esta fase se basa en el uso de una plantilla optimizada como la que se presenta en la Figura 3, que representa un turno de 8 horas (96 slots) con periodos de descanso de 6 slots donde "R" representa descanso y "W" representa trabajo, diferenciando los dos tipos de trabajo (ejecutivo y planificador) mediante las mayúsculas y minúsculas. Como se puede observar, en la figura se repite un patrón. Los periodos de trabajo (tando "'w" como "W") duran siempre el doble que los periodos de descanso ("R").

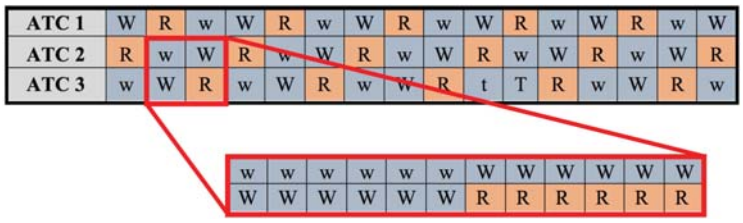



**Fig. 3:** Ejemplo de una solución.

Como veremos en las secciones posteriores, la modelización de las soluciones se ha diseñado de manera que el "slot" más pequeño que se contempla actualmente es de 5 minutos, por lo que si los periodos de descanso, tal y como dicen las restricciones, deben durar al menos quince minutos, y el máximo periodo de trabajo continuo debe durar como máximo dos horas. Esto equivaldría a que los periodos de descanso solo podrían durar entre tres y doce slots, por lo

que se generarán 10 soluciones iniciales, cada una con una longitud del periodo de descanso (y por lo tanto, de trabajo) distinta.

La heurística de creación de soluciones iniciales desrita en Tello et al. (2018) tiene diversos pasos:

- *Paso 1.Añadir plantillas para cubrir la sectorización aérea.* Para cada sector abierto $i \in \{1, ..., s\}$ hacer
  1. Insertar una plantilla vacía en la matriz solución.
  2. Para cada slot de tiempo $j \in \{1, ..., nCol\}$ hacer:
     *a*) Si el sector $i$ está abierto durante el slot $j$, entonces introducir el slot correspondiente a la plantilla en la posición $j$ de la matriz solución con el nombre del sector $j$.
     *b*) Si el sector $i$ no está abierto durante el slot $j$, entonces introducir a la matriz solución los bloques de descanso en la posición $j$.

Al final del Paso 1, se le habrán asignado a todos los sectores una pareja de controladores (ejecutivo y planificador). A continuación, definimos *nRow* como el número de filas de nuestra matriz solución y procedemos con el Paso 2.

- *Paso 2. Reparar la solución para mejorar la factibilidad.* Para cada línea $l \in \{1, ..., nRows\}$ en la matriz solución que no tiene el periodo mínimo de trabajo, tratamos de asignar ese periodo de trabajo que dura menos que el mínimo a otro controlador. Para esto:
  1. Intentamos transferir un periodo de trabajo por debajo del mínimo de un controlador a otro:
     *a*) Comprobar si hay otro controlador trabajando en el mismo sector y que está a punto de empezar a descansar justo en el instante donde empieza el periodo de trabajo a ser transferido y que la longitud del periodo de descanso es mayor al periodo de trabajo en cuestión. Si es así, además se deberán cumplir las siguientes restriccioens antes de transpasar la carga de trabajo:
        1) Todos los controladores deberán descansar 30 minutos cada dos horas.
        2) El periodo de trabajo máximo es de 2 horas.
        3) El periodo mínimo de descanso continuo es de 15 minutos.
        4) El periodo mínimo de trabajo continuo es de 15 minutos.
     *b*) Comprobar si hay algún otro controlador que fuese a empezar a trabajar en el mismo sector justo en el instante en el que el periodo a transferir termina y tuviera un periodo de descanso mayor al tiempo de trabajo a transpasar. Si es así, comprobar las mismas condiciones que se cumplen en (a).
  2. Intentar extender un periodo de trabajo sin la longitud mínima usando un periodo de trabajo de otro controlador:

*a*) Comprobar si hay otro controlador trabajando en el mismo sector en el cuál el periodo de trabajo empezó y tiene suficientes slots para completar el tiempo de trabajo mínimo. Antes de hacer el cambio, comprobar que el mismo no empeora la factibilidad.

*b*) Comprobar si hay otro controlador el cuál empezó a trabajar en el mismo sector justo después del periodo de trabajo que queremos expandir y tiene suficiente tiempo de trabajo inmediatamente después para completar el mínimo tiempo de trabajo.

*c*) Comprobar la lista de restricciones que los controladores deben cumplir.

- *Paso 3. Colocación de los recursos disponibles.*

  Para cada línea $l \in \{1, ..., nATC\}$, de la matriz solución, hacer:

  1. Si la línea $l$ contiene sectores de aproximación durante un periodo largo, entonces buscamos un controlador con acreditación PTD asociado al turno largo que pueda operar en ese núcleo. Este controlador será asignado a la línea $l$.
  2. Si la línea $l$ contiene sectores de aproximación durante un turno corto, entonces buscamos un controlador con acreditación PTD asociado con el turno corto que pueda operar en ese núcleo y le asignamos esa línea.
  3. Si la línea $l$ contiene sectores de tipo ruta durante un turno largo, entonces buscamos un controlador disponible con acreditación CON asociado con el turno largo y que además pueda operar en ese núcleo. Si existe, se lo asignamos a la línea $l$, si no, buscamos a un controlador disponible con acreditación PTD asociado al turno largo que pueda operar en ese núcleo y le asignamos la línea.
  4. Si la línea $l$ contiene sectores de tipo ruta durante el turno corto, buscamos un controlador disponible con acreditación CON asociado con el turno corto. y que además pueda operar en ese núcleo. Si existe, se lo asignamos a la línea $l$, si no, buscamos a un controlador disponible con acreditación PTD asociado al turno largo que pueda operar en ese núcleo y le asignamos la línea.

  En este punto, los controladores pueden no estar asignados por no haber cumplido las condiciones por línea anteriores, por lo que se asignan de la siguiente manera:

  Comprobar si el controlador $c \in \{1, ..., nATC\}$, está asignado a una línea de la matriz solución:

  Si el controlador $c$ no ha sido asignado y es un controlador CON (PTD) entonces buscamos una línea que incluya sectores de ruta (y aproximación).

  1. Si hay una línea disponible que incluya sectores de ruta y aproximación se le asignará al controlador $c$.

2. Si no, se le asignará al controlador $c$ una línea disponible aleatoria.

Una vez que hayan concluido estas tres fases, puede ser que haya líneas a las que no se le haya asignado controlador. Para solucionar esto se crean unos controladores artificiales que serán eliminados en la Fase 2. En la Fase 1, la matriz solución tendrá habitualmente más líneas que controladores. El resultado de esta fase puede dar lugar a dos casos:

1. Si no existe ninguna solución factible entre las generadas, se irá a la Fase 2.
2. Si existe alguna solución factible entre las generadas, se pasa directamente a la Fase 3.

### 4.2.2. Fase 2

En esta fase se pretende obtener soluciones factibles a partir de soluciones infactibles. Para esta fase se utiliza una estrategia que a cada etapa reordena la matriz solución en orden ascendente de carga de trabajo y utiliza una función objetivo consistente en la agregación de dos términos $f_1$ y $f_2$:

$$max \;\; f = \begin{cases} w_1 * f_1 + w_2 * f_2, & \text{si } \; nATC > nATCaval \\ f_2, & \text{si } \; nATC \leq nATCaval \end{cases}$$

donde $w_1$ y $w_2$ representan la importancia relativa de los objetivos, $nATC$ en el número de controladores usados en la solución actual y $nATCaval$ es el número de controladores reales disponibles.

Además, $f_1$ es la componente que está enfocada a reducir el número de controladores:

$$f_1 = \frac{1}{nATC^2} \sum_{i=1}^{nATC} ih_i,$$

donde $h_i$ es el número de slots que el controlador $i$-ésimo está trabajando.

El término $\frac{1}{nATC^2}$ implica que cuando el número de controladores baja, mejora la función objetivo. Para un número dado de controladores, los valores de la función objetivo, son mayores para soluciones donde la carga de trabajo se concentra en las partes inferiores de la matriz de solución.

De esta a es fácil ver que los algoritmos que estén dirigidos por esta función objetivo intentarán amontonar la carga de trabajo en la parte inferior de la matriz, esto ayudará a ir eliminando controladores una vez se queden sin nada de carga de trabajo.

Con el objetivo de utilizar bien el sistema de ponderaciones es necesario normalizar sus valores:

$$f_1 = 1 - \frac{f_{1,max} - f}{f_{1,max} - f_{1,min}},$$

donde

$$f_{1,max} = \sum_{k=nATC-nATC^*}^{nATC} \frac{k \times nSlot}{nATC^2},$$

$$f_{1,min} = \sum_{k=1}^{nATC^*} \frac{k \times nSlot}{nATC^2},$$

y $nATC^*$ es el número de controladores necesarios para cubrir la sectorización sin descanso y $nSlot$ es el número de slots considerados en el turno.

Además, $f_2$ es la componente que cuantifica el número de restricciones incumplidas. Para normalizar este término:

$$f_2 = \frac{f_{2,max} - f_2}{f_{2,max}},$$

donde $f_{2,max}$ es el número máximo de veces que las restricciones pueden ser incumplidas. Su valor puede variar dependiendo del turno que estemos optimizando. Si estuvieramos optimizando un turno de noche, entonces

$$\begin{aligned} f_{2,max} &= 8 \times nATC + 2 \times 2nATC + \\ &\quad 4(nATC + nATC \times nSlot \times \frac{1}{nSlot}) \\ &= 12 \times nATC + 4(nATC + nATC \times \frac{nSlot}{nSlot}) \\ &= 20 \times nATC, \end{aligned}$$

donde hay 8 restricciones (LC3, LC5, LC6, LC9, LC10, LC11, LC12 y LC14) no cumplidas por $nATC$ controladores, dos restricciones (LC4 y LC13) no cumplidas por $2 \times nATC$ y cuatro restricciones (LC1, LC2, LC7 y LC8) no cumplidas por $nATC$ controladores, donde cada grado de incumplimiento se contabiliza y es multiplicado por $\frac{1}{nSlot}$.

En cambio, si estamos buscando la factibilidad de un turno de tarde o mañana, la restricción LC4 no se considera, por lo que

$$f_{2,max} = 10 \times nATC + 4(nATC + nATC).$$

Como resultado de esta fase existen dos casos posibles:

1. En caso de no poder encontrar ninguna solución factible, se indicará que no existen soluciones factibles.
2. En caso de encontrar una solución factible, se pasará a la Fase 3.

### 4.2.3. Fase 3

En esta fase procederemos a optimizar la solución factible hallada anteriormente mediante el uso de algún algoritmo MIR (*Multiple Independent Run*) basado, en nuestro caso en VNS (Variable Neighbourhood Search), pero en otros trabajos se han usado algoritmos como Recocido Simulado, descrito en Tello et al. (2018), o Búsqueda Tabú, descrito en Suárez 2017.

En esta fase se pretende optimizar una función multiobjetivo con cuatro objetivos, como son:

1. Las **condiciones de elección de soluciones**:
   *a*) Tiempo óptimo de trabajo en posición, es decir, tiempo en el que un controlador se encuentra sin cambio de sector ni tipo de control (Ej. Planificador.: 45 minutos).
   Para cada línea de la matriz solución calculamos el número de slots cosnecutivos que un controlador está en un mismo sector y mismo puesto de trabajo, y realizamos la diferencia entre este valor y el óptimo, que es de 9 slots. Sumamos este valor en todos los periodos de trabajo de cada línea, y sumamos el valor de todas las líneas. Por último dividimos entre el número de controladores para hallar la media:
   $$f_{1,1} = \frac{\sum_{i=1}^{nATC} \sum_{j=1}^{n_i} |45 - l_{i,j}|}{nATC},$$
   donde $n_i$ es el número de periodos de trabajo donde el controlador $i-$ésimo se mantiene en la misma posición de trabajo y sector y $l_{ij}$ es su duración en minutos. Normalizando esta función objetivo:
   $$g_{1,1} = \frac{f_{1,1}^{max} - f_{1,1}}{f_{1,1}^{max}},$$
   donde $f_{1,1}^{max} = |45 - posMin| \times 8 \times \frac{nSlot}{30}$ y $posMin$ es el tiempo mínimo que un controlador puede permanecer en una posición sin inflingir ninguna restricción de trabajo y $\frac{nSlot}{30}$ es el número de veces que hay un turno de trabajo de dos horas seguido de un turno de descanso de 30 minutos en un mismo turno.
   *b*) Tiempo óptimo de trabajo entre descansos, en nuestro caso, 90 minutos.
   Calculamos para cada línea la longitud de cada periodo de trabajo en minutos y realizamos la diferencia con el periodo óptimo de trabajo, que son 90 minutos. Sumamos estas diferencias para cada línea y sumamos el valor de cada línea. Por último dividimos entre el número de controladores para realizar la media:
   $$f_{1,2} = \frac{\sum_{i=1}^{nATC} \sum_{j=1}^{m_i} |90 - r_{i,j}|}{nATC},$$

donde $m_i$ es el número de periodos de trabajo del controlador $i$ y $r_{ij}$ es su duración en minutos. Para normalizar:

$$g_{1,2} = \frac{f_{1,2}^{max} - f_{1,2}}{f_{1,2}^{max}},$$

donde $f_{1,2}^{max} = |90 - workMin| \times \frac{nSlot}{6}$ es el máximo valor de $f_{1,2}$, siendo $workMin$ la mínima duración del periodo de trabajo para un controlador (15 minutos). Dividimos el número de slots entre 6 porque la longitud del periodo mínimo de trabajo es de tres slots al igual que el de descanso.

*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).
Para ello, dividimos el tiempo total que un controlador está trabajando en una posición ejecutiva entre el tiempo total de trabajo de esa línea, obteniendo el porcentaje de tiempo que está como ejecutivo ($f_{1,3}$). Di no está entre el 40 % y 60 % calculamos la diferencia al valor más cercano (40 o 60 dependiendo de si está por encima o por debajo) y lo multiplicamos por 0.4 que corresponde con un 40 %, que es la máxima diferencia para normalizar. Porteriormente sumamos las diferencias de cada línea y dividimos por el número de líneas:

$$g_{1,3} = \frac{1}{0{,}4} \sum_{i=0}^{nATC} \begin{cases} \frac{0{,}4 - f_{1,3}}{nATC}, & \text{si } f_{1,3} < 0{,}4 \\ \frac{f_{1,3} - 0{,}6}{nATC}, & \text{si } f_{1,3} > 0{,}6. \end{cases}$$

Estos tres subobjetivos tendrán la misma importancia por lo que

$$g_1 = g_{1,1}/3 + g_{1,2}/3 + g_{1,3}/3.$$

2. Mantener una **estructura similar a los estadillos/plantillas** anteriormente utilizados, facilitando el entendimiento por el personal de CRIDA de la solución proporcionada y permitiendo cambios sencillos:

   Para esto se estableció que los periodos de trabajo y descanso para un mismo sector deberían estar tan agrupados como fuera posible. Esta premisa se formuló mediante la siguiente función objetivo:

$$f_2 = \sum_{i=1}^{nATC-1} \sum_{j=1} nSlot - 1 \begin{cases} 1, & \text{si } element_{i,j} = element_{i+1,j} \\ 1, & \text{si } element_{i,j} = element_{i,j+1} \end{cases}$$

   Esta función recorre todos los elementos de la matriz de solución comprobando si la posición a la derecha y la posición debajo tiene el mismo sector y clase de trabajo. Para normalizar:

$$g_2 = 1 - \frac{f_2^{max} - f_2}{f_2^{max}},$$

donde el valor teórico máximo es

$$f_2^{max} = (nSlot - 1) \times (nATC - 1) \times 2$$

3. Minimizar el **número de cambios en sala**:

   Para afrontar este objetivo será clave minimizar el número de descansos (no la duración). Para ello contaremos el número de descansos en cada línea y lo normalizaremos a través del valor mínimo que etá ligado al número de líneas puesto que todos los controladores deben tener al menos un periodo de descanso. Por lo que podremos decir que $f_{3,1}^{min} = nATC$ y

$$f_3^{max} = \frac{nSlot}{6} \times nATC,$$

   donde el número 6 refleja la suma de los periodos mínimos de trabajo y descanso. para normalizar:

$$g_3 = \frac{f_3^{max} - f_3}{f_3^{max} - f_3^{min}}.$$

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

   Este subobjetivo intentará equilibrar la carga de trabajo entre los distintos controladores a través de la desviación estándar de los periodos de trabajo de las distintas líneas de la matriz de soluciones.

   El máximo valor para la desviación estándar será igual a la media, por ejemplo, si tenemos una carga de trabajo de 200 slots, y tenemos cuatro controladores que pueden tarabajar como máximo 100 slots, la media será de 50 slots. Para maximizar la varianza 100 slots de detrabajo serán asignados a dos controladores mientras que los otros se quedarán totalmente libres. Esto supondrá una varianza de $50^2$, una desviación típica de 50, que encaja con el valor de la media. Entonces:

$$g_4 = \frac{\mu - f_4}{\mu},$$

   donde $\mu$ es la media y $f_4$ es la desviación estándar de la carga de trabajo de los controladores.

Estos cuatro subobjetivos serán ponderados mediante unos pesos obtenidos del método ROC (*Rank Order Centroid*) propuesto por CRIDA donde

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

por lo tanto

$$w_1 = \frac{1 + 1/2 + 1/3 + 1/4}{4} = 0{,}52,$$
$$w_2 = \frac{1/2 + 1/3 + 1/4}{4} = 0{,}27,$$
$$w_3 = \frac{1/3 + 1/4}{4} = 0{,}15,$$
$$w_4 = \frac{1/4}{4} = 0{,}06.$$

Toda la información acerca de las funciones objetivo ha sido extraida de Tello et al. (2018).

## 4.3. Parámetros de entrada del problema

A continuación se detallan los distintos parámetros de entrada que terminarán de definir el problema a resolver:

- **ListaSectoresElementales_<Unidad de Control>.csv**: Lista de todos los sectores pertenecientes a la unidad de control y los sectores elementales por los que están formados cada uno de los sectores. Nota: En caso que la columna de sectores elementales se encuentre vacía, quiere decir que el propio sector es un sector elemental.
- **MatrizAfinidad_<Unidad de Control>.csv**: Matriz de afinidad entre los sectores de una unidad de control. Si los sectores son afines, la matriz incluirá un 1 en la posición determinada, si no son afines un 0. Nota: En la diagonal de la matriz (mismo sector columna-fila) siempre se pondrá un 1.
- **SectoresNucleos_<Unidad de Control>.csv**: Lista de los sectores pertenecientes a la unidad de control, en la que nos indica si el sector es PDT o Ruta y a los núcleos a los que pertenece. Nota: Un sector siempre tiene que ser de tipo PDT o Ruta (CON), y pertenecer a un núcleo como mínimo.
- **SectorizacionesSectoresVolumenes_<Unidad de Control>.csv**: Lista de todas las configuraciones de sectores para cada uno de los núcleos existentes en la unidad de control.
- **AperturaSectorizaciones_<Identificador-dd-mm-aaaa>.csv**: Contiene la sectorización de todos los núcleos existentes durante un turno. Cada fila hace referencia a un intervalo de tiempo dentro de un turno en el que está abierto un sector o configuración de sectores determinada. Además, nos indica si es sector o sectores son nocturnos, es decir, se tiene que cubrir con 4 controladores, con el número de la columna "#SECTORNOCTURNO" nos indica la agrupación de sectores que se

cubre con los 4 controladores. Por último, la sectorización entrante debe cumplir las siguientes restricciones: Como mínimo se debe mantener la misma configuración de sectores (listado de sectores abiertos representados en la sectorización por un intervalo de tiempo determinado) durante 20 minutos. El problema tiene una ventana de discretización de tiempo, y la entrada del problema se debe adaptar a esta discretización.

- **RecursosDisponibles_<Identificador-dd-mm-aaaa>.csv**: Muestra el número de controladores disponibles para cubrir la sectorización del turno. Nota: Los controladores están divididos por núcleo, turno (largo y corto) y acreditación de sector (CON y PTD).
- **Turno_<Identificador-dd-mm-aaaa>.csv**: Muestra el inicio y fin del turno de mañana, tarde o noche (incluyendo corto y largo); para cada uno de los núcleos existentes.

## 4.4. Implementación de la metaheurística VNS

El diseño de la metaheurística (recuérdese el esquema visto en el Algorithm 3) se puede dividir en diversas etapas. Una primera etapa tendría como objetivo definir cuales son los tipos de vecindarios (a lo largo de este trabajo usaremos indistintamente los términos "entorno" y "vecindario" con el mismo significado) con los que vamos a trabajar. Una segunda etapa se vería envuelta en la discusión y razonamiento de en qué orden se recorrerán los entornos, de cara a que sea beneficioso para conseguir los objetivos. Por último, una tercera etapa consistiría en la definición del método que se va a usar como búsqueda local. A continuación procedemos a explicar detalladamente cada una de las 3 etapas.

### 4.4.1. Definición de los entornos para la Fase 2

Como ya hemos explicado con anterioridad, la distinción que permite a esta metaheurística escapar de los óptimos locales es la variación del tipo de entorno que utiliza y por el cual se van moviendo las soluciones candidatas, por lo que la definición de los distintos entornos es de alguna manera vital y marcará los resultados que pueda obtener el algoritmo.

Antes de proceder a especificar cada uno de los entornos, es conveniente definir el término "tamaño del bloque elemental". Este término hace referencia al número de slots con el que trabajaremos en cada momento dentro de cada tipo de vecindad y nos permitirá, variando el tamaño de este bloque elemental, conseguir muchas más vecindades sin cambiar la lógica del programa.

Para cada entorno, si no encontramos soluciones mejores iremos aumentando el número de slots que manejamos hasta que lleguemos al tope, y sólo entonces cambiaremos de vecindad. El tamaño mínimo y máximo de slots viene definido

por el tiempo mínimo y máximo que puede trabajar seguido un controlador, habitualmente entre quince minutos y una hora y cuarto.

Como entrada a la Fase 2 tendremos una matriz de turnos normalmente con más filas que controladores disponibles (así se ha definido la salida de la Fase 1), y estas filas las iremos ordenando de manera piramidal en cuanto a carga de trabajo, es decir, las filas con menos carga de trabajo las situaremos en las filas superiores, de manera que podremos trabajar de manera que el objetivo sea liberar filas completamente de trabajo y eso según nuestro modelado, será transmitir trabajo desde lo alto de la matriz hacia abajo intercambiando bloques elementales. Esto se puede ver reflejado en la Figura 4 que muestra menos carga de trabajo en las filas superiores y más en las inferiores.

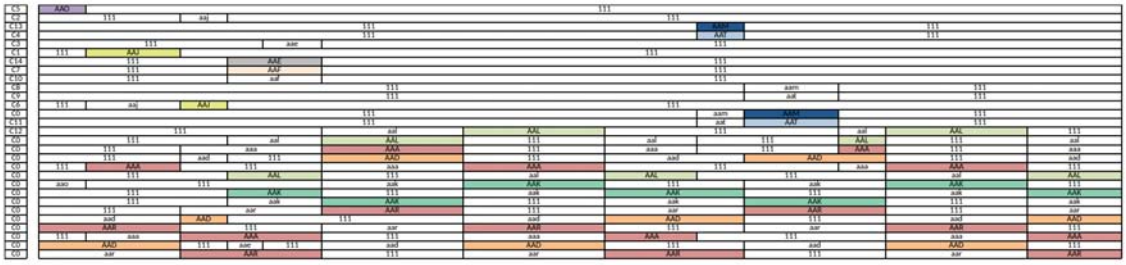


**Fig. 4:** Ejemplo distribución de carga piramidal

### 4.4.2. Entorno de tipo 1

El primer tipo de entorno buscará mover cargas de trabajo de arriba a abajo en la matriz, sustituyéndolas por otras que estén totalmente libres de trabajo. Es decir, dada una solución $s$, se considera vecindad inmediata a las soluciones resultantes de intercambiar dos bloques elementales llamados B1 y B2 de manera que se cumplan las siguientes condiciones:

1. B1 debe tener algo de carga de trabajo.
2. B2 debe estar libre de carga de trabajo.
3. B1 en la posición de B2 deberá prolongar la secuencia inmediatamente anterior o posterior. Es decir, el cambio prolongará una carga de trabajo.
4. B2 inicialmente siempre deberá estar en una fila inferior a B1 en la matriz de soluciones.

En la Figura 5 podemos ver un ejemplo donde el bloque B1 "aav" se va intentar mover al bloque B2 "111"(que significa que no tiene carga de trabajo) y lo consigue al cumplir que B1 tiene carga de trabajo, B2 está libre de carga de trabajo, el cambio alarga una secuencia de trabajo tanto por detrás como por delante (aunque solo una de las dos condiciones es necesaria) y que el bloque B2 por el que se va a intercambiar está en una fila inferior.


**Fig. 5:** Ejemplo de soluciones pertenecientes al entorno de tipo 1

### 4.4.3. Entorno de tipo 2

El segundo tipo entorno buscará mover cargas de trabajo de arriba a abajo en la matriz, pero en este caso en vez de intercambiar un bloque B1 con carga de trabajo por otro bloque B2 sin nada de trabajo, se buscará que B2 si tenga carga de trabajo. Las condiciones son las siguientes:

1. B1 debe tener algo de carga de trabajo.
2. B2 debe tener algo de carga de trabajo de trabajo.
3. B1 en la posición de B2 deberá prolongar la secuencia inmediatamente anterior o posterior. Es decir, el cambio prolongará una carga de trabajo.
4. B2 inicialmente siempre deberá estar en una fila inferior a B1 en la matriz de solución.

En la Figura 6 podemos observar un ejemplo en el cual el bloque B1 "aav" puede desplazarse hacia el bloque B2 "ABD" puesto que B1 tiene algo de carga de trabajo, B2 también, B1 alarga una secuencia de trabajo en B2 por delante o por detrás y B2 está en una fila inferior a B1,


**Fig. 6:** Ejemplo de soluciones pertenecientes al entorno de tipo 2

### 4.4.4. Entorno de tipo 3

El tercer tipo entorno es similar al primero pero relajando la restricción que hace que deba concatenarse con bloques del mismo tipo por delante o por detrás. Para no tener entornos solapados negaremos esta condición. Las condiciones quedarían de la siguiente forma:

1. B1 debe tener algo de carga de trabajo.
2. B2 debe estar sin carga de trabajo.
3. B1 en la posición de B2 **no** deberá prolongar la secuencia inmediatamente anterior **ni** la posterior. Es decir, el cambio no prologará una carga de trabajo.

4. B2 inicialmente siempre deberá estar en una fila inferior a B1 en la matriz de soluciones.

En la Figura 7 podemos observar un ejemplo muy similar al ejemplo visto en la Figura 5, pero ahora B1 no deberá prolongar la secuencia anterior o posterior a B2.


**Fig. 7:** Ejemplo de soluciones pertenecientes al entorno de tipo 3

### 4.4.5. Entorno de tipo 4

El cuarto tipo entorno es el resultado de negar la tercera condición al Segundo Entorno. Las condiciones quedarían de la siguiente manera:

1. B1 debe tener algo de carga de trabajo.
2. B2 debe tener algo de carga de trabajo.
3. B1 en la posición de B2 **no** deberá prolongar la secuencia inmediatamente anterior **ni** la posterior. Es decir, el cambio no prologará una carga de trabajo.
4. B2 inicialmente siempre deberá estar en una fila inferior a B1 en la matriz de soluciones.

En la Figura 8 podemos observar un ejemplo donde el bloque B1 "AAW" se puede intercambiar por el bloque B2 "aav" puesto que se cumplen las 4 condiciones que moldean el entorno.

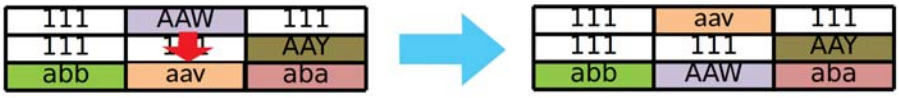



**Fig. 8:** Ejemplo de soluciones pertenecientes al entorno de tipo 4

Estos cuatro tipos de entornos están diseñados expresamente para que se ordene la matriz de soluciones de una forma piramidal, y se intente traspasar carga de trabajo de arriba hacia abajo.

Una vez que la matriz de trabajo tiene el mismo número de filas que controladores disponible para realizar el trabajo, procederemos a dejar de ordenar la matriz de soluciones de manera piramidal y eliminaremos la cuarta condición de los cuatro entornos que sólo permite buscar en filas inferiores puesto que lo que en esta etapa pretendemos es repartir la carga de trabajo para intentar cumplir con las restricciones estipuladas.

Por lo tanto, para diferenciar estos 4 entornos ya definidos de lo que serían los mismos entornos, pero sin la cuarta restricción, definiremos a estos últimos como **Entorno 1′**, **Entorno 2′**, **Entorno 3′** y **Entorno 4′**.

### 4.4.6. Definición de los entornos para la Fase 3

En la Fase 3 partimos de una solución factible, esto quiere decir que el número de filas de la matriz de controladores y el número de controladores disponibles son iguales y además cumple todas las restricciones, por lo que para este caso, como hemos explicado antes, no tiene sentido mantener la estrategia piramidal diseñada para el inicio de la Fase 2 por lo que procederemos a trabajar con los entornos 1′, 2′, 3′ y 4′.

Una característica nueva que se introduce en esta fase, ya que partimos de una solución factible, y no queremos perder esta característica, es que a los 4 entornos que hemos definido habrá que añadirles la restricción que impone para que una solución pueda ser vecina de otra, tendrá que mantener la totalidad de la factibilidad, es decir, que cumpla todas las restricciones, así como lo hacía la solución inicial.

De esta manera, los entornos a usar en esta fase quedarían de la siguiente manera:

**Entorno** 1′

1. B1 debe tener algo de carga de trabajo.
2. B2 debe estar libre de carga de trabajo.
3. B1 en la posición de B2 deberá prolongar la secuencia inmediatamente anterior o posterior. Es decir, el cambio prolongará una carga de trabajo.
4. El cambio deberá mantener la total factibilidad de la solución.

**Entorno** 2′

1. B1 debe tener algo de carga de trabajo.
2. B2 debe tener algo de carga de trabajo de trabajo.
3. B1 en la posición de B2 deberá prolongar la secuencia inmediatamente anterior o posterior. Es decir, el cambio prolongará una carga de trabajo.
4. El cambio deberá mantener la total factibilidad de la solución.

**Entorno** 3′

1. B1 debe tener algo de carga de trabajo.
2. B2 debe estar libre de carga de trabajo.

3. B1 en la posición de B2 **no** deberá prolongar la secuencia inmediatamente anterior **ni** la posterior. Es decir, el cambio no debe prolongar una carga de trabajo.
4. El cambio deberá mantener la total factibilidad de la solución.

**Entorno** $4'$

1. B1 debe tener algo de carga de trabajo.
2. B2 debe tener algo de carga de trabajo.
3. B1 en la posición de B2 **no** deberá prolongar la secuencia inmediatamente anterior **ni** la posterior. Es decir, el cambio no debe prolongar una carga de trabajo.
4. El cambio deberá mantener la total factibilidad de la solución.

## 4.5. Elección del orden de los entornos

La elección del orden de los entornos a través del cual se manejará el algoritmo será determinante a la hora de conseguir lograr los objetivos o no, y será uno de los principales parámetros que se variarán durante las pruebas.

En un principio, para la Fase 2, tal y como hemos detallado anteriormente, el primer orden en el que se ha planteado el algoritmo es el mismo en el que se han definido, puesto que aplicando el dominio del problema, y según la modelación de la solución propuesta, es el orden que parece más favorable. Como ya veremos en el análisis de los resultados, empíricamente resultará más favorable que cuando se haya conseguido reducir el número de filas de la matriz de controladores al número de controladores disponibles, cambiar el orden de los entornos para seguir la secuencia de entornos $2'$, $1'$, $3'$, $4'$ definidos en la Sección 4.4.5. De esta manera se pretende ser algo más exploratorio, puesto que el tamaño del entorno $2'$ suele ser algo mayor que el entorno $1'$.

En la Fase 3, continuaremos con el orden de entornos $2'$, $1'$, $3'$, $4'$ puesto que se ha comprobado empíricamente que favorece a la destrucción controlada de la estructura piramidal mejorando el valor en la función objetivo de la Fase 3 puesto que ahora se encarga de repartir carga de trabajo.

En la sección de análisis de los resultados volveremos a discutir sobre este parámetro.

## 4.6. Método de búsqueda local dentro de los entornos

Para esta etapa de búsqueda local se ha planteado un método con pocos parámetros, puesto que el estudio de los parámetros propios de esta metaheurística unido a los parámetros de búsqueda local podría resultar demasiado complejo.

Se ha planteado un algoritmo multicomienzo donde se elegirán un número $n$ de semillas por las que empezará el algoritmo. Cada semilla será repartida en una fracción del subespacio aleatoriamente siguiendo una distribución uniforme. Esta fracción de subespacio es el resultado de divdir el espacio total en *n+1* partes iguales.

El dominio en el que se han distribuido las soluciones es una línea recta en la cuál haremos *n+1* cortes para dividir los rangos. Una vez situada cada semilla aleatoriamente dentro de cada respectivo rango, la semilla elegirá una dirección, izquierda o derecha, aleatoriamente para comenzar a moverse.

La condición de parada es parametrizable en tanto por ciento del tamaño total del espacio y simbolizará la cantidad de soluciones que la semilla podrá visitar sin haber encontrado una mejor a la mejor actual.

El pseudocódigo se representa en el Algorithm 6.

**Algorithm 6** Búsqueda Local Simple

Definir el número $n$ de semillas para el multicomienzo
Definir el movimiento máximo por semilla como $sm$ en % del tamaño total de la vecindad
Dividir el espacio $S$ en $n$ subespacios $S_n$
Definir $MaxMov = sm$ * Tamanio de la vecindad
Definir $BestSol$ = Mejor solución global
**for** Desde $i = 0$ hasta $n$ **do**
    Establecer punto inicial aleatorio $x$ según distribución uniforme en el espacio $S_i$
    Elegir una dirección aleatoria
    Definir $Contador = 0$
    **repeat**
        Avanza de $x$ a $x'$ según la dirección elegida
        **if** $x'$ es mejor solución que $BestSol$ **then**
            $BestSol$ = Mejor solución global
            Contador = 0
            $x = x'$
            Volver al inicio de repeat
        **end if**
        **if** $x'$ fuera de los límites **then**
            Volver al inicio de for con $i++$
        **end if**
        $x = x'$
        $Contador++$
    **until** $Contador > MaxMov$
**end for**

### 4.6.1. Número de vueltas

El algoritmo puede tardar en torno a 5 minutos (depende del caso) en recorrer los entornos definidos para la correspondiente fase. Esto es un tiempo bastante pequeño en comparación con los órdenes de tiempo que se están manejando para resolver el problema, por eso vamos a permitir que el algoritmo dé varía vueltas cuando se quede estancado.

Diremos que el algoritmo ha recorrido 1 vuelta cuando se haya pasado por todos los entornos definidos sin encontrar ninguna solución mejor que la actual. Se ha propuesto dejar un total de cuatro vueltas sin mejorar la función objetivo como condición de parada del algoritmo. Este valor se ha determinado a partir del cálculo en el cual medimos que la búsqueda local explora en torno a un $30\,\% - 40\,\%$ del entorno cuando no está encontrando soluciones mejores. De esta manera damos la posibilidad de que el algoritmo encuentre una solución mejor en las siguientes vueltas.

### 4.6.2. Esquema global del algoritmo

Definida cada parte independiente del algoritmo, el esquema general se puede observar en la Figura 9. El diagrama de flujo está planteado de tal modo que cada una de las dos fases con las que trabajamos empezarán desde el punto inicial, y la salida de la Fase 2, será el punto de inicio de la Fase 3.

## 4.7. Optimización de tiempo

En el desarrollo de las metaheurísticas destinadas no sólo a la investigación si no también a la puesta en marcha en un entorno de producción como es el caso, a parte de la relación entre resultados e iteraciones también suele importar la relación de resultados y tiempo. Al introducir el tiempo en la ecuación, entra en juego como se ha implementado la metaheurística a nivel de programación, por eso se han analizado partes de la metaheurística que pudieran ser paralelizables sin verse afectado el resultado.

El algoritmo se ha divido en tres fases principales, la primera rellena una lista con todas las soluciones vecinas de la solución principal en ese momento, la segunda realiza la búsqueda local dentro del subsespacio formado por las vecindades incluidas en la lista, y por último, la tercera es la fase de evaluación.

De estas 3 fases las dos primeras son las más costosas computacionalmente, la primera se paralelizó de una forma casi directa, puesto que se encarga de realizar diversos cambios siempre a partir de la misma solución inicial, por lo que esta tarea se podía dividir en grupos y repartir el conjunto a los distintos procesadores o componentes de un sistema distribuido.

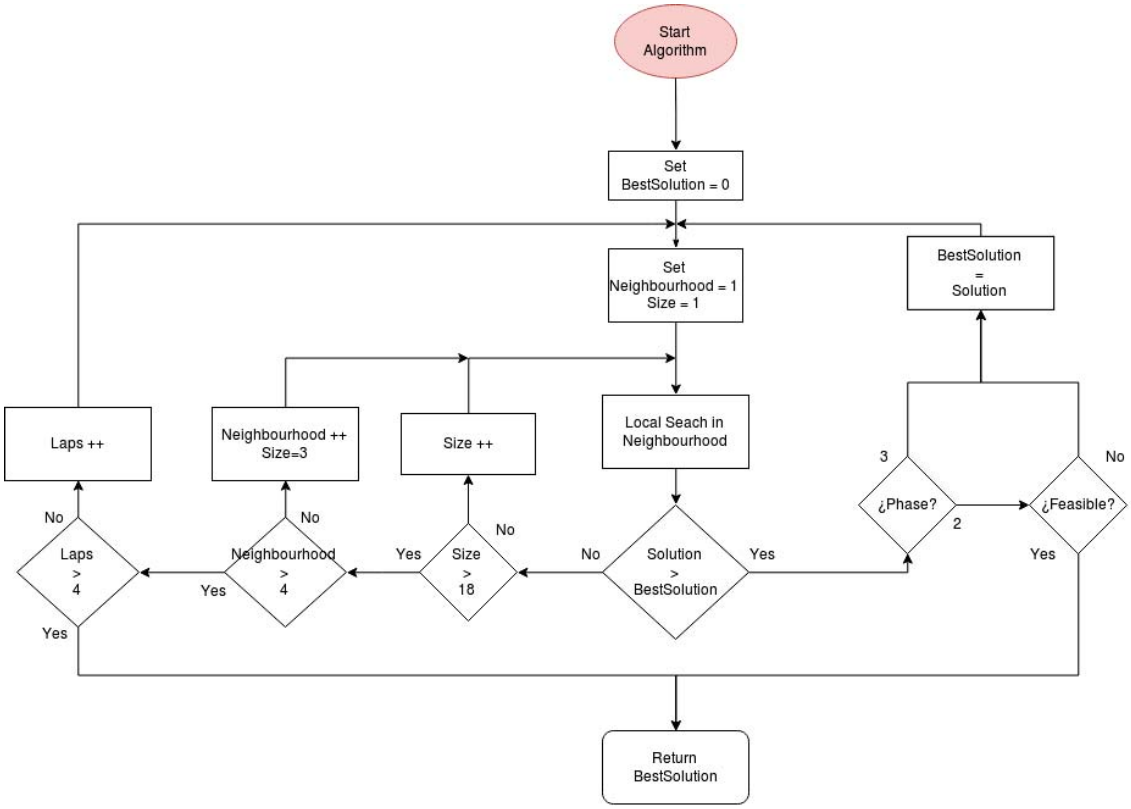



**Fig. 9:** Esquema general del algoritmo

Por desgracia la fase de búsqueda local no pudo ser paralelizada, a pesar de ser la más costosa, porque implicaría asumir ciertas restricciones fuertes modificando por completo los resultados.

# 5. Análisis de resultados

En este capítulo se propone realizar un análisis de los resultados explicando el comportamiento de la metaheurística estudiada en distintos casos reales, cuyos parámetros de entrada están proporcionados por CRIDA.

Dividiremos el análisis de los resultados en dos etapas, primero un análisis de cómo funciona el algoritmo en la Fase 2, donde intentaremos encontrar una solución factible con el número y tipo de controladores proporcionado, y posteriormente procederemos a hacer su análisis en la Fase 3, donde intentaremos optimizar el valor de la función objetivo creada a partir de los cuatro objetivos determinados en la Sección 4.2.3.

Antes de analizar los resultados cabe recordar que prevalecerá el encontrar una solución factible con menos controladores en la Fase 2, antes que encontrar una solución con más controladores en la Fase 2 y mejor función objetivo en la Fase 3, puesto que la función objetivo de la Fase 3, entre otras cosas, es una medida de comodidad para los controladores disponibles y en ningún caso tiene en cuenta el número de controladores.

Los resultados se han obtenido ejecutando la metaheurística en un ordenador con las siguientes características:

- **Procesador**: Intel Core i5-6500
- **RAM**: 16GB DDR4
- **SO**: Linux Mint 16.04

## 5.1. Estudio del algoritmo en la Fase 2

En esta sección se introducirá el análisis de resultados para la Fase 2. Esta fase empieza a partir de 10 plantillas generadas automáticamente por la Fase 1. Se aplicará el algoritmo a cada plantilla parando totalmente la ejecución cuando encontremos una solución factible. Cuando comparemos 2 evoluciones con algún cambio en la configuración del algoritmo, la evoluciones de ambos casos se mostrarán siempre partiendo de la misma plantilla.

### 5.1.1. Definición de los casos

En la Tabla 2 se detallan los distintos parámetros que caracterizan a cada uno de los casos que vamos a estudiar en esta sección. Podemos encontrar información acerca del centro de control al que pertenecen los sectores, la hora de inicio y de fin del periodo a resolver, los núcleos involucrados en el problema, el tipo de turno ("MC" hace referencia a Mañana-Corto mientras que "N" hace referencia a Noche), tipo de sector ("Ruta" o "App") y los recursos disponibles

de cada tipo. En los dos casos de Canarias se verán involucrados dos núcleos distintos, por eso hay especificados algunos parámetros adicionales.

**Tab. 2:** Especificación de los casos

| | **Caso 1** | **Caso 2** | **Caso 3** | **Caso 4** | **Caso 5** |
|---|---|---|---|---|---|
| **Centro de Control** | Barcelona | Canarias | Barcelona | Canarias | Barcelona |
| **Hora de inicio** | 05:20:00 | 07:00:00 | 06:20:00 | 22:00:00 | 05:20:00 |
| **Hora de fin** | 13:05:00 | 15:00:00 | 14:00:00 | 07:30:00 | 13:00:00 |
| **Núcleo** | Barcelona RutaW | Canarias Ruta | Barcelona RutaO | Canarias Ruta | Barcelona RutaE |
| **Tipo de turno** | MC | MC | MC | N | MC |
| **Tipo de sector** | Ruta | Ruta | Ruta | Ruta | Ruta |
| **Recursos disponibles** | 16 CON | 6 CON | 15 CON | 4 CON | 15 CON |
| **Núcleo** | - | Canarias App | - | Canarias App | - |
| **Tipo de turno** | - | MC | - | N | - |
| **Tipo de sector** | - | App | - | App | - |
| **Recursos disponibles** | - | 8 PTD | - | 9 PTD | - |

En la Tabla 3 podemos ver a que núcleo pertenecen y de que tipo son los distintos sectores que componen los centros de control de Barcelona y Canarias.

**Tab. 3:** Información de los sectores

| **Barcelona** | | | | **Canarias** | | | |
|---|---|---|---|---|---|---|---|
| **Sector** | **Tipo Sector** | **Núcleos** | | **Sector** | **Tipo Sector** | **Núcleos** | |
| | | **Barcelona RutaE** | **Barcelona RutaW** | | | **Canarias Aproximacion** | **Canarias Ruta** |
| LECBBAS | Ruta | X | | GCCCAAC | APP | X | |
| LECBBKE | Ruta | X | | GCCCACW | APP | X | |
| LECBCBM | Ruta | X | | GCCCD26 | APP | X | |
| LECBCCC | Ruta | X | X | GCCCF03 | APP | X | |
| LECBCCL | Ruta | X | X | GCCCF21 | APP | X | |
| LECBCCU | Ruta | X | X | GCCCFCN | APP | X | |
| LECBCVN | Ruta | X | X | GCCCFCW | APP | X | |
| LECBMLS | Ruta | X | | GCCCGCN | APP | X | |
| LECBMMI | Ruta | X | | GCCCGCS | APP | X | |
| LECBMNI | Ruta | X | | GCCCIGC | APP | X | |
| LECBMNL | Ruta | X | | GCCCINB | APP | X | |
| LECBMNU | Ruta | X | | GCCCTM3 | APP | X | |
| LECBMUS | Ruta | X | | GCCCTW2 | APP | X | |
| LECBMVI | Ruta | X | | GCCCCES | Ruta | X | X |
| LECBMVS | Ruta | X | | GCCCCWS | Ruta | X | X |
| LECBVMN | Ruta | X | | GCCCNWW | Ruta | X | X |
| LECBVMS | Ruta | X | | GCCCOCE | Ruta | X | X |
| LECBVNI | Ruta | X | X | GCCCR2E | Ruta | X | X |
| LECBVVI | Ruta | X | | GCCCRC2 | Ruta | X | X |
| LECBVVS | Ruta | X | | GCCCRCE | Ruta | X | X |
| LECBBKW | Ruta | | X | GCCCRCS | Ruta | X | X |
| LECBLGL | Ruta | | X | GCCCRCW | Ruta | X | X |
| LECBLGU | Ruta | | X | GCCCRE2 | Ruta | X | X |
| LECBLLI | Ruta | | X | GCCCRES | Ruta | X | X |
| LECBLVL | Ruta | | X | GCCCRNE | Ruta | X | X |
| LECBLVS | Ruta | | X | GCCCRU6 | Ruta | X | X |
| LECBLVU | Ruta | | X | GCCCRW3 | Ruta | X | X |
| LECBP1I | Ruta | X | X | GCCCRW4 | Ruta | X | X |
| LECBP1L | Ruta | X | X | GCCCRWS | Ruta | X | X |
| LECBP1U | Ruta | X | X | GCCCT13 | Ruta | X | X |
| LECBPP2 | Ruta | X | X | GCCCUW4 | Ruta | X | X |

En la Figura 10 se refleja la apertura y cierre de los sectores con el paso del tiempo en el Caso 1. Cada sector estará abierto mientras haya un cajetín con su nombre, mientras que si hay espacio en blanco refleja que el sector estará cerrado en ese periodo de tiempo. Por ejemplo en este caso, el sector "LECBP1I" estará abierto de 06:05 a.m. a 08:50 a.m. y de 12:05 a.m. a 13:05 a.m.

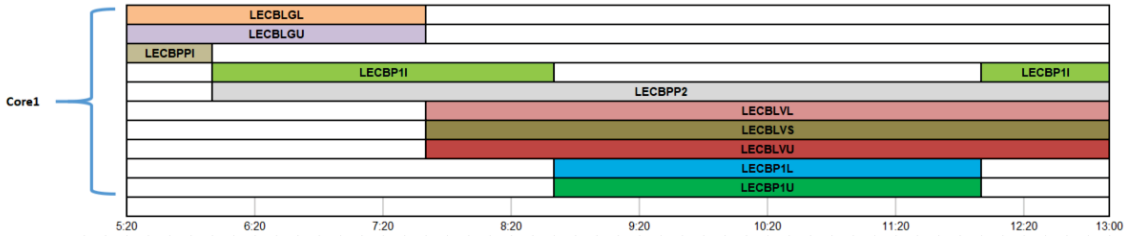



**Fig. 10:** Apertura y cierre de los sectores del Caso 1

Para el Caso 2, todos los sectores involucrados ( "GCCCOCE", "GCCCRU6", "GCCCAAC", "GCCCIGC", "GCCCINB") estarán abiertos de 07:00 a.m. a 15:00 p.m. como se observa en la Figura 11. En este tipo de gráfico, un color gris indicará que cierto sector está cerrado, un color verde supondrá que cierto sector de "Ruta" está abierto, y azul cuando sea un sector "App" el que esté abierto.

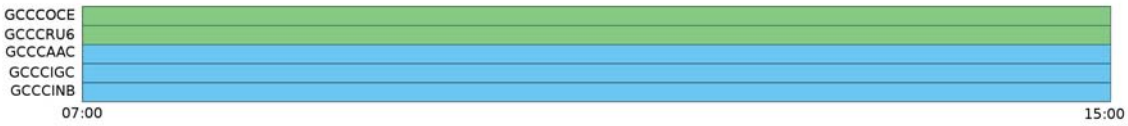



**Fig. 11:** Apertura y cierre de los sectores del Caso 3

Para el caso 3 la apertura y cierre de los sectores se pueden observar en la Figura 12.

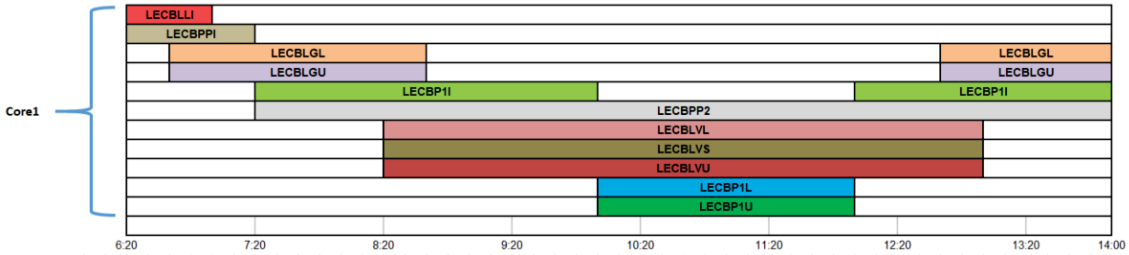



**Fig. 12:** Apertura y cierre de los sectores del Caso 3

La apertura y cierre de sectores involucrados en el Caso 4 se pueden ver reflejadas en la Figura 13.


**Fig. 13:** Apertura y cierre de los sectores del Caso 4

En la Figura 14 podemos ver la apertura y cierre de los distintos sectores a lo largo del periodo estudiado en el Caso 5.

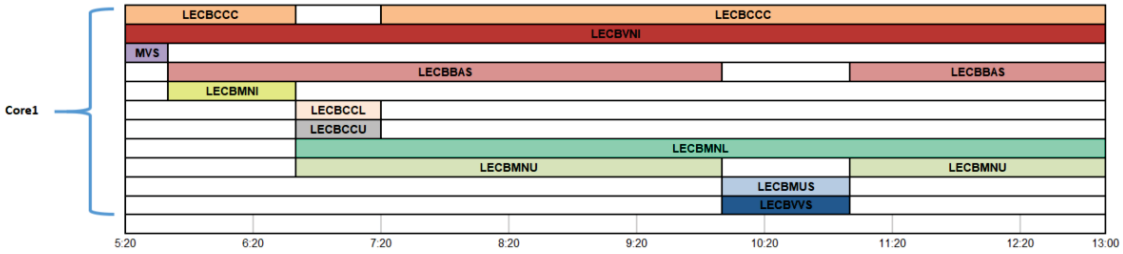



**Fig. 14:** Apertura y cierre de los sectores del Caso 5

### 5.1.2. Parámetros del algoritmo utilizados

En la Tabla 4 se definen los parámetros que usaremos en nuestro algoritmo, los cuales se han recogido en dos variantes que compararemos en los siguientes apartados. Las dos variantes únicamente difieren en el orden en el que se recorren los entornos de la metaheurística. Para la Fase 2 Se cambiará del "Orden de entornos 1" al "Orden de entornos 2" cuando el número de controladores involucrados en la solución sea el mismo que el número de controladores disponibles, recordando que los entornos pertenecientes al "Orden de entornos 2" de la Fase 2 se deshacen de la restricción que les obligaba a cambiar siempre un bloque por que estuviera en una parte inferior de la solucion con el objetivo de empezar a romper la estructura piramidal.

**Tab. 4:** Parámetros del algoritmo

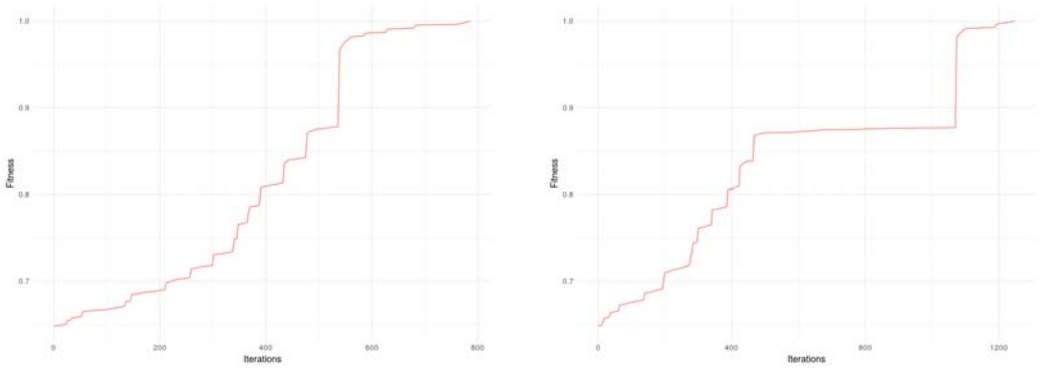
| | Variante A | Variante B |
|---|---|---|
| **Orden de entornos 1** | 1 - 2 - 3 - 4 | 1 - 2 - 3 - 4 |
| **Orden de entornos 2** | 1' - 2' - 3' - 4' | 2' - 1' - 3' - 4' |
| **Máximo de vueltas** | 4 | 4 |
| **Semillas** | 2 | 2 |
| **Máximo recorrido** | 25 % | 25 % |

El parámetro "Máximo recorrido" hace referencia al explicado en la Sección 4.6, el cual establece el máximo porcentaje del vecindario que podemos recorrer sin encontrar una solución mejor a la actual. El parámetro "Semillas" hace referencia al número de veces que la búsqueda local comenzará a iterar en un mismo vecindario y el parámetro "Máximo de vueltas" establecerá el número máximo de veces que recorreremos los 4 entornos sin haber encontrado una solución mejor.

### 5.1.3. Resultados de la Fase 2

En la Figura 15 se puede observar la evolución de la función objetivo para el Caso 1 utilizando tanto la Variante A de los parámetros como la Variante B.

Los saltos de la función que se pueden observar se deben a que el algoritmo ha conseguido reducir en esa iteración una fila de la matriz de controladores, y la función objetivo lo premia con un sustancial incremento de valor. A priori, se puede apreciar que la Variante A tiene una convergencia más adecuada en este caso puesto que consigue lograr la total factibilidad en torno a la iteración 790 mientras que la variante B lo consigue entorno a la iteración 1250 tras haberse quedado estancada un largo periodo de tiempo. La Variante A es más voraz que la Variante B y no se comportará tan bien en ejemplos más complejos como en el Caso 2.



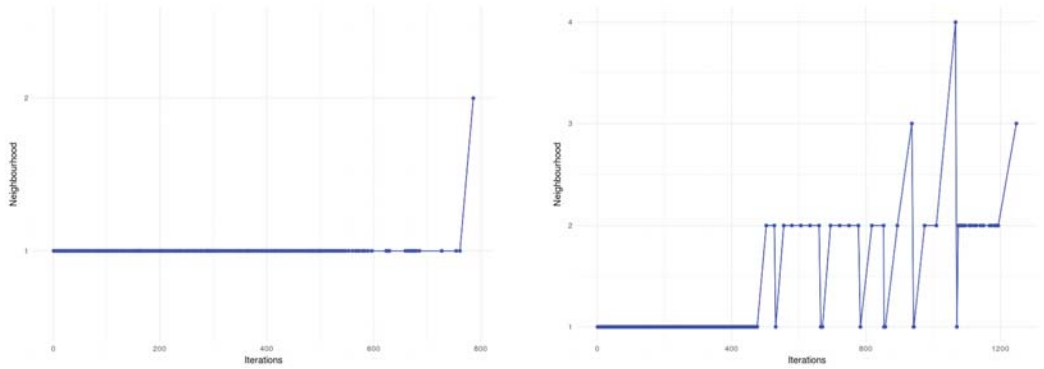

**(a)** Función objetivo del Caso 1 con la Variante A

**(b)** Función objetivo del Caso 1 con la Variante B

**Fig. 15:** Funciones objetivo del Caso 1

En la Figura 16 podemos observar en qué vecindad se ha encontrado las soluciones que iban incrementando el valor de la función objetivo para el Caso 1. Para la Variante A vemos como se van encontrando todas las soluciones en el vecindario 1 hasta que finalmente encuentra el cambio que otorga la solución factible en la vecindad 2'. Para la Variante B, comparándola con la Figura 15b podemos observar como para el periodo que estuvo la metaheurística estancada se estuvieron encontrando soluciones en las vecindades 1 y 2, pero que sólo suponían pequeñas mejoras y no es hasta entorno a la iteración 1080 en la que se encuentra una solución en la Vecindad 4 y se desestanca consiguiendo volver a converger a una solución factible rápidamente.

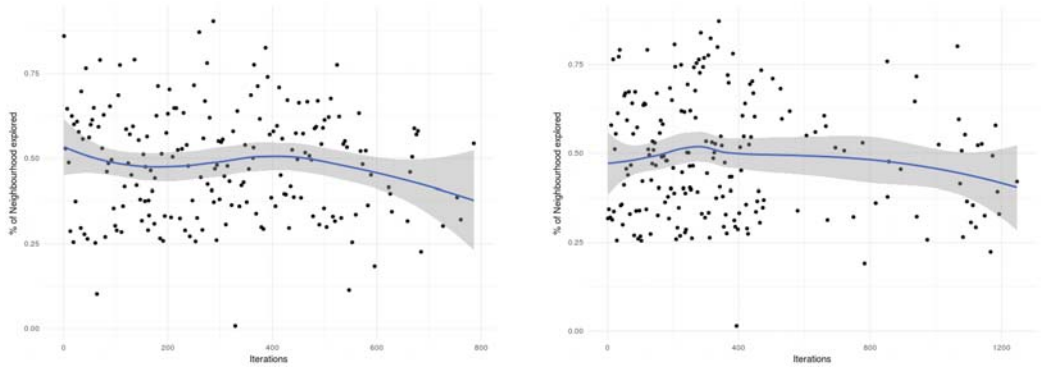
En la Figura 17 se representa el porcentaje de vecindario que se ha explorado para las dos variantes en las iteraciones donde se ha encontrado una mejor solución cuya media se sitúa en algo menos que el 50 %. Hay que tener en cuenta que estas trazas se obtienen sólo en las iteraciones en las que se mejora el valor en la función objetivo, por lo que la media global de exploratoriedad será menor que la mostrada, dado el algoritmo que hemos diseñado. Con esta configuración, la media de exploratoriedad cuando no se mejora, se sitúa en torno a un 30 % de todo el vecindario.




**(a)** Vecindades en las que se encuentra una mejor solución. Variante A.

**(b)** Vecindades en las que se encuentra una mejor solución. Variante B.

**Fig. 16:** Vecindades en las que se encuentra una mejor solución. Caso 1

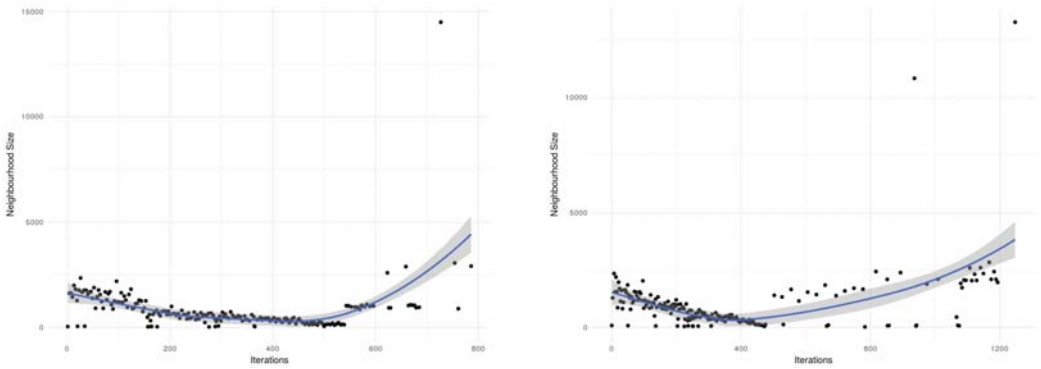



**(a)** Porcentaje del vecindario explorado. Variante A.

**(b)** Porcentaje del vecindario explorado. Variante B.

**Fig. 17:** Porcentaje de los vecindarios explorados. Caso 1

Por otro lado, en la Figura 18 podemos observar la curiosa y marcada tendencia descendente en las primeras iteraciones que casualmente coincide con el rápido aumento del valor de la función objetivo hasta llegar a su estancamiento. Este fenómeno tiene una explicación lógica, y es que al principio, con el orden establecido de los entornos y el diseño en estructura piramidal, se van reduciendo filas de la matriz de soluciones. Al reducir filas, los posibles cambios según el entorno definido son menores, lo que conlleva a menos vecinos. Si comparamos las Figuras 16a y 18a podemos ver que cuando el algoritmo busca de acuerdo a otros conceptos de entorno, va obteniendo vecindades con distinto número de soluciones.

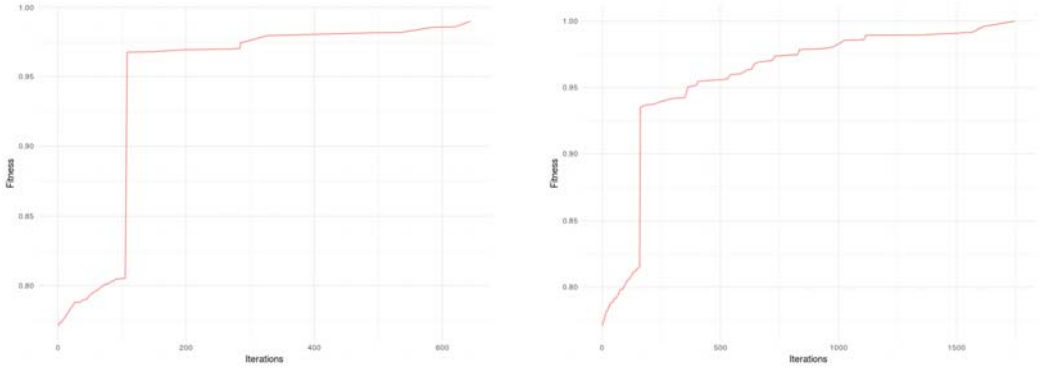



**(a)** Tamaño de las vecindades exploradas. Variante A.

**(b)** Tamaño de las vecindades exploradas. Variante B.

**Fig. 18:** Tamaño de las vecindades en las que se encuentra una mejor solución. Caso 1.

En la Figura 19 se estudia la evolución de la función objetivo del Caso 2 mediante las dos variantes. Se puede observar como en el caso de la Variante A no se consigue alcanzar la factibilidad completa al ser demasiado voraz y quedarse estancada en torno a un valor de 0.98 en 600 iteraciones sobrepasándose las 4 vueltas máximas especificadas como criterio de parada. Seguramente esta situación se dé porque se haya encasillado una estructura piramidal al ser demasiado voraz de tal manera que no logre encontrar ninguna alternativa que mejore la función objetivo. Sin embargo, la Variante B demuestra una convergencia más lenta y exploratoria que le permite llegar a la solución factible en 1750 iteraciones.



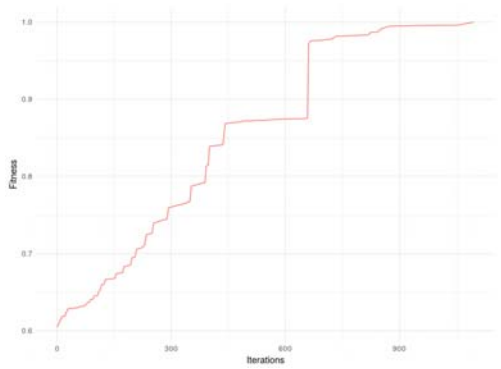

**(a)** Función objetivo del Caso 2. Variante A.

**(b)** Función objetivo del Caso 2. Variante B.

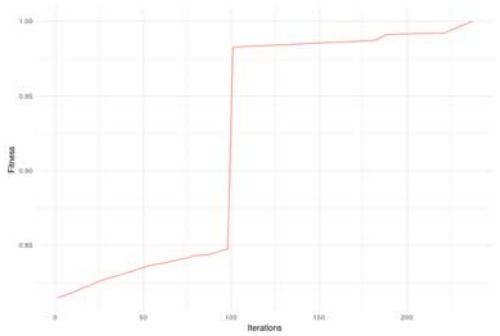
**Fig. 19:** Funciones objetivo del Caso 2.

Este caso expuesto es representativo de un conjunto de casos complejos donde la búsqueda voraz realizada tal y como en la variante A se queda estancada al converger demasiado rápido en la etapa final. En cambio, si optamos por aumentar la exploratoriedad en la etapa final, cambiando el orden de los entornos como en la Variante B, se consigue encontrar una solución factible.

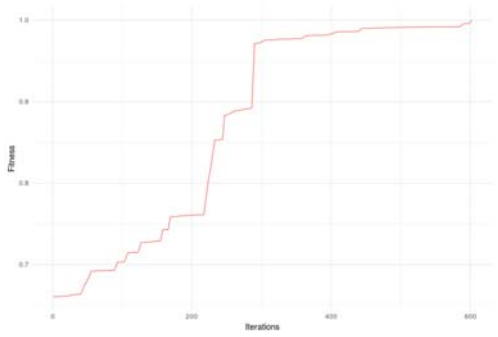
En la Figura 20 podemos observar la evolución de las funciones objetivo en los 3 casos restantes. Cabe recalcar que en el Caso 3 se consiguió alcanzar una solución factible con un controlador menos que con el método actual que implementa CRIDA.




**(a)** Función objetivo del Caso 3




**(b)** Función objetivo del Caso 4




**(c)** Función objetivo del Caso 5

**Fig. 20:** Funciones objetivo de los Casos 3, 4 y 5.

### 5.1.4. Conclusión de los resultados en la Fase 2

Tras este estudio del funcionamiento de la metaheurística propuesta en la Fase 2 podemos concluir que el algoritmo implementado posee unas características muy potentes que le ayudan a evitar óptimos locales y lograr en los casos propuestos soluciones factibles en menos tiempo y menos iteraciones que otras metaheurísticas implementadas, como la descrita en Tello et al. (2018) y posteriores variantes que se han ido desarrollando.

Además, nuestro algoritmo ha sido capaz de resolver uno de los cinco problemas propuestos con menos recursos que los utilizados en la mejor solución propuesta por CRIDA. Todo esto no hace más que recalcar la gran adaptabilidad y potencial de la metaheurística escogida en conjunto con el conocimiento del dominio introducido mediante la estrategia de resolución.

En la Tabla 5 se realiza una comparación de los rendimientos de tiempo y controladores necesarios para solventar cada caso entre el sistema de plantillas usado por CRIDA, el Recocido Simulado descrito en Tello et al. (2018) y VNS. En los campos donde se ha escrito una "-" significa que no se ha podido medir dicho elemento de la tabla. Como podemos observar, el algoritmo VNS consigue reducir el número de controladores en el Caso 3, siendo siempre más rápido que el Recocido Simulado. Sólo existe un caso en el que el sistema de plantillas da una solución con menos controladores y esto se produce porque para ese caso en concreto, desde CRIDA se simplifica el problema simulando que todos los sectores siempre están abiertos, por lo que no tienen los mismo parámetros de entrada al problema y los valores no son directamente comparables.

**Tab. 5:** Comparativa de la Fase 2

| **Fase 2** | | | | | | |
|---|---|---|---|---|---|---|
| **Caso** | **Plantillas** | | **SA** | | **VNS** | |
| | **Controladores** | **Tiempo (min)** | **Controladores** | **Tiempo (min)** | **Controladores** | **Tiempo (min)** |
| 1 | 16 | - | 16 | 70 | **16** | **9.11** |
| 2 | 14 | - | 14 | 58.81 | **14** | **15** |
| 3 | 16 | - | 16 | 16 | **15** | **9.30** |
| 4 | 13 | - | 13 | 1.36 | **13** | **1.34** |
| 5 | **14*** | **-** | 15 | 42 | 15 | 4.44 |

## 5.2. Estudio del algoritmo en la Fase 3

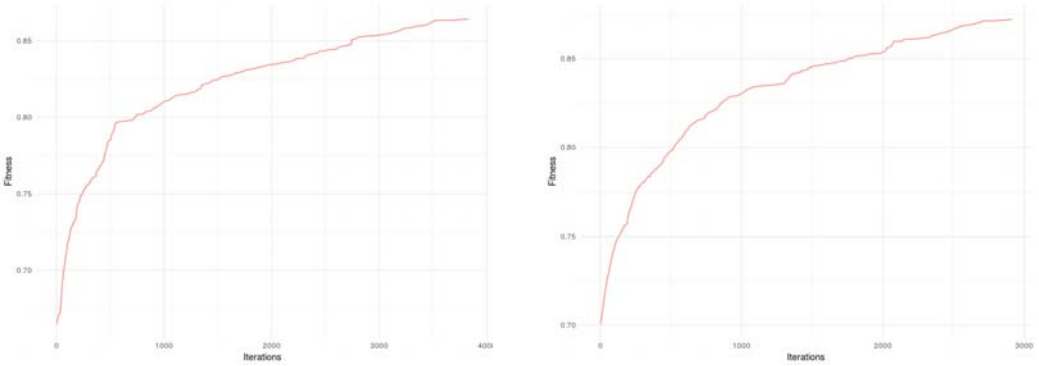
A continuación se muestran y analizan los resultados correspondientes a la Fase 3. La Fase 3 tiene como objetivo optimizar la solución factible ya obtenida de acuerdo a cuatro funciones objetivo explicadas en la Sección 3.2. El alcance de este proyecto se centraba principalmente en el desarrollo y estudio del algoritmo a la Fase 2, pero este es fácilmente modificable para trabajar en la Fase 3. La numeración de los casos siguientes tienen concordancia con la numeración de los casos de la Fase 2.

En esta fase se mantendrán las dos variantes descritas en la sección 5.1.2 con la única diferencia de que no hay un segundo orden de entornos, es decir que durante toda esta fase se mantiene el mismo. Cuando hablemos de Variante A nos referiremos al orden más voraz (1' - 2' - 3' - 4') mientras que nos referiremos a la Variante B (2' - 1' - 3' - 4') como al más explorativo.

En cada caso, el algoritmo se comparará con el valor en la función objetivo que tienen las plantillas tipo que se usan en CRIDA actualmente así como con los resultados obtenidos por el Recocido Simulado desarrollado en Tello (2015).

### 5.2.1. Reultados de la Fase 3

En la Figura 21 se presentan la evolución de las funciones objetivo de Caso 1 en la Fase 3 de acuerdo a las dos variantes definidas. Mediante la Variante A




**(a)** Función objetivo del Caso 1. Variante A.

**(b)** Función objetivo del Caso 1, Variante B.

**Fig. 21:** Funciones objetivo del Caso 1 en la Fase 3

se obtiene un valor de la función objetivo de 0.8641974 mientras que con la variante B alcanzamos un valor de 0.8724144.

En la Figura 22 mostramos como queda la solución final del problema obtenida a través de la Variante B. Se puede observar como las cargas de trabajo están bastante bien repartidas, aunque es difícil hacerse una idea a simple vista de los posibles cambios que se podrían realizar para mejorar la solución debido a la complejidad de las restricciones.

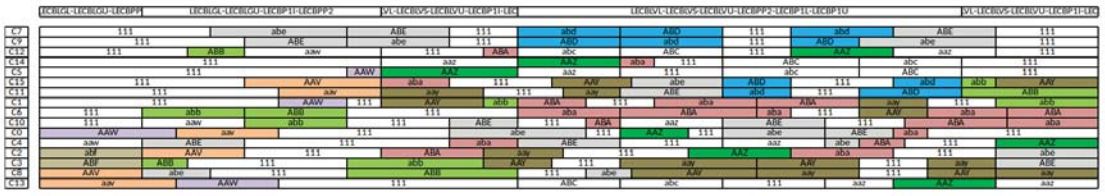
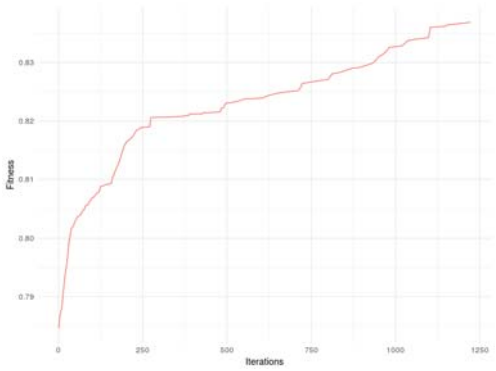
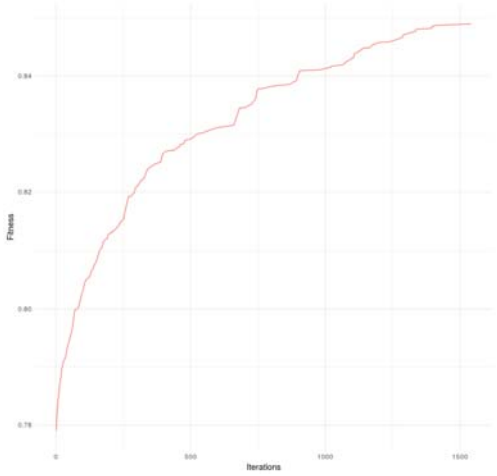


**Fig. 22:** Solución final para el Caso 1-B de la Fase 3

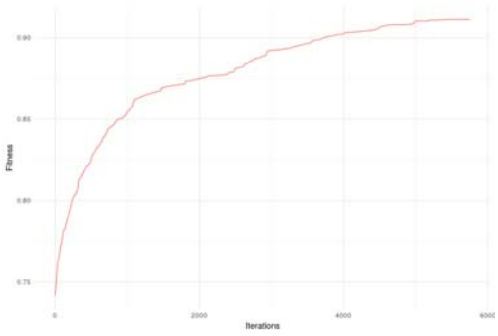
En la Figura 23 se puede observar la evolución de las funciones objetivo para los Casos 2, 3, 4 y 5 alcanzando unos valores de 0.8489, 0.9112, 0.7514 y 0.8531 respectivamente.



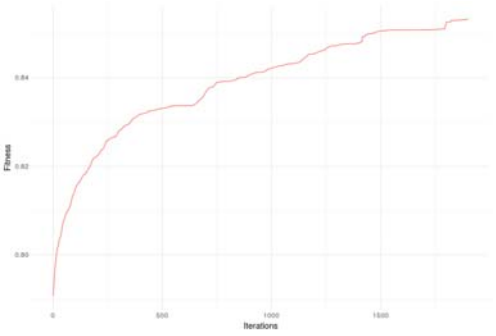

**(a)** Función objetivo del Caso 2 en la Fase 3




**(b)** Función objetivo del Caso 3 en la Fase 3




**(c)** Función objetivo del Caso 4 en la Fase 3




**(d)** Función objetivo del Caso 5 en la Fase 3

**Fig. 23:** Funciones objetivo de los casos 2, 3, 4 y 5 en la Fase 3.

### 5.2.2. Conclusión de los resultados en la Fase 3

En la Tabla 6 mostramos una comparativa de los rendimientos de tiempo y controladores necesarios para solventar cada caso así como su valor objetivo entre el sistema de plantillas usado por CRIDA, el Recocido Simulado descrito en Tello (2015) y VNS. En los campos donde se ha escrito una "-" significa que no se ha podido medir dicho elemento de la tabla. Como podemos observar para el Caso 1 y Caso 2, la metaheurística SA alcanza un valor objetivo algo más alto que VNS pero en un tiempo el doble de largo en un caso, y 17 veces mayor en otro, por lo que dependerá de la aplicación el elegir cuál de los dos algoritmos es más recomendable. En el Caso 3 se consigue reducir en un controlador la solución usando VNS, resultado que prevalece sobre cualquier comparación de la función objetivo. En el Caso 4 se consigue un valor objetivo mayor que cualquiera de los otros dos métodos en un tiempo considerablemente menor que el SA. En el Caso 5 CRIDA una solución con 14 controladores pero no usa

los mismos parámetros de entrada que usan los otros dos algoritmos, puesto que ellos en este caso dejan abiertos todos los sectores durante todo el turno lo que provoca que los resultados no sean comparables. SA consigue un valor mayor que VNS en este caso aunque también en un tiempo considerablemente mayor.

**Tab. 6:** Comparativa Fase 3

| Fase 3 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Caso** | **Plantilla** | | | **SA** | | | **VNS** | | |
| | **Controladores** | **Tiempo (min)** | **Valor Función Objetivo** | **Controladores** | **Tiempo (min)** | **Valor Función Objetivo** | **Controladores** | **Tiempo (min)** | **Valor Función Objetivo** |
| 1 | 16 | - | 0.8616 | 16 | 68 | **0.8790*** | 16 | 37.40 | 0.8724 |
| 2 | 14 | - | 0.8624 | 14 | 131 | **0.83942*** | 14 | 7.42 | 0.8369 |
| 3 | 16 | - | 0.8571 | 16 | 48 | 0.8811 | **15** | 18.91 | 0.8489 |
| 4 | 13 | - | 0.8581 | 13 | 96.77 | 0.9043 | 13 | 49.46 | **0.9112** |
| 5 | **14*** | - | 0.8571 | 15 | 61 | 0.8708 | 15 | 29.07 | 0.8531 |

# 6. Conclusión y trabajos futuros

En este trabajo se muestra la potencia y versatilidad que tiene la metaheurística *Variable Neibourhood Search* para resolver problemas de *time-tabling* con la particularidad de que es posible diseñar los entornos de acuerdo a la información del dominio que se tenga. En especial, esta metaheurística se aplica en este trabajo a dos fases de una estrategia propuesta para resolver un problema de asignación de controladores aéreos a sectores aéreos resolviendo un problema de una dimensionalidad media con grandes restricciones escapando de óptimos locales.

La Fase 2 del problema se soluciona de manera rápida y eficiente en comparación con otros métodos probados por el grupo de investigación, consiguiendo soluciones factibles en todos los casos propuestos y en uno de ellos consiguiéndolo con incluso un recurso menos de lo que actualmente se implementa. En la Fase 3 se consiguen unos resultados mejores que los que CRIDA está usando a fecha de la publicación en un número de iteraciones reducido.

En el desarrollo de este trabajo se han encontrado inconvenientes a esta metaheurística como puede ser el hecho de que no está planteada la posibilidad de que el algoritmo se mueva a soluciones peores fuera de la búsqueda local, lo que provocará que si a partir del punto encontrado, dentro de sus posibles vecindades, no hay una solución mejor, nunca podrá escapar del óptimo local, problema que solventan otras metaheurísticas permitiendo aceptar soluciones peores para aumentar la exploración.

Desarrollos futuros pueden ser la implementación de una variación de la metaheurística que permita escapar de los óptimos locales explicados en el párrafo anterior, o la implementación de otras metaheurísticas diferentes para resolver el problema y compararse con este método desarrollado.

## Referencias

Emile HL Aarts and Jan HM Korst. Simulated annealing. *ISSUES*, 1:16, 1988.

Hans J Bremermann. Optimization through evolution and recombination. *Self-organizing systems*, 93:106, 1962.

Alberto Colorni, Marco Dorigo, and Vittorio Maniezzo. Metaheuristics for high school timetabling. *Computational optimization and applications*, 9(3): 275–298, 1998.

Fred Glover. Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

Pierre Hansen, Nenad Mladenović, and Dionisio Perez-Britos. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4):335–350, 2001.

Alain Hertz. Tabu search for large scale timetabling problems. *European journal of operational research*, 54(1):39–47, 1991.

Alain Hertz and Michel Mittaz. A variable neighborhood descent algorithm for the undirected capacitated arc routing problem. *Transportation science*, 35(4):425–434, 2001.

Madhukar R Korupolu, C Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. *Journal of algorithms*, 37(1):146–188, 2000.

Rhydian Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR spectrum*, 30(1):167–190, 2008.

Helena R Lourenço, Olivier C Martin, and Thomas Stutzle. Iterated local search. *International series in operations research and management science*, pages 321–354, 2003.

Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

José Andrés Moreno and Nenad Mladenović. Búsqueda por entornos variables para planificación logística. *Universidad de La Laguna*, 38271.

John F Muth and Gerald Luther Thompson. *Industrial scheduling*. Prentice-Hall, 1963.

University of Twente. International Timetabling Competition. `http://aiweb.techfak.uni-bielefeld.de`, 2011. [Online; accessed 12-February-2018].

Adán Suárez. Asignación de controladores a sectores aéreos mediante heurísticas trayectoriales: Búsqueda tabú. *Tesis Fin del Máster Universitario en Inteligencia Artifiicial*, 2017.

Faustino Tello. Asignación óptima de expertos a puestos de trabajo en base a expresiones regulares y el uso de recocido simulado. 2015.

Faustino Tello, Alfonso Mateos, Antonio Jiménez-Martín, and Adán Suárez. The air traffic controller work-shift scheduling problem in spain from a multiobjective perspective: A metaheuristic and regular expression-based approach. *Mathematical Problems in Engineering*, 2018, 2018.