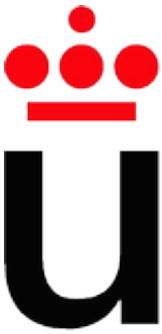

Universidad Rey Juan Carlos

Desarrollo e implementación de algoritmos heurísticos para la gestión óptima de flujo en tráfico aéreo

Trabajo Fin de Grado

Escuela Técnica Superior de Ingeniería Informática

Ingeniería del Software

Curso 2016–2017

Carlos Vázquez Sánchez

Tutor:
Antonio Alonso Ayuso

Índice general

Índice de figuras

Capítulo 1

Introducción y nuevos objetivos

1.1. Introducción

En las últimas décadas el número de pasajeros de avión ha aumentado exponencialmente, conllevando un aumento prácticamente similar del tráfico aéreo. Tan sólo en España en el año 2012 hubo 195 millones de viajeros en la red AENA, siendo el aeropuerto de Madrid-Barajas con 45 millones de viajeros el más visitado, seguido de cerca por el de Barcelona-El Prat con 35 millones.

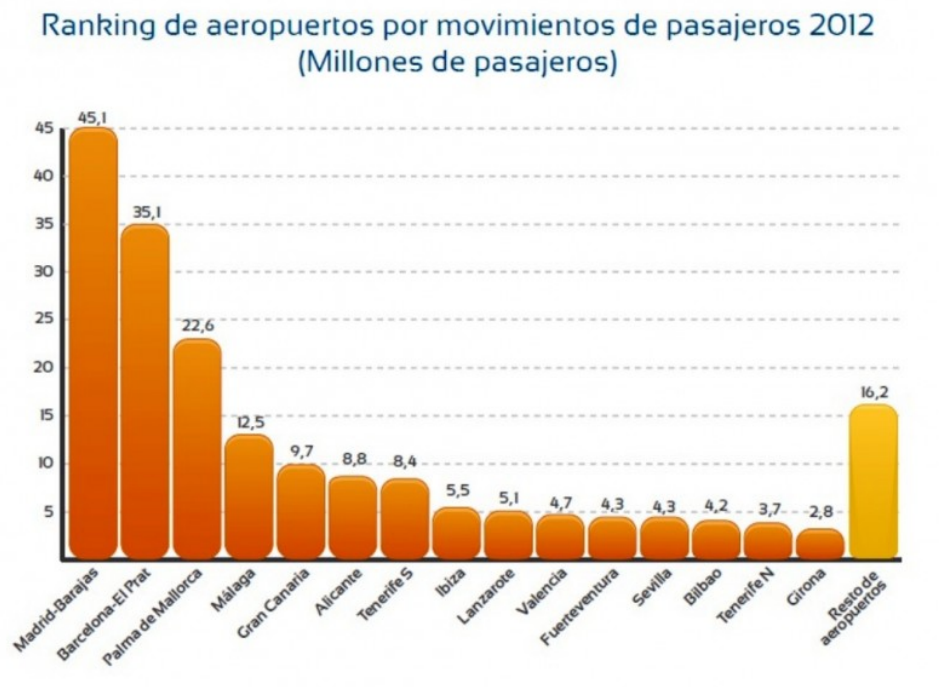

Figura 1.1: Viajeros en aeropuertos españoles. *Fuente: AENA*

Este incremento del tráfico aéreo (actualmente hay de media 11.000 vuelos simultáneos por minuto) sumado a las limitaciones estructurales que tienen los aeropuertos para expandirse, ha ocasionado un importante problema de sobresaturación del espacio aéreo.

Para tratar de solucionar este problema, se creó en 1988 el Central Flow Management Unit (CDMU), entidad dependiente de EUROCONTROL cuyo contenido es centralizar y estructurar todo el tráfico europeo, al igual que hace el Air Traffic Control System Command Center en Estados Unidos. El CFMU trabaja a 3 niveles, según sea el espacio temporal sobre el que estén trabajando:

- **Planificación táctica:** se realizan el mismo día para manejar las excepciones no previstas. Se trata de un modelo matemático diseñado para dar una respuesta muy rápida a una situación de excepción.
- **Planificación pre-táctica:** llevada a cabo con 2 días de antelación, se encarga de analizar el tráfico de los días previos, así como la previsión meteorológica.
- **Planificación estratégica:** se lleva a cabo cada 6 meses. En ella se elaboran los planes de vuelo para cada compañía y se asignan los recorridos disponibles para cada vuelo.

A pesar de todas estas estrategias de planificación, el conjunto histórico de datos, y las mejoras de previsión meteorológica, sigue existiendo un porcentaje importante de retrasos y cancelaciones. Según datos de EUROCONTROL, en el año 2016 un 20 % de los vuelos tuvieron un retraso superior a los 15 minutos:

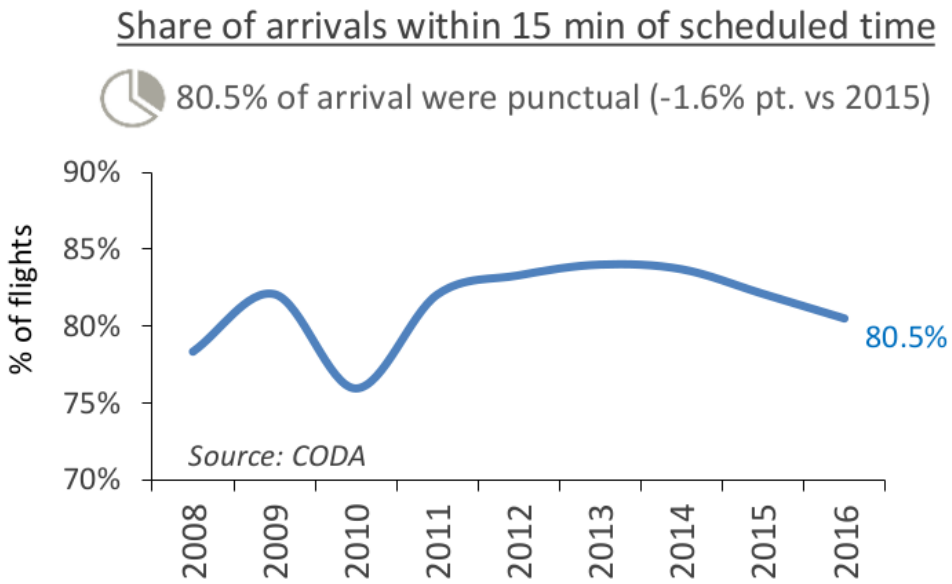

Figura 1.2: Porcentaje retrasos en 2016. *Fuente: EUROCONTROL*

Los motivos por los que un vuelo no llega en la hora prevista a su destino depende en gran medida de la zona geográfica que estudiemos: mientras que en Estados Unidos la mayor parte de los retrasos se debe a un cuello de botella que existe en sus aeropuertos, en Europa es la sobresaturación del espacio aéreo el principal problema, ya que hay un gran número de aeropuertos con mucho tráfico muy cercanos unos de otros, mientras que en EEUU están más dispersos.

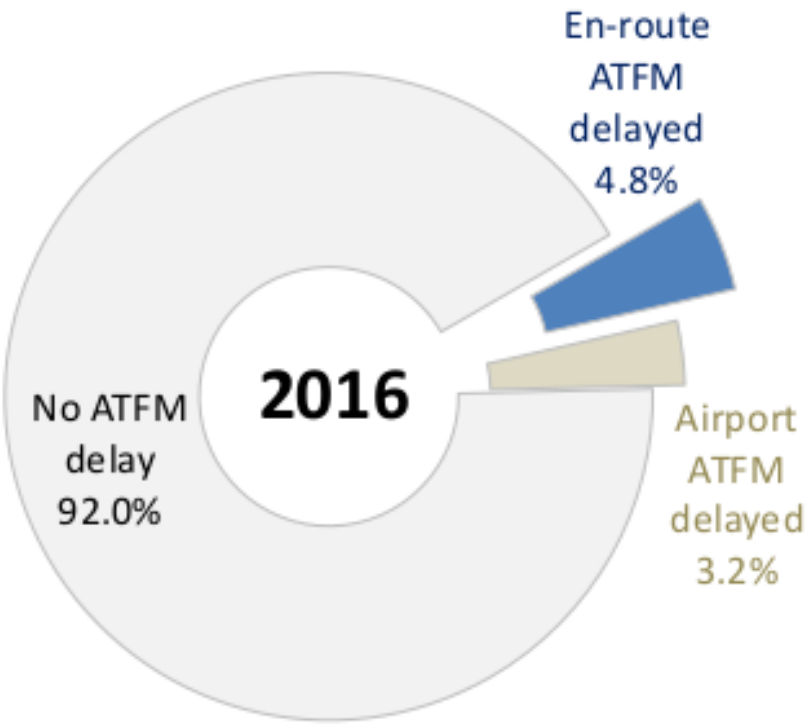

Figura 1.3: causas posibles de retraso de un vuelo. *Fuente: EUROCONTROL*

Durante las últimas décadas se han desarrollado varios modelos cuya finalidad es tratar de resolver (o al menos minimizar) los retrasos en vuelos. Aunque los primeros estudios se centraron más en el modelo estadounidense, y por tanto trataban de resolver el cuello de botella de sus aeropuertos, recientemente también se han hecho estudios europeos más centrados en la saturación del espacio aéreo:

1. **Singe-Airport Ground-Holding Problem (SAGHP):** poco utilizada a excepción de algunos aeropuertos italianos, este modelo tiene en cuenta el número de despegues y aterrizajes que puede soportar un aeropuerto por unidad de tiempo.
2. **Multi-Airport Ground-Holding Problem (MAGHP):** muy similar al anterior, salvo que maneja varios aeropuertos de forma conjunta, teniendo en cuenta las relaciones entre ellos.

3. **Air Traffic Flow Management Problem (ATFMP)**: introduce en el modelo el espacio aéreo. Como se puede ver en la figura 1.3, el porcentaje de retrasos en ruta fue mayor que el retraso en tierra. Esto se debe a que si un determinado espacio aéreo no está disponible (por sobresaturación, causas meteorológicas, etc), solo se verán afectados los vuelos cuya ruta estaba programada por ese sector.
   Sin embargo, la saturación de un aeropuerto ocasiona un retraso en cadena, afectando a todos los vuelos que iban a despegar o aterrizar en dicho aeropuerto.

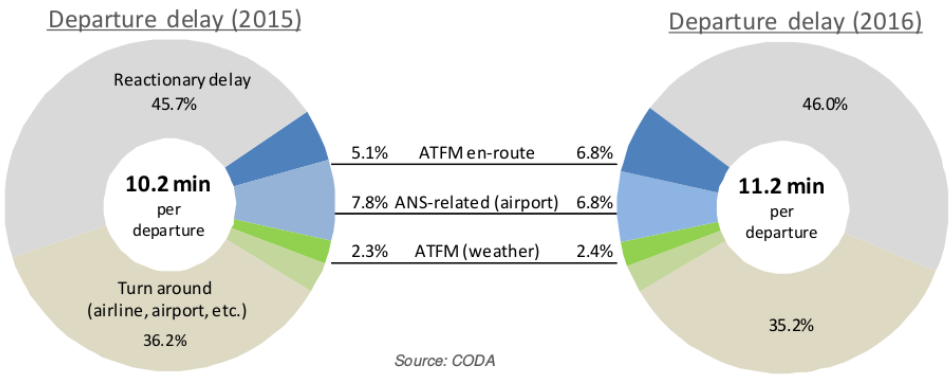

Figura 1.4: Comparativa retrasos 2015-2016. *Fuente: EUROCONTROL*

4. **Air Traffic Flow Management Rerouting Problem (ATFMRP):** muy similar a ATFMP, pero añade además la posibilidad de desviar vuelos.

5. **Air Traffic Flow Management Rerouting with Flight Cancellation Problem (ATFMRCP):** añade la posibilidad de cancelar vuelos. Se trata de un modelo más teórico que práctico, ya que la opción de cancelar vuelos no es una opción válida para casos reales.

1.2. Versiones anteriores

Este Trabajo Final de Grado es la continuación del trabajo que llevaron a cabo Diego Ruiz Aguado y Gonzalo Quevedo García en 2012 en sus Proyectos Finales de Carrera, los cuales se apoyaron a su vez en la Tesis Doctoral de Alba Agustín Martín (2011).
A continuación se hace una breve descripción del trabajo de Diego Ruiz Aguado y Gonzalo Quevedo García:

1. Los datos del problema se encontraban en una base de datos no relacional con redundancias. El primer paso consistió en migrar esta base de datos no relacional a una base de datos MySQL relacional y bien estructurada.

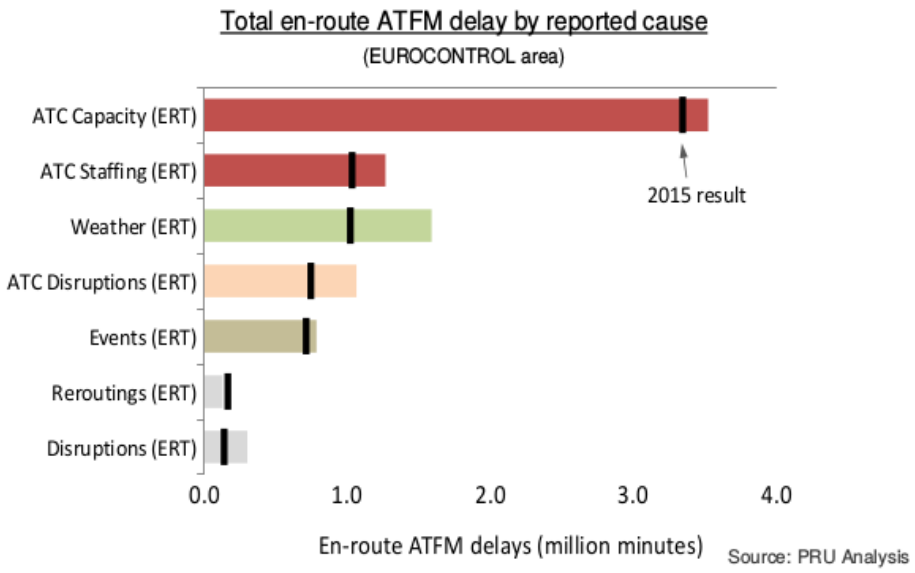

Figura 1.5: Motivos de retrasos en 2016. *Fuente: EUROCONTROL*

2. Para obtener los datos que necesitaba el problema, se realizó un programa en java que se conectaba a la BBDD y creaba varios ficheros .txt en la que se volcaba toda la información necesaria para el posterior modelado del problema.

3. A continuación, el programa en java leía estos ficheros .txt y creaba las estructuras de datos necesarias (árbol de rutas, vuelos, waypoints, etc).

4. Posteriormente una subrutina en C se encargaba de definir un problema de CPLEX con la función objetivo y las restricciones necesarias.

5. Finalmente, se ejecutaba el problema de optimización mediante la librería CPLEX para obtener la mejor solución del problema.

Una vez seleccionado el vuelo al que se le iba a encontrar solución, el algoritmo CPLEX se encargaba de buscar una solución factible. A continuación se explica brevemente el funcionamiento de la librería de optimización:

CPLEX

CPLEX es una librería de optimización actualmente propiedad de IBM implementada en el lenguaje de programación C. CPLEX permite modelar un problema de optimización definiendo la función objetivo y todas las restricciones.
En la versión del problema de 2012 se utilizaba de la siguiente manera:

1. Creación del modelo: mediante los ficheros .txt que contenían la información de la base de datos del problema se obtienen los nodos, arcos, restricciones y función objetivo del problema.

2. Se optimiza el problema mediante CPLEX, del que obtendremos el valor final de la función objetivo

1.3. Nuevos objetivos

La versión anterior del problema adolecía de un importante inconveniente: la función que se encargaba de tratar de encontrar la solución factible para cada vuelo se basaba en una función voraz [1], por lo que el problema no podía salir de los máximos locales. De esta forma, el resultado del problema dependía en gran medida del orden en que se intentara encontrar una solución para cada vuelo.

Por tanto los objetivos marcados para este Trabajo de Fin de Grado han sido los siguientes (ordenados en decreciente prioridad):

1. **Mejorar heurístico:** el objetivo principal de este TFG consiste en sustituir el algoritmo voraz por un heurístico que permita al problema escapar de los máximos locales, y por tanto aumentar en gran medida la calidad de la solución encontrada.

2. **Desacoplar el programa de CPLEX y nueva estructura:** con la implementación de los nuevos heurísticos no es necesaria la librería de optimización. Se pasará de un sistema clásico de optimización (función objetivo y restricciones) a una estructura de objetos que permitan un manejo óptimo de las estructuras de datos durante la ejecución del algoritmo.

3. **Mejorar el sistema de lectura de datos:** la versión actual del programa crea ficheros .txt en los que se vuelca toda la A2 del problema (vuelos, waypoints, rutas, etc) que pueden superar las 100.000 lineas. Estos ficheros auxiliares pueden sustituirse por ficheros mucho más pequeños en los que se exporta la información de la base de datos, y de forma interna el problema se encarga de crear las estructuras necesarias. De esta forma se reduce en gran medida el tiempo de creación del modelo del problema, además de obtener un módulo de lectura de datos exportable y fácilmente entendible.

4. **Representación gráfica:** aunque estrictamente no aporta a mejorar la solución del problema, su representación gráfica puede ayudar a modelizar mejor el algoritmo, ya que permite visualizar de manera rápida y sencilla el estado del problema.

Debido al enfoque teórico de este Trabajo de Fin de Grado, el modelo que se va a implementar se basa en el ATFMRCP, ya que permitirá cancelaciones, retrasos en aeropuertos, retrasos en ruta, y desvíos.

[1] Un algoritmo voraz es aquel que sigue una heurística consistente en elegir la opción óptima en cada iteración para encontrar la mejor solución dado un determinado problema.

Capítulo 2

Descripción del problema

El modelo implementado se puede resumir como un espacio aéreo dividido en diferentes sectores por el que viajan una serie de aviones. Los vuelos parten y finalizan de un aeropuerto, y tienen una serie de rutas para llegar a su destino.

El objetivo del problema es encontrar una solución factible al mayor número de vuelos cumpliendo una serie de restricciones. Las entidades de las que se compone el modelo son sectores, waypoints, vuelos, rutas y waypointsRoute. A continuación se describe co más detalle las características de cada uno de estos elementos.

2.1. Sectores

Representan las diferentes zonas en las que se divide el espacio aéreo. Los sectores tienen un límite de capacidad, de modo que para un instante e tiempo t solo puede haber un número n de vuelos simultáneamente en un sector. De este modo, si un vuelo tiene programada una ruta en un momento de tiempo que pasa por un sector que está al límite de su capacidad, esa ruta no será válida, por lo que tendrá que intentar retrasar su ruta, usar una ruta alternativa, y si no tiene otra opción, cancelarse.

Para las pruebas que hemos realizado se ha considerado el escenario más restrictivo posible: la capacidad de un sector en cada instante de tiempo es 1.

2.2. Waypoints

Los waypoints representan puntos de ruta en las trayectorias de los vuelos a través del espacio aéreo. Dependiendo de su ubicación, los waypoints pueden estar en el interior del sector, por ejemplo en el caso de los aeropuertos, o en la intersección entre dos o más sectores para representar el paso de un sector a otro.

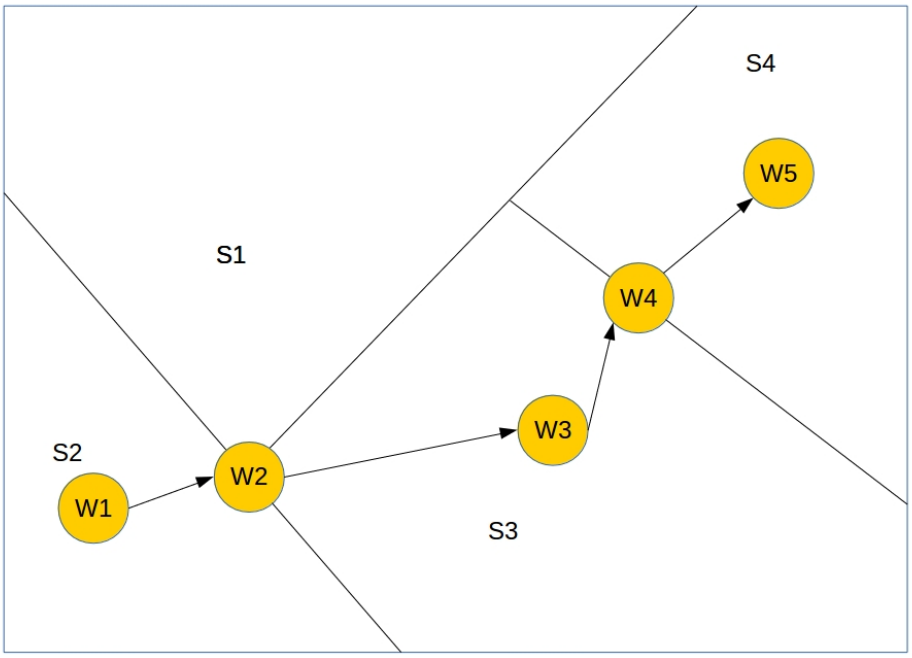

Figura 2.1: Ejemplo sectores y waypoints.

2.3. Vuelos

Los vuelos parten siempre de un Waypoint aeropuerto y a través de una serie de rutas preasignadas, que marcan los posibles itinerarios de vuelo a través de Waypoints, llegan a un aeropuerto destino. Los vuelos pueden llegar a su aeropuerto de destino por cualquiera de sus rutas, así como también retrasar su itinerario si fuera necesario, pero si debido a las restricciones del problema no se encontrara una solución factible, el vuelo sería cancelado.

Por tanto, los vuelos se pueden clasificar en función de su estado al final del problema en:

- **Vuelos programados:** se ha encontrado una ruta para el vuelo que cumple con las restricciones del problema. A su vez, los vuelos con solución se pueden subdividir en
  - **Solución por defecto:** la solución del vuelo es la ruta inicial sin retrasos. Es el mejor resultado posible.
  - **Retrasado:** se encuentra una solución factible en la ruta por defecto del vuelo, pero se ha producido un retrasos entre alguno de sus waypoints.
  - **Desviado:** la solución encontrada para el vuelo no es la ruta a priori. Puede ser tan corta como la solución por defecto.

- **Vuelos no programados** no se ha podido localizar ninguna ruta factible para el vuelo, por lo que se considera como cancelado.

Los vuelos tienen un coste de cancelación y están asociados a determinadas aerolíneas.

2.4. Rutas

Las rutas son el conjunto de trayectorias que tiene un vuelo para llegar desde su aeropuerto de origen al de destino, y se representan como un grafo ponderado y dirigido en el que el coste de cada arista coincide con el intervalo de valores entre los cuales un vuelo puede hacer el trayecto entre 2 waypoints.

De todas las posibles rutas, una es la que se toma como ruta preestablecida, y es la que se indica en el plan de vuelo original. Esta ruta predefinida será siempre igual o mejor (más rápida) que el resto de sus rutas. El objetivo del problema es encontrar un plan de vuelo que suponga la menor variación posible sobre el plan de vuelo original.

Una de las características más importantes del modelo es que se permite que un vuelo pueda retrasar su trayecto entre 2 waypoints, normalmente para no coincidir en el tiempo y el espacio con otro vuelo o porque el sector al que intenta acceder está sobrecargado.

Por tanto en cada trayectoria entre waypoints se permite un retraso máximo, de forma que entre 2 waypoints un vuelo puede recorrer esa distancia entre

$$[T_{mín}, T_{máx}] \tag{2.1}$$

, donde $T_{mín}$ y $T_{máx}$ son el tiempo mínimo y máximo en el que un vuelo puede ir desde un waypoint a otro, respectivamente.

Para modelar que los aeropuertos no tienen capacidad, creamos unos waypoints $aeropuerto'$ que simulan el waypoint en el que los vuelos aterrizan o despegan (estos waypoints sí que tienen las restricciones habituales del resto de waypoints). Por ejemplo, un vuelo entre 2 aeropuertos con un waypoint entre ellos en el que se permita en cada trayectoria un retraso de 1, daría como resultado el siguiente grafo:

Figura 2.2: Ejemplo ruta de un vuelo.

En este ejemplo, el retraso entre $A1$ y $A1'$ representa el retraso en tierra que se permite para el vuelo. El retraso entre los nodos $A3'$ y $A3$, al igual que en cualquier conexión entre los nodos $aeropuerto\_aterrizaje'$ y $aeropuerto\_aterrizaje$ será siempre 0, ya que no tiene sentido que un vuelo que llega a su aeropuerto de destino retrase su ruta.

2.5. Waypoint Route

Los waypoint route son la entidad que representa los waypoints en diferentes instantes de tiempo. Los waypoint routes son necesarios para crear el grafo de recorridos *completo* de un vuelo, ya que si en alguno de sus trayectos un vuelo permite algún retraso, obtendremos que las aristas del grafo representan a un conjunto de valores. Dado que los waypoint route dependen de las rutas de cada vuelo, los waypoints route son únicos para vuelo, de forma que el grafo de recorrido de cada vuelo es independiente de los otros vuelos.

Para crear el grafo que represente las posibles rutas de un vuelo y todos sus posibles retrasos, tenemos que crear un nodo por cada waypoint en cada posible instante de tiempo t. De esta forma, tendremos un grafo con un conjunto de nodos de la forma W_t, siendo W el nombre del waypoint y t el instante de tiempo.
Tras realizar este proceso, se incorpora al modelo que un vuelo puede estar en el mismo waypoint en diferentes momentos de tiempo si se ha elegido otra ruta más larga o se ha producido un retraso.

En el ejemplo de Figure 2.2, la forma expandida del grafo sería (suponiendo que el vuelo despega en el instante $t = 0$) correspondería con la Figure 2.3:

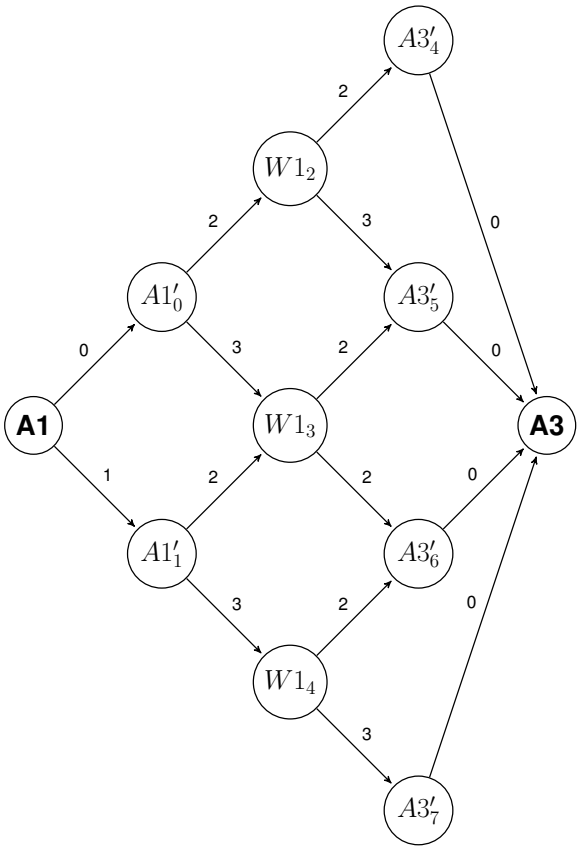

Figura 2.3: Ejemplo grafo de recorridos de un vuelo.

Un vuelo puede además tener más de una ruta. En este caso el grafo de recorridos tendrá más de un camino para llegar al nodo de destino:

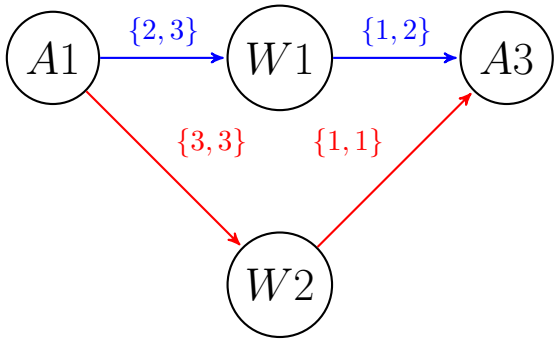

Figura 2.4: Ejemplo vuelo con 2 rutas.

Su grafo de recorridos se compondría de forma análoga a los anteriores como se puede ver en la Figure 2.5

2.6. Descripción del modelo

Por tanto, el problema se puede resolver mediante un modelos de optimización combinatoria lineal con los siguientes elementos:

- **Función objetivo:** hay que maximizar el número de vuelos a los que se les encuentra una solución factible.
  Dado que los vuelos no tienen solución binaria (cancelado/con solución), sino que dentro de las soluciones factibles hay unas mejores que otras, se ha creado un sistema de evaluación de vuelos con solución factible: los vuelos colocados en su solución a priori aportan 5 pts, los retrasados 3, los desviados en tiempo 2 y los desviados y retrasados 1.
  Este sencillo sistema se ideó para evaluar lo factible que es una solución, ya que como se explicó anteriormente no estamos utilizando el coste de cancelación (en ese caso se trataría de un problema de minimización de costes, en vez de maximizar la solución).

- **Restricciones:** el modelo tiene tan sólo dos:

  1. Para cualquier instante de tiempo t, no puede haber más de 1 vuelo en un arco que conecta 2 waypoints.
  2. Dado cualquier instante de tiempo t, no puede haber más de n vuelos simultáneamente en el mismo sector. Aunque n es un parámetro variable, todas las pruebas que hemos realizado hemos usado $n = 1$, poniéndonos así en el caso más estricto de todos.

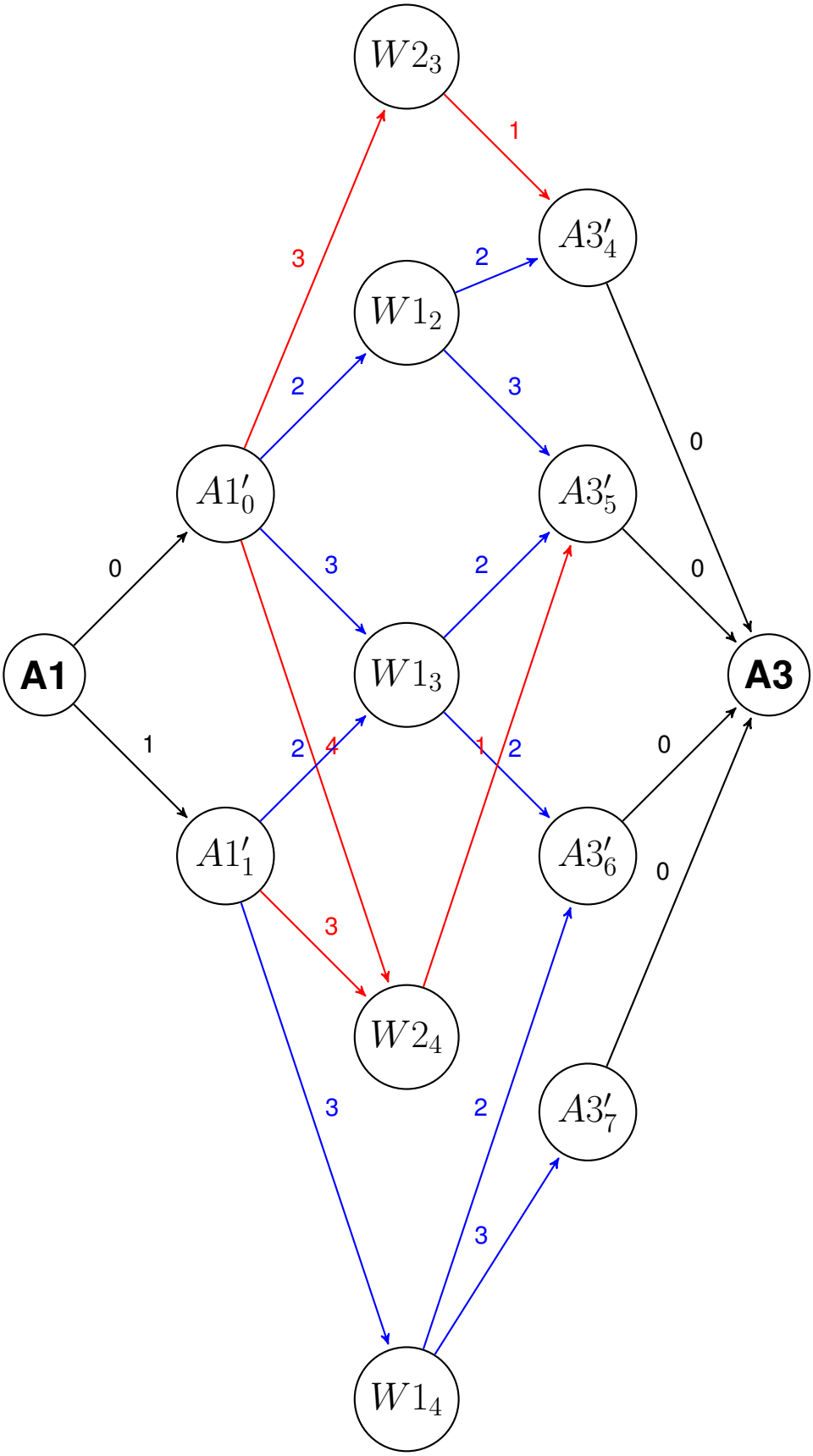

Figura 2.5: Ejemplo grafo recorridos con 2 rutas: ruta 1 en azul y ruta 2 en rojo.

2.7. Modelo base de datos

Como ya se ha explicado, no se ha realizado ninguna modificación sobre la base de datos obtenida en la versión anterior del problema de 2012. El diagrama de entidad relación es el siguiente:

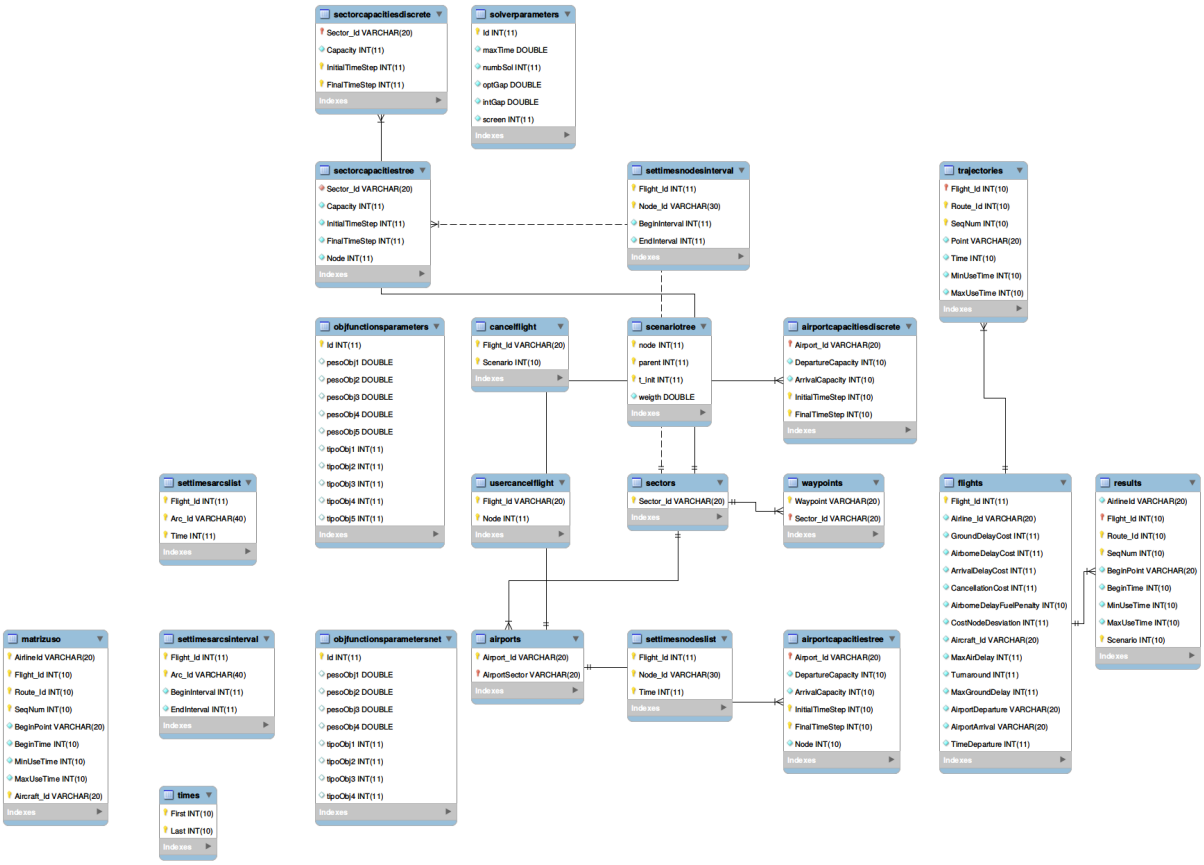

Figura 2.6: Modelo BBDD.

Capítulo 3

Tecnologías utilizadas

Las tecnologías que se han utilizado a lo largo de este Trabajo de Fin de Grado han sido las siguientes:

3.1. C++

Todo el código ha sido desarrollado utilizando el lenguaje C++, creado por Bjarne Stroustrup en 1983. C++ sigue el paradigma de programación imperativa, y es considerado un lenjuaje orientado a objeto híbrido al ser una extensión del lenguaje C. Desde los años 90 se ha mantenido como uno de los lenguajes más utilizados, y actualmente ocupa el puesto 3º en el rankin TIOBE.

Aunque se comenzó utilizando C en las primeras versiones del TFG, finalmente se optó por utilizar C++. Esto fue debido principalmente a 4 factores, que son además las ventajas de C++ sobre C:

- **Permite la orientación a objetos:** a la hora de modelar de nuevo el problema, era necesario crear una estructura de objetos que permitieran un fácil manejo de los datos, y características de la orientación a objetos como la herencia o los constructores permitía manejar la información de forma sencilla y estructurada.
- **Estructuras de datos:** aunque en C también existen, en C++ ya vienen implementadas como parte básica del lenguaje. El uso de estructuras como vectores, tuplas o mapas (y sus métodos) permiten implementaciones más sencillas y eficientes, que reducen en gran medida el tiempo de programación.
- **Sigue permitiendo un manejo de memoria a bajo nivel:** debido a la índole del problema este punto era muy importante, ya que un mal uso de la memoria podría hacer inviables problemas demasiado grandes o con muchas iteraciones.
- **Portabilidad:** el cambio de C a C++ es prácticamente inmediato, por lo que se pudo reaprovechar todo el trabajo realizado.

3.2. MySQL

MySQL es un sistema relacional de gestión de base de datos multiplataforma desarrollado en ANSI C y C++ en 1995 por Michael Widenius. Actualmente es uno de los 3 sistemas de BBDD más utilizados del mundo, junto a Oracle y Microsoft SQL Server.

A pesar de ser un proyecto de Apache, está patrocinado por la empresa Oracle que posee la mayororía de los derechos, lo que ocasionó que en 2009 varios antiguos desarroladores de MySQL crearan MariaDB, un fork[1] de MySQL con licencia opensource.

Al igual que en la versión anterior del proyecto, la BBDD que usamos será la relacional que realizó Diego Ruiz Aguado en el 2012. En esta base de datos se encuentran todos los datos del problema: waypoints, rutas, aeropuertos, vuelos ... etc, los cuales serán utilizados para crear la estructura del problema.

3.3. HTML y JavaScript

HTML (HyperText Markup Language) es un lenguaje de marcado creado por Tim Berners-Lee en 1991 para la creación de páginas web. Es el lenguaje más utilizado para la elaboración de páginas web, además de un estándar a cargo del consorcio WWW. HTML se considera el lenguaje web más importante (entre todos los lenguajes alternativos a HTML no alcanzan el 0.001 % de uso) y ha tenido un impacto muy importante en la expansión del WWW. Entre sus funcionalidades básicas de HTML 5 (la última versión liberada) se encuentran funcionionalidades como añadir audio, vídeo, canvas o *drag and drop*.

JavaScript es un lenguaje de programación interpretado, imperativo, dinámico, débilmente tipado y orientado a objetos. Es parte del estándar ECMAScript, soportado por la gran mayoría de los navegadores desde 2012, lo que lo convierte en el lenguaje más utilizado para el lado cliente en aplicaciones web. JS se basa en el manejo del DOM (Document Object Model) de código HTML y algunas de sus funcionalidades más sencillas podrían ser el de crear contenido interactivo, animaciones o validacuión de formularios. Sin embargo, debido a la popularidad de los últimos años, las funcionalidades de JS, y principalmente, de frameworks para aplicaciones web [2] han hecho que el lenguaje incorporar un gran número de mejoras como peticiones asíncronas, funciones lambda, funciones de orientación a objetos,... etc. Actualmente el ritmo al que se crean nuevas tecnologías basadas en JavaScript es elevadísimo, habiendo multitud de frameworks como AngularJS (desarrollado por Google), ReactJS (desarrolado por Facebook),EmberJS, lenguajes derivados como TypeScript que permiten añadir funcionalidades de tipado, o gran cantidad de librerías como Data-Driven Documents.

Para la representación gráfica del problema se ha utilizado JS para leer y parsear la información almacenada en un fichero y HTML para su visualización (para la creación del grafo se ha utilizado la librería vis.js).

[1] Un fork es la creación de un proyecto software con un nuevo propósito a partir de código ya existente.

[2] Un framework para aplicaciones web es un conjunto de tecnologías, módulos de desarrollo, y capas de abstracción destinadas a facilitar el desarrollo.

3.4. Git y Github

Git es un software de control de versiones distribuido creado por Linus Torvalds en 2005, cuyos dos mayores sistemas de hosting son Bitbucket y Github. Un sistema de control de versiones permite gestionar los cambios llevados a cabo sobre documentos a lo largo del tiempo, así como coordinar el trabajo de diferentes desarrolladores trabajando sobre el mismo documento simultáneamente.

La importancia de Github en los últimos años ha ido en aumento al haberse convertido en el repositorio de proyectos openSource muy importantes como el framework Bootstrap, el lenguaje de lado de servidor Node.js, la librería Jquery o el framework del lenguaje Ruby Rails.

Se ha utilizado Git como sistema de versiones debido a las facilidades que otorga para trabajar desde distintos terminales, así como su fácil manejo de versiones . Todo el código se puede descargar y consultar en Github.

Capítulo 4

Heurístico

El algoritmo heurístico que se ha creado para el problema está inspirado en el metahurístico *GRASP (Greedy Randomized Adaptive Search Procedure)*, sobre el que se han realizado una serie de modificaciones para adaptarlo al modelo en cuestión. A continuación se explicará el funcionamiento de un GRASP, para después explicar el algoritmo propuesto, comparando sus diferencias y semejanzas.

4.1. GRASP

Un GRASP es un metaheurístico constructivo desarrollado inicialmente por T. Feo M. Resende, y se definie como ->cita libro Abraham<- *un GRASP es un procedimiento multiarrranque en el que cada arranque se corresponde con una iteración. Cada iteracción tiene dos fases bien diferenciadas, la fase de construcción, que se encarga de obtener una solución factible de alta calidad; la fase de mejora, que se basa en la optimización (local) de la solución obtenida en la primera fase.*

4.1.1. Fase constructiva

La fase constructiva es un proceso iterativo en el que se construye una solución elemento a elemento. Inicialmente se parte de un componente o conjunto de componentes que conforman una solución parcial, y no serán seleccionables. Posteriormente se ordenan los elementos seleccionables utilizando una función voraz (Greedy), la cual asignará a cada elemento un valor que indique como variaría la función objetivo si se añade a la solución parcial.

Una vez están ordenados todos los elementos seleccionables, hay que seleccionar qué elemento es añadido a la solución parcial. GRASP no añade el mejor candidato posible, ya que esto no garantiza que se obtenga la solución óptima, sino que se elije aleatoriamente un candidato del conjunto de candidatos restringido (o RCL por sus siglas en inglés). Para crear esta RCl, se crea un conjunto de candidatos que cumplan

$$RCL_{umbral} = (c_{min} + \alpha(c_{max} - c_{min})) \tag{4.1}$$

, donde c_{min} y c_{max} son respectivamente los valores más bajo y más alto del coste asignado a los elementos seleccionables, y el parámetro α que determina el umbra permitido oscila entre $0 \leq \alpha \leq 1$, de forma que se escoge un candidato aleatorio. Si fijamos $\alpha = 1$, en la RCL solo contaríamos con el mejor candidato, y sería una función miope pura. Por el contrario si fijamos $\alpha = 0$, serán seleccionados todos los candidatos.

Una vez seleccionado el elemento o elementos, se introducen en la solución parcial y se les marca como no seleccionables. El resto de candidatos siguen siendo seleccionables, por tanto cuando se les vuelva a evaluar para ser candidatos, sus valores serán distintos a los de la iteración anterior, ya que la solución parcial ha variado al haber añadido el último elemento conjunto de candidatos.

La fase constructiva finaliza cuando se dispone de una solución factible.

4.1.2. Fase de mejora

Pero la fase constructiva no garantiza que la solución sea óptima respecto a su vecindad. Por ello GRASP incorpora la fase de mejora, que consiste en un procedimiento de optimización local, el cual puede ser otra metaheurística o una función de búsqueda local.

4.2. Heurístico implementado

4.2.1. Introducción

El algoritmo diseñado para este problema se resume en el siguiente pseducódigo:

```
inizializarProblema();
while N < iteracionesMáximas do
    añadirVuelosEnColaCandidatos();
    lanzarVuelosSoloSolucionesIniciales();
    intercambiarVuelos();
    lanzarVuelosPermitiendoRetrasos();
    lanzarVuelosPermitiendoDesvíos();
    buscarWaypointsSinUsar();
    retrasarVuelosConSolución();
    if N %númeroSolucionesExaminar == 0 then
        crearColaCandidatos();
    end
end
```

Algorithm 1: Esquema algoritmo implementado

Como se comentó anteriormente, al igual que un algoritmo GRASP tradicional, se compone de una fase constructiva en la que se obtiene una solución de alta calidad y una fase de mejora que se produce cada N iteraciones, en la cual se analizarán las soluciones anteriores para

seleccionar los vuelos más prometedores y añadirlos al problema, de forma que tengan más posibilidad de ser seleccionados antes. A continuación se explican con más detalle ambas fases.

4.2.2. Fase constructiva

Dado que este problema tiene siempre una solución factible (cancelar todos los vuelos), el objetivo de esta fase es conseguir una solución de alta calidad. Esta fase consta de 6 subrutinas.

1. **Lanzar vuelos con las soluciones iniciales:** se lanzan todos los vuelos de manera aleatoria sin permitir retrasos o desvíos, la única ruta que se permite es la solución por defecto.

   Tras realizar varias pruebas, se ha podido comprobar que los pasos siguientes del algoritmo dependen en gran medida de los vuelos a los que se encuentra solución en esta fase. Esto se debe a que un "mal" vuelo colocado en su solución inicial puede sobrecargar sectores clave para otros muchos vuelos.

   En las pruebas que hemos realizado, en esta fase se colocan con éxito entre un 5 % y un 15 % de los vuelos.

2. **Intercambio de vuelos:** se intenta sustituir uno de los vuelos exitosos del paso anterior por 2 o más vuelos aleatorios a los que no se les halló solución.

   Este paso fue introducido para paliar el problema que se indicaba en el paso anterior, y se realiza partiendo de una idea básica: si cancelando un vuelo que teníamos con solución conseguimos colocar con éxito 2 o más vuelos, será siempre una mejora.

   Por ejemplo, si en la fase anterior fue colocado con éxito un vuelo de la siguiente forma:

Figura 4.1: Ejemplo vuelo colocado en la fase 2.

   podría cancelarse para colocar 2 vuelos más cortos:

Figura 4.2: Resultado tras el intercambio de vuelos.

3. **Se intentan colocar vuelos permitiendo retrasos**: se lanzan aleatoriamente los vuelos que aun no tienen solución, permitiendo retrasos en sus rutas, pero no desvíos. En el problema se considera que un vuelo retrasado es preferible a un vuelo desviado, así que primero se intenta encontrar soluciones que no conlleven desvíos.

   En este paso se colocan ente un 50 % y un 65 % de los vuelos.

4. **Se intentan colocar vuelos permitiendo retrasos y desvíos**: se lanzan aleatoriamente los vuelos que aun no tienen solución, permitiendo retrasos y desvíos en sus rutas. En este punto si un vuelo tiene alguna solución factible, se le asignará.

   Se colocan con éxito entre el 20 % y el 30 %.

5. **Se buscan los waypoints sin usar y se les intenta asignar una ruta**: se localizan los waypoints por los que no pasa ningún vuelo a lo largo de todo el problema. Si algún vuelo tiene alguna solución factible que utilice alguno de estos waypoints, se la asigna.

   Con los modelos de ejemplo con los que contábamos esta fase proporciona mejoras en solo un 5 % de los casos.

6. **Retrasar vuelos con solución para colocar 2 o más cancelados**: se intenta retrasar alguno de los vuelos con solución factible para poder encontrar de forma aleatoria uno o más vuelos que estaban cancelados.
   Este paso se basa en el mismo concepto que en el intercambio de vuelos: si a costa de empeorar la solución de vuelo (ya sea en su solución inicial o retrasado) se consigue encontrar la solución de 1 o más vuelos cancelados, se esta mejorando el resultado global.

4.2.3. Fase de mejora

El objetivo de esta fase es analizar las últimas N iteraciones, y escoger de entre ellas a los mejores vuelos de forma aleatoria para que en las siguientes N iteraciones tengan más probabilidades de ser lanzados antes. Aquí radica la mayor diferencia con un GRASP: en el metahurístico constructivo hay que evaluar y seleccionar de los candidatos disponibles a uno o varios de ellos según la ya citada fórmula

$$RCL_{umbral} = (c_{min} + \alpha(c_{max} - c_{min})) \tag{4.2}$$

para a continuación volver a evaluar a la lista de candidatos disponibles y repetir el proceso con la solución parcial actualizada. En este algoritmo se analizan las iteraciones anteriores y se trata de extraer lo mejor de ellas. El proceso se realiza en 2 fases:

1. **Analizar las soluciones previas:** de las N iteraciones anteriores, calculamos el valor de la función objetivo para cada una de ellas.

2. **Selección proporcional de candidatos:** una vez asignado un valor a cada iteración, se seleccionan vuelos que fueron colocados o en su solución inicial o retrasados muy poco de forma proporcional al valor de la solución. El número máximo de vuelos que se seleccionan está marcado por el parámetro G.

   Por ejemplo, si tenemos $N = 3$ y $G = 3$, y las soluciones anteriores a, b, c tienen funciones objetivo con valores de $100, 200, 300$ respectivamente, se escogerán vuelos de la solución a con una probabilidad de $1/6$, de la solución b $2/3$ y de la solución c con $3/6$

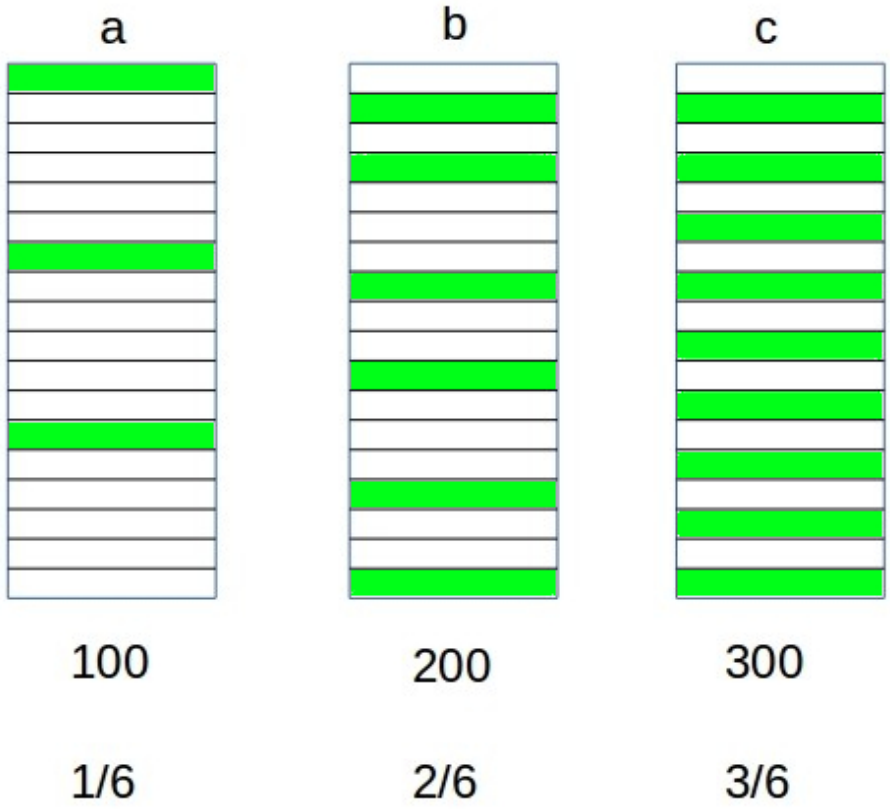

Figura 4.3: Elección ponderada de candidatos.

3. **Añadir vuelos a la cola de candidatos:** los vuelos seleccionados en la fase anterior conforman la cola de vuelos extras.

   Cada N iteraciones, antes de realizar la fase constructiva, se añadirán los vuelos de la cola de candidatos al resto de vuelos que conforman el problema. El proceso que se realiza es el siguiente:

   a) Si no hay cola de candidatos, se obtiene el id de cada vuelo del problema y se ordenan de forma aleatoria:

   $$\{1, 2, 3, 4, 5\} \Rightarrow \{2, 4, 5, 1, 3\} \tag{4.3}$$

   b) Si por el contrario sí que tenemos una cola de candidatos, la juntaremos con la cola de vuelos habituales. Posteriormente se ordenan y se descartan los elementos repetidos, de forma que los vuelos que añadimos de la cola de candidatos, al estar

por duplicado o incluso triplicado, tienen altas probabilidades de quedar colocados los primeros de la cola resultante.

Por ejemplo, si en el caso anterior obtenemos que hemos seleccionado los vuelos 2, 4 para añadir al problema, el orden para lanzar los vuelos en los siguientes pasos del algoritmo se calculará de la siguiente manera

$$\begin{gathered}\{1,2,3,4,5\}+\{2,4\}=\{1,2,3,4,5,2,4\}\\ \{1,2,3,4,5,2,4\}\Rightarrow\{2,5,4,3,1\}\end{gathered} \tag{4.4}$$

4. **Descarte de mala solución** si la mejor de las N soluciones candidatas que se analizan no supera el valor de la mejor solución encontrada hasta el momento, se descarta la cola actual y se reinicia el problema. Si se da este caso, indicará que la última cola de candidatos que seleccionamos no ha conseguido mejorar la solución del problema, por tanto podemos descartarla.

4.2.4. Parámetros heurístico

Por tanto el heurístico depende de dos parámetros:

- N: cada cuantas iteraciones actualizamos la cola de vuelos adicional.
- G: el tamaño máximo de la cola de vuelos adicionales.

Tras realizar pruebas sobre distintos problemas, se ha obtenido que las mejores soluciones se tienen a alcanzar con un parámetro N pequeño y G grande, de forma que la cola de vuelos candidatos sea lo más grande posible y que la fase constructiva se realice con mucha frecuencia. Los resultados se pueden observar en chapter 5.

De esta forma, un alto parámetro G permitirá que muy probablemente muchos de los vuelos que en la mejor solución encontrada hasta ese momento fueron colocados con éxito, se lance, intentando así reproducir"la mejor solución que se había encontrado.

Por contra un bajo parámetro N hará que se cree cada pocas iteraciones una nueva cola de vuelos candidatos, lo que permitirá que en caso de no encontrase rápidamente una solución mejor, se elimine la cola actual y se reinicie el problema.

4.3. Encontrar ruta óptima

En muchas de las fases del algoritmo es necesario encontrar la ruta óptima para un vuelo. Debido a que es el núcleo del algoritmo, vamos a explicarlo con más detalle.

4.3.1. Pre solución: creación del grafo

El primer paso es crear el grafo de rutas que tiene un vuelo. La creación de este grafo va a depender del caso en el que estemos: por ejemplo, si estamos en la fase *lanzarVuelosSoloSolucionesIniciales()*, solo crearemos un grafo con nodos que simulen la ruta inicial (sin retrasos ni rutas alternativas), o si estamos en la fase *lanzarVuelosPermitiendoRetrasos()* crearemos solo un grafo en el que haya nodos que pertenezcan a la ruta original.

Esto implica que dependiendo de la fase en la que estemos, el grafo de un vuelo será diferente. Esta decisión de diseño alivia en gran medida el esfuerzo computacional del algoritmo de Djistra del siguiente paso, ya que el grafo para el que ejecutamos el algoritmo tiene solo soluciones válidas, excluyendo así caminos que en dicha fase no estarían permitidos, y por tanto solo aumentarían el tiempo de ejecución sin llegar a aportar una solución válida.

Además, a la hora de crear el grafo se tienen en cuenta las condiciones y restricciones de cada instante de tiempo. Por ello, si un waypoint en un momento de tiempo fuera a ser añadido a la lista de waypoints pero no cumple con las restricciones del problema (ese arco ya está ocupado, o su sector está ya al máximo de capacidad), ese nodo no se añadirá. Esto implica que se pueden crear grafos no conexos, y por tanto sin solución. Si se da este caso, el vuelo se marcará como cancelado.

4.3.2. Djistra

Una vez creado el grafo sobre el que tenemos que encontrar el camino más corto entre 2 puntos, ejecutamos un algoritmo de Djistra con lista, que con una complejidad de $O(n^2)$, y dado que el número n de nodos no va a ser muy elevado, permite que las operaciones se hagan con la rapidez suficiente.

4.3.3. Post solución: actualización de las restricciones

Tras ejecutar el algoritmo de Djistra, podemos obtener dos resultados:

- **Se encuentra el camino más corto:** obtenemos los waypoints por el que pasa el camino más corto, así como su longitud. Con estos datos tenemos que actualizar las restricciones del problema para cada instante de tiempo t en el que el vuelo ha estado en el aire. El estado del vuelo dependerá de la fase en la que estamos, si estamos en *lanzarVuelosSoloSolucionesIniciales()*, el estado del vuelo pasará a ser solución inicial, mientras que si estamos en *lanzarVuelosPermitiendoDesvíos()* el estado del vuelo será desviado y retrasado, o tan sólo retrasado, dependiendo de si la solución encontrada es más o igual de larga que su solución iniciale

- **No se encuentra solución:** este caso puede darse cuando debido a las restricciones del problema el grafo creado no es conexo. El vuelo se marca como cancelado.

Capítulo 5

Resultados experimentales

En este capítulo se resumen los resultados obtenidos. Los detalles de los experimentos se pueden consultar en chapter 7.

5.1. Comparación de parámetros G y N

Como se explicó en el chapter 4, el algoritmo desarrollado depende de dos parámetros: G, la frecuencia con la que realizamos la fase constructiva; y N, el número máximo de elementos que puede contener la cola de vuelos extras.

Tras realizar varias pruebas, hemos podido comprobar que, como era de esperar, los mejores resultados se obtienen con un parámetro N pequeño y G grande:

- Un ***N* pequeño** implica realizar la fase constructiva con mucha frecuencia. Esto hace que cada pocas iteraciones busquemos un intento de mejora, y por tanto en caso de no conseguir mejorar la solución actual descartaremos la cola de vuelos extras con mucha rapidez.

  Un N pequeño también aporta más desviación a los resultados, ya que al descartar colas con mucha facilidad, hace que el problema se reinicie con más frecuencia, y por tanto tenga más aleatoriedad.

- Un ***G* grande** implica que se de más peso a las buenas soluciones que se localizaron en la cola anterior. Por tanto al tener una cola de vuelos extras de gran tamaño, permite que las buenas soluciones tengan una probabilidad mucho mayor que salir antes en el proceso aleatorio de selección de orden de los vuelos para intentar ser colocados.

Concretamente, hemos podido comprobar mediante pruebas con diferentes parámetros que los la configuración óptima de los parámetros es la siguiente:

- **N:** si es demasiado pequeño, el algoritmo no dispone de las iteraciones suficientes para tratar de obtener información relevante para futuras simulaciones, por lo que el valor óptimo del parámetro es un 5 % del total de iteraciones. De esta forma, en un ejemplo sobre el que realizamos 100 iteraciones, realizaremos la fase constructiva en 20 ocasiones.

- **G:** aunque el algoritmo funciona mejor con un parámetro *G* elevado, los valores óptimos se alcanzan cuando es un 50 % sobre el número total de vuelos. Por tanto si disponemos de un modelo con 100 vuelos, la cola máxima de vuelos que seleccionaremos para las siguientes iteraciones será de como mucho 50. De esta forma se evita que las iteraciones se parezcan demasiado entre si.

5.2. Evolución de la función objetivo

Como ya comentamos, con la configuración de parámetros G y N al descartarse muy rápido las malas combinaciones obtenemos un gran número de reseteos del problema, por lo que obtenemos una desviación alta. En la Figure 5.1 se muestra la evolución del valor de la función objetivo en un ejemplo de 100 vuelos con parámetro $N = 5$:

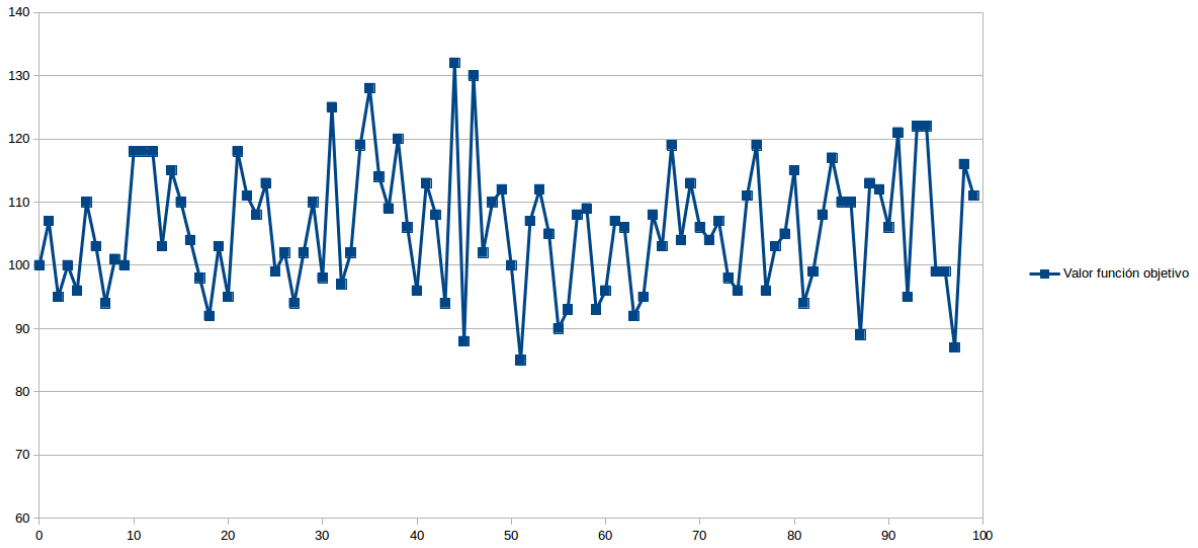

Figura 5.1: Evolución función objetivo con $N = 5$ y $G = 20$ con ejemplo de 100 vuelos

Como se puede comprobar, en las iteraciones en las que se produce la fase de evaluación y se decide si se descarta la cola, se pueden dar importantes saltos. Por ejemplo, en la Figure 5.1 se puede observar como en la iteración con $x = 40$ ha habido un descarte de la cola que se había elegido en la anterior fase de mejora, ya que en la siguiente iteración ha habido una gran mejora.

Por el contrario en las iteraciones $x = 60$ o $x = 80$ se puede observar que no se ha descartado la cola de candidatos, ya que la mejora en la función objetivo ha sido mas leve.

5.3. Resultados globales

A continuación se muestran resumidos los resultados de las pruebas realizadas en los 3 problemas que disponíamos para probar el algoritmo. Los valores se corresponden con la simulación de 100 problemas de 100 iteraciones cada uno con los parámetros G y N óptimos:

Cuadro 5.1: Resultados pruebas 20 vuelos

	% Vuelos con solución	Valor función objetivo
Caso A	99.61 %	46.8
Caso B	100 %	47
Caso C	100 %	45

Cuadro 5.2: Resultados pruebas 100 vuelos

	% Vuelos con solución	Valor función objetivo
Caso A	61.69 %	118.57
Caso B	93.73 %	200.88
Caso C	???	???

Como se puede observar, exceptuando el caso de pruebas *A* que tiene un gran número de vuelos iguales simultáneos, los resultados se mueven siempre en porcentajes mayores al 90 %.

5.4. Representación gráfica

A continuación se muestra la representación gráfica obtenida para un problema con 20 vuelos.

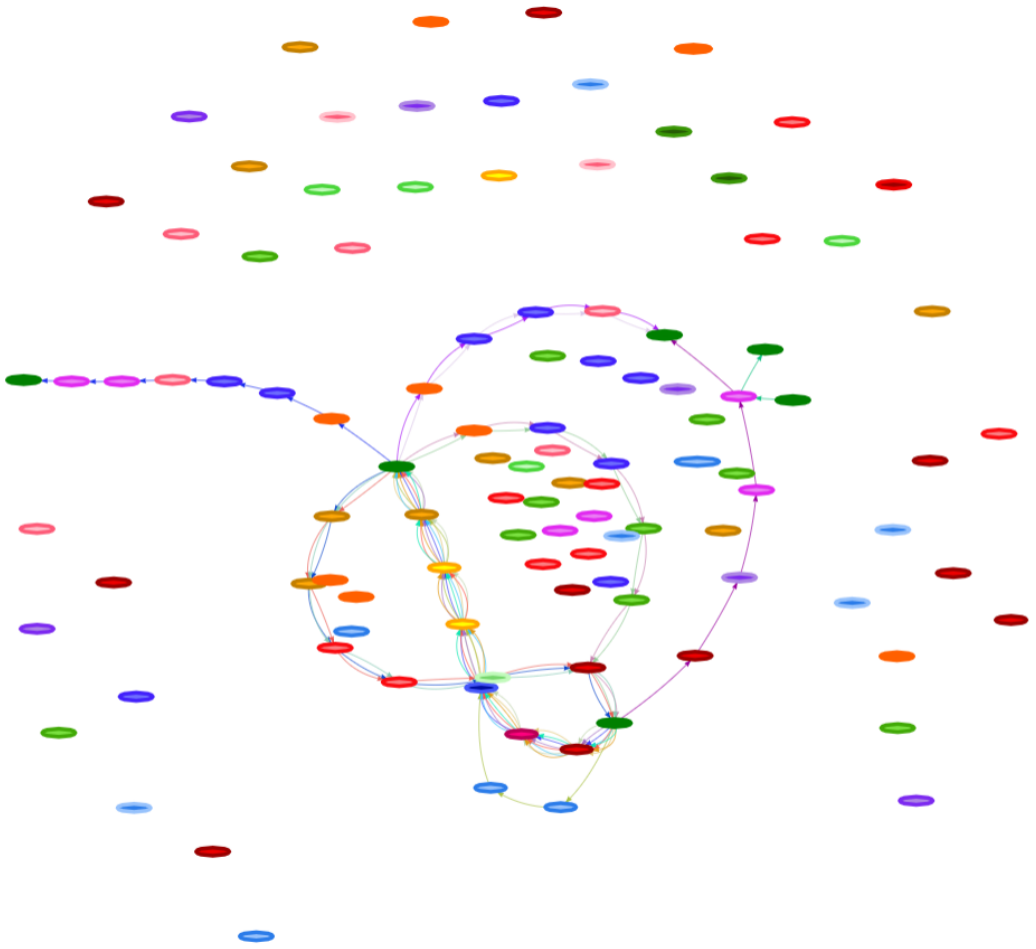

Figura 5.2: Representación gráfica de un problema de 20 vuelos

La representación gráfica muestra el resultado final del problema, es decir, se van a representar todos los vuelos a los que se les encontró solución independientemente de los instantes de tiempo en los que estuvieron en tránsito. Por tanto la representación gráfica nos permite observar qué rutas van a ser las más utilizadas.

Algunas características de la representación gráfica:

- Sólo muestra los vuelos a los que se les encontró solución.
- Los waypoints que están ubicados en el mismo sector se marcan del mismo color (si un waypoint pertenece a más de un sector se escoge solamente el primero para simplificar).
- Cada vuelo tiene asignado un color para poder identificar su ruta.
- En las aristas del grafo representado se indica el instante de tiempo en el que un vuelo realiza ese trayecto

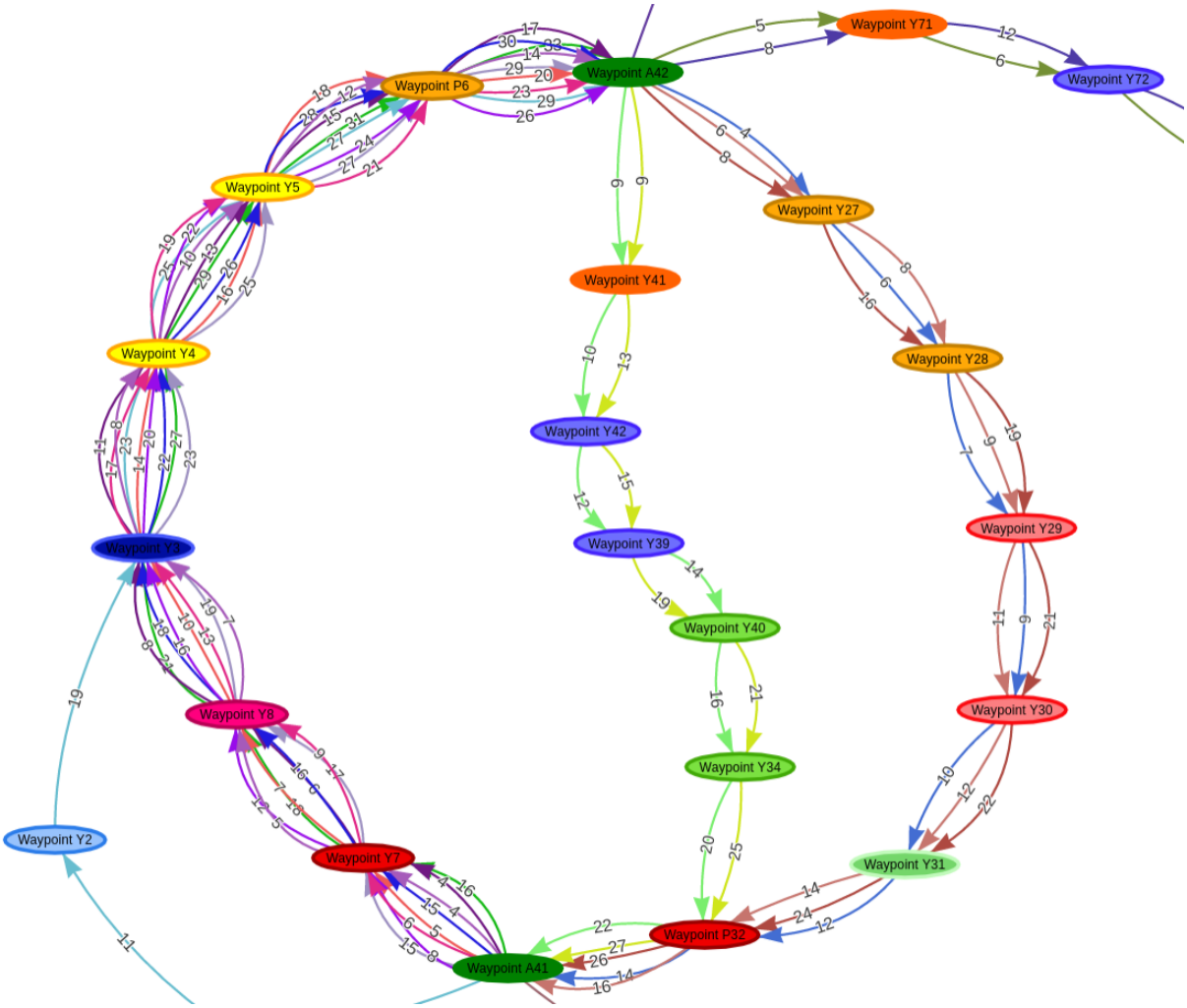

Figura 5.3: Ejemplo rutas entre 2 aeropuertos

En la Figure 5.3 se puede observar que en el trayecto entre el *Aeropuerto 42* y el *Aeropuerto 41* existen 2 posibles rutas: la rama derecha (más rápida y por tanto con más vuelos) y la rama entral (la ruta alternativa, ya que es más lenta).

5.5. Tiempo de ejecución

Debido al gran volumen de información se maneja el problema, comentamos brevemente el tiempo que equiere el algoritmo. Todas las pruebas han sido realizadas en un Acer Aspire-E5-573G con las siguientes características:

1. Sistema operativo Ubuntu 16.04.2 LTS 64-bit.

2. 8GB de memoria RAM.

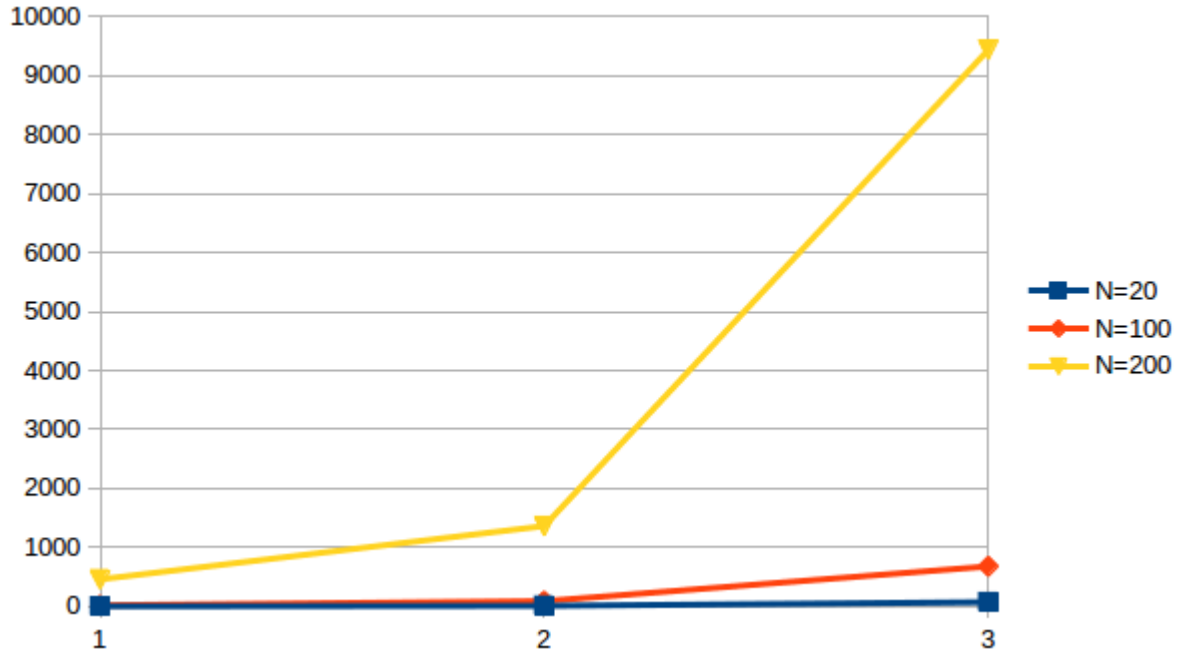

Figura 5.4: Tiempos de ejecución del programa en segundos

3. Procesador Intel Core i5-4210U CPU 1.70GHz x 4.

4. tarjeta gráfica GeForce 920M/PCIe/SSE2.

Los tiempos de ejecución en segundos para un problema de N vuelos se resumen en la siguiente tabla y I iteraciones se resumen en la Table 5.3 y en la Figure 5.4:

Cuadro 5.3: Tiempos de ejecución en segundos

	N=20	**N=100**	**N=800**
I=10	3	21	460
I=10	11	91	1363
I=100	74	680	9436

Capítulo 6

Conclusiones

6.1. Valoración de los resultados

Al final del Trabajo de Fin de Grado se han cumplido todos los objetivos que se establecieron en su momento:

- Modelar de nuevo el problema sin que éste dependiera de CPLEX.
- Mejorar el sistema de lectura de datos.
- Mejora del heurístico para lanzar los vuelos.
- Nueva estructura de objetos.
- Creación de una simple representación visual del problema.

Aunque se han cumplido todos los objetivos cumplidos, podemos comparar los resultados que se obtuvieron en la versión anterior del problema con los de este TFG: en la versión de junio de 2012 con una base de datos de 65 vuelos y las capacidades de los sectores a 1, obtuvieron un porcentaje de vuelos cancelados del 11 %. Con el nuevo algoritmo lo reducimos a un 3 %.
Por tanto podemos concluir que la implementación del nuevo heurístico a mejorado notablemente la solución, y la estructura de búsqueda multiarrranque acompañada de una posterior fase de retrospectiva donde se extraen las características positivas de soluciones previas han sido beneficiosas para el modelo.
También se ha enriquecido el problema añadiendo a todos las fases del heurístico un componente aleatorio, dando un gran número de casos de los que extraer información para futuras iteraciones.

6.2. Futuras líneas de trabajo

Las siguientes versiones del algoritmo podrían añadir algunas funcionalidades extra:

- Para enriquecer más el modelo, se podrían añadir algunas de las restricciones que se eliminaron para simplificar el modelo. La más importante sería la de añadir el coste de cancelación de un vuelo, de forma que a la hora de elegir entre 2 vuelos similares se tenga en cuenta el coste de cancelación.
- Considerar la cancelación de un vuelo como prácticamente inviable para acercar más el modelo a la realidad.
- Optimizar los algoritmos de búsqueda para reducir el tiempo de ejecución del problema.

Capítulo 7

Anexo 1: resultados experimentales

7.1. Comparación parámetros G y N

A continuación se muestran los resultados de las pruebas que hemos realizados variando los parámetros G y N en los tres escenarios de prueba que disponíamos. En la Table 7.1 se muestran la media de parámetros obtenidos tras realizar 25 experimentos sobre cada caso de prueba.

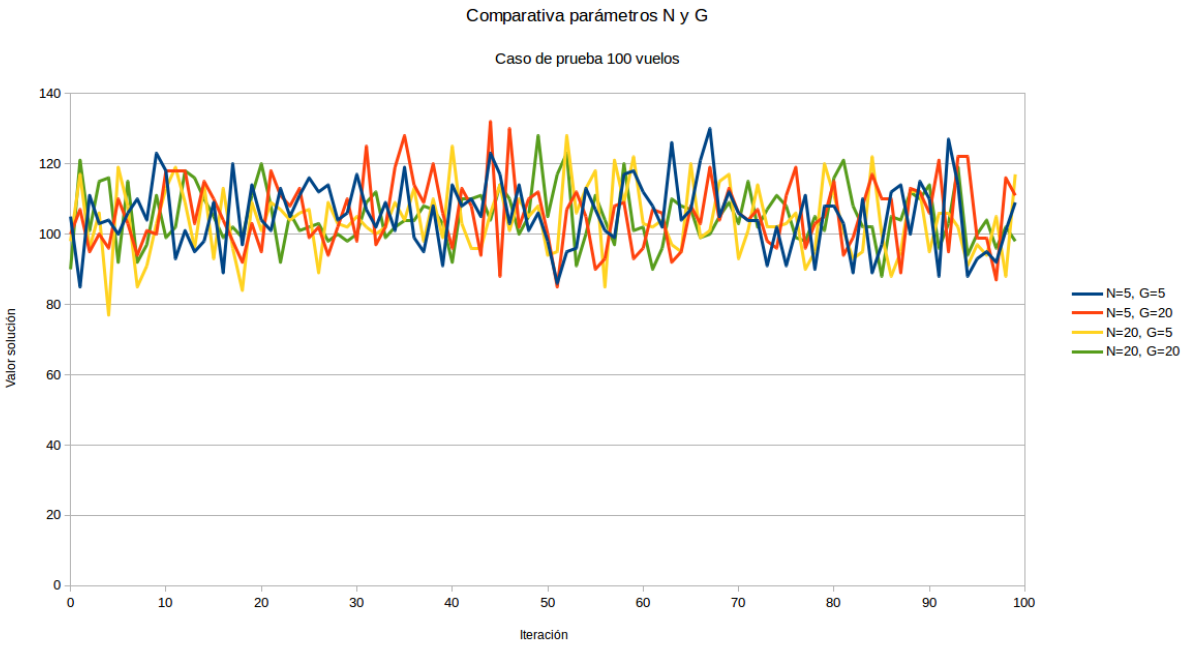

Figura 7.1: Evolución función objetivo con diferentes parámetros G y N

Cuadro 7.1: Comparativa de parámetros G y N

	Problema A	**Problema B**	**Problema C**	**Media porcentaje éxito**
N=2, G=10	% éxito: 64.1	% éxito: 94.2	% éxito: 99.5	85.93 %
N=2, G=30	% éxito: 65.2	% éxito: 95.1	% éxito: 99.5	86.60 %
N=2, G=50	% éxito: 66.1	% éxito: 95.6	% éxito: 99.6	87.10 %
N=5, G=10	% éxito: 65.2	% éxito: 94.3	% éxito: 99.4	86.30 %
N=5, G=30	% éxito: 65.7	% éxito: 95.7	% éxito: 99.8	87.06 %
N=5, G=50	% éxito: 66.8	% éxito: 95.6	% éxito: 99.8	87.40 %
N=10, G=10	% éxito: 64.2	% éxito: 94.1	% éxito: 99.3	85.86 %
N=10, G=30	% éxito: 65.2	% éxito: 95.1	% éxito: 99.5	86.60 %
N=10, G=50	% éxito: 65.6	% éxito: 95.1	% éxito: 99.6	86.76 %

Las características de las simulaciones corresponden a las siguientes:

- Las pruebas se han realizado sobre los 3 problemas disponibles.
- En todos los experimentos se considera el peor escenario: la capacidad de los sectores está limitada a 1.
- Se realizan 100 iteraciones sobre cada experimento.
- El parámetro G

7.2. Resultados simulación

7.2.1. Caso de pruebas A

20 vuelos

Iteración	**Valor de la función objetivo**	**% Vuelos con solución**
0	46	100 %
1	46	100 %
2	48	100 %
3	47	100 %
4	46	100 %
5	48	100 %
6	48	100 %
7	47	100 %
8	47	100 %
9	46	100 %

10	46	100 %
11	46	100 %
12	46	100 %
13	48	100 %
14	47	100 %
15	47	100 %
16	47	100 %
17	45	95 %
18	47	100 %
19	47	100 %
20	48	100 %
21	45	95 %
22	47	100 %
23	47	100 %
24	47	100 %
25	48	100 %
	Media: **46.8**	Media: **99.61** %

Cuadro 7.2: Resultados problema A 20 vuelos

100 vuelos

Iteración	**Valor de la función objetivo**	**% Vuelos con solución**
0	124	65 %
1	112	61 %
2	120	63 %
3	122	64 %
4	122	62 %
5	120	60 %
6	114	58 %
7	116	62 %
8	127	55 %
9	119	61 %
10	121	63 %
11	111	57 %
12	115	60 %
13	114	58 %
14	121	62 %
15	110	60 %
16	119	61 %

17	124	65 %
18	118	65 %
19	112	60 %
20	112	60 %
21	131	68 %
22	116	60 %
23	123	64 %
24	121	65 %
25	119	61 %
	Media: **118.57**	Media: **61.69** %

Cuadro 7.3: Resultados problema A 100 vuelos

7.3. Caso de pruebas B

20 vuelos

Iteración	**Valor de la función objetivo**	**% Vuelos con solución**
0	47	100 %
1	47	100 %
2	47	100 %
3	47	100 %
4	47	100 %
5	47	100 %
6	47	100 %
7	47	100 %
8	47	100 %
9	47	100 %
10	47	100 %
11	47	100 %
12	47	100 %
13	47	100 %
14	47	100 %
15	47	100 %
16	47	100 %
17	47	100 %
18	47	100 %
19	47	100 %
20	47	100 %

21	47	100 %
22	47	100 %
23	47	100 %
24	47	100 %
25	47	100 %
	Media: **47**	Media: **100** %

Cuadro 7.4: Resultados problema B 20 vuelos

100 vuelos

Resultados del problema 2 con 100 vuelos:

Iteración	**Valor de la función objetivo**	**% Vuelos con solución**
0	200	93 %
1	202	96 %
2	204	96 %
3	203	96 %
4	203	95 %
5	198	93 %
6	201	93 %
7	201	94 %
8	197	93 %
9	199	92 %
10	199	93 %
11	199	93 %
12	200	92 %
13	199	93 %
14	198	92 %
15	202	94 %
16	205	96 %
17	214	96 %
18	199	93 %
19	202	96 %
20	200	93 %
21	201	93 %
22	200	93 %
23	198	92 %
24	200	93 %
25	199	93 %
	Media: **200.88**	Media: **93.73** %

Cuadro 7.5: Resultados problema B 100 vuelos

7.3.1. Caso de pruebas C

20 vuelos

Resultados del problema 3 con 20 vuelos:

Iteración	**Valor de la función objetivo**	**% Vuelos con solución**
0	45	100 %
1	45	100 %
2	45	100 %
3	45	100 %
4	45	100 %
5	45	100 %
6	45	100 %
7	45	100 %
8	45	100 %
9	45	100 %
10	45	100 %
11	45	100 %
12	45	100 %
13	45	100 %
14	45	100 %
15	45	100 %
16	45	100 %
17	45	100 %
18	45	100 %
19	45	100 %
20	45	100 %
21	45	100 %
22	45	100 %
23	45	100 %
24	45	100 %
25	45	100 %
	Media: **45**	Media: **100** %

Cuadro 7.6: Resultados problema C 20 vuelos

100 vuelos

Resultados del problema 3 con 100 vuelos:

Iteración	**Valor de la función objetivo**	**% Vuelos con solución**
0	218	100 %
1	218	100 %
2	218	100 %
3	218	100 %
4	218	100 %
5	218	100 %
6	218	100 %
7	218	100 %
8	218	100 %
9	218	100 %
10	218	100 %
11	215	99 %
12	218	100 %
13	218	100 %
14	218	100 %
15	218	100 %
16	218	100 %
17	218	100 %
18	218	100 %
19	218	100 %
20	218	100 %
21	218	100 %
22	218	100 %
23	218	100 %
24	218	100 %
25	218	100 %
	Media: **217.88**	Media: **99.96** %

Cuadro 7.7: Resultados problema C 100 vuelos

Capítulo 8

Anexo 2: Estructura de clases

Para diseñar la estructura del modelo, se ha creado un esquema de clases cuyas entidades y sus atributos son:

8.1. Flight

Es la clase que se encarga de representar a los vuelos en el problema.

- **id**: identificador del vuelo.
- **timeStart**: instante de tiempo en el que el vuelo tiene planeado el despegue.
- **groundDelay**: matriz de costes del grafo de recorridos.
- **routes**: lista de los waypoint route que possee el vuelo.
- **numWaypoint**: número de waypoint que tiene un vuelo.
- **idWaypointStart**: identificador del waypoint en el que se encuentra el aeropuerto de salida.
- **idWaypointEnd**: identificador del waypoint en el que se encuentra el aeropuerto de llegada.
- **status**: estado del vuelo.
- **numRoutes**: número de rutas de las que dispone el vuelo
- **timeFinish**: tiempo de llegada al aeropuerto de destino en su ruta por defecto.
- **waypointnames**: listado de nombres de los waypoints que recorre en todas sus rutas.
- **waypointRoute**: listado de los waypointRoutes que tiene el vuelo
- **numWaypointRoute**: número de waypoint routes que tiene el vuelo.

- **initialSolution**: listado de waypoint route que corresponde a la solución por defecto del vuelo.
- **currentSolution**: solución que tiene el vuelo en cada iteración.
- **flightInterchangeCandidates**: listado de vuelos con sol que el vuelo comparte sectores.

8.2. Problem

Representa las características del problema

- **numAirports**: número de aeropuertos que tiene el problema.
- **numSectors**: número de sectores que tiene el problema.
- **numTrajectories**: número de distintas trayectorias que en total tienen todos los vuelos del problema.
- **numWaypoints**: número de waypoints que tiene el problema.
- **numFlights**: número de vuelos que tiene el problema.
- **numTimes**: número de instantes de tiempo que se analizan en el problema.
- **listWaypoints**: listado de los diferentes waypoints de los que se compone el problema.
- **listSectors**: listado de los diferentes sectores de los que se compone el problema.
- **listTimeMoment**: listado de las capacidades y estado de los vuelos en los diferentes instantes de tiempo.
- **listFlights**: listado de los diferentes vuelos de los que se compone el problema.
- **iteration**: iteración en la que se encuentra el problema.
- **log**: documento de teto en el que se guarda un registro de las operaciones que se producen.
- **queueExtraFlights**: cola que contendrá los vuelos seleccionados durante la fase constructiva del algoritmo.
- **solutions**: lista de las diferentes soluciones (estado de cada vuelo y su valor en la función objetivo) obtenidas en cada iteracción.
- **valueBestSolution**: valor de la mejor solución encontrada hasta el momento.
- **routeFlightResults**: ruta del documento en el que se volcarán los datos para su posterior visualización.

8.3. Sector

- **id**: identificador del sector.
- **name**: nombre del sector.
- **capacity**: capacidad del sector. En nuestro problema todos los sectores tienen capacidad 1.

8.4. Solution

Representa la solución obtenida en cada iteración del problema.

- **flightSolutions**: es un mapa de la forma vuelo =>solución del vuelo.
- **value**: valor que tiene la solución. Se obtiene sumando el valor de todos los vuelos de una iteración

8.5. TimeMoment

Representa las capacidades del problema para cada instante de tiempo.

- **numFlightMatrix**: es una matriz en la que se indica en que arco de su grafo de recorridos se encuentra cada vuelos. Se utiliza para controlar que no haya 2 vuelos en el mismo instante de tiempo entre 2 waypoints.
- **numFlightsSector**: indica el número de vuelos que hay en cada sector en un momento de tiempo

8.6. Wapoint

- **id**: identificador del waypoint.
- **name**: nombre del sector.
- **sectors**: listado de los sectores a los que pertenece el waypoint.
- **isAirport**: booleano que indica si el waypoint es un aeropuerto.

8.7. WaypointRoute

Representa a los puntos de ruta que posee cada vuelo. Están formados por un waypoint y un instante de tiempo.

- **id**: identificador del waypointRoute.
- **inTime**: momento de tiempo en el que se da l waypointRoute