
**Universidad** de **Valladolid**
Escuela de Ingeniería Informática de Segovia

## Trabajo Fin de Grado

Grado en Ingeniería Informática de Servicios y Aplicaciones

---

# Reconstrucción de trayectorias de aeronaves usando heurísticas de mejora para resolver una versión del *problema del viajante* (TSP)

---

**Alumno: Juan Manuel Velasco Heras**

**Tutores: Anibal Bregón Bregón,**
**Miguel Ángel Martínez Prieto**

# Reconstrucción de trayectorias de aeronaves usando heurísticas de mejora para resolver una versión del *problema del viajante* (TSP)

Juan Manuel Velasco Heras

# Agradecimientos

A Boeing Research and Technology Europe por poner a disposición del proyecto datos e información propia, a mi compañera de estudios y promoción María Zorita Mínguez por su colaboración, a los profesores D. Aníbal Bregón Bregón y D. Miguel Ángel Martínez Prieto por haber aceptado la dirección de este Trabajo Fin de Grado y al profesor D. Pedro César Álvarez Esteban por sus valiosas aportaciones y directrices.

# Resumen

El tráfico aéreo mundial está cada vez más congestionado. Solo en Europa, Eurocontrol pronostica para 2023 un aumento del 14 % en el número de vuelos que transitarán por su espacio aéreo con respecto a 2016, lo que exige una profunda renovación de los sistemas de gestión actuales apostando por tecnologías *bigdata*. El proyecto AIRPORTS (CIEN, 2015), liderado por *Boeing Research & Technology Europe*, involucra a la Universidad de Valladolid en esta tarea, abarcando la reconstrucción de trayectorias de aeronaves a partir de los datos de seguimiento obtenidos mediante la nueva tecnología ADS-B (Automatic Dependent Surveillance - Broadcast), que, a pesar de sus ventajas (menores costes, comunicación entre aeronaves, etc), presenta algunos problemas como la desincronización de las señales recibidas por los diferentes receptores terrestres.

El presente Trabajo Fin de Grado estudia esta problemática y propone una solución que transforma el supuesto en una variante del conocido *problema del viajante* (TSP) en la que el nodo inicial y final no coinciden. Se abordan los principales métodos de resolución del TSP profundizando sobre las *heurísticas de mejora local*. El modelo de solución es analizado y se concluye como satisfactorio al combinarlo con algoritmos como el *2-Opt* o el *Lin-Kernighan*, motivando la implementación escalable de uno de ellos. Se aborda también la posterior corrección de las marcas temporales de los datos de seguimiento.



**Palabras clave:** TSP, ADS-B, ATM, R, MapReduce.



# Abstract

International air traffic is more crowded each day. In Europe, Eurocontrol forecasts for 2023 a rise of 14 % in the number of flights moving throught the European air space compared to 2016, requiring to renovate the current management systems with new bigdata technologies. The AIRPORTS project (CIEN, 2015) leaded by *Boeing Research & Technology Europe* involves University of Valladolid in this task, including the construction of aircrafts trajectories using the *Automatic Dependent Surveillance—Broadcast* (ADS-B) technology, which, despite its advantages (cheaper, aircrafts inter-communications, etc), it has few time synchronizing problems involving signals received by different receivers.

This document aims to solve these problems, modeling the resorted trajectories as a variant of the famous Traveling Salesman Problem (TSP) in which the first and last node are not the same. The main TSP solving algorithms are studied, focusing on the *local search heuristics*. Since the solution model is proved to be satisfying combined with few algorithms such as *2-Opt* or *Lin-Kernighan*, a scalable implementation of one of them is motivated. Besides, a timestamps correction model for the surveillance data is approached.



**Keywords:** TSP, ADS-B, ATM, R, MapReduce.

# Índice general

**Lista de figuras** v

**Lista de tablas** ix

**1. Introducción** 1
- 1.1. Motivación . . . 3
- 1.2. Objetivos . . . 4
- 1.3. Organización de la memoria . . . 5

**2. Metodología de trabajo** 7
- 2.1. Entregables del proyecto . . . 8
- 2.2. Ciclo de vida del proyecto . . . 9
- 2.3. Herramientas y técnicas . . . 10

**3. Gestión del Tráfico Aéreo** 13
- 3.1. Introducción . . . 13
- 3.2. SESAR . . . 14
- 3.3. Los sistemas ADS . . . 15
- 3.4. El proyecto AIRPORTS . . . 17
- 3.5. El problema que motiva este trabajo . . . 20

**4. El Problema del Viajante** 25
- 4.1. Introducción al problema del viajante . . . 26
- 4.2. Historia del TSP . . . 30
- 4.3. Aplicaciones generales del TSP . . . 32
- 4.4. Algoritmos de resolución . . . 33
  - 4.4.1. Los algoritmos exactos . . . 33
  - 4.4.2. Los algoritmos de aproximación . . . 35
- 4.5. Heurísticas de mejora local . . . 37
  - 4.5.1. Los algoritmos de la familia *k-Opt* . . . 38
  - 4.5.2. El algoritmo de Lin-Kernighan . . . 41
  - 4.5.3. Algoritmo Or-Opt . . . 46

**5. Estudio preliminar de los algoritmos** **51**
5.1. Introducción a R . . . 52
5.2. Selección de librerías . . . 52
5.3. Análisis . . . 53
5.4. Desarrollo de la aplicación . . . 56
5.4.1. El modelo de reordenación de trayectorias: la técnica de las ventanas de ejecución . . . 60
5.5. Modelo de corrección de timestamps . . . 63

**6. Análisis de resultados** **71**
6.1. Estudio comparativo de los algoritmos de resolución del TSP aplicados a la reordenación de mensajes ADS-B . . . 71
6.2. Análisis del comportamiento del algoritmo 2-Opt a la reordenación de mensajes ADS-B en base a métricas de eficiencia . . . 75

**7. Implementación escalable** **79**
7.1. Implementación del *k-Opt* . . . 80
7.2. Desarrollo de una aplicación en Java que realice reordenaciones a través de la implementación de los métodos *k-Opt* . . . 81
7.3. Estudio de rendimiento de la nueva implementación para el *2-Opt* . . . 83
7.4. Adaptación a MapReduce . . . 85
7.4.1. Introducción a MapReduce . . . 86
7.4.2. Implementación escalable en MapReduce . . . 88

**8. Conclusiones** **95**
8.1. Consideraciones finales . . . 95
8.2. Aprendizaje personal . . . 96
8.3. Líneas de trabajo futuro . . . 97

**Bibliografia** **98**

**A. Contenido del CD y manuales de las aplicaciones** **101**
A.1. Contenido del CD . . . 101
A.2. Manuales . . . 103
A.2.1. Manuales de instalación . . . 104
A.2.2. Manuales de usuario . . . 106

# Índice de figuras


2.1. Gráfico ilustrativo de la metodología desarrollada para llevar a cabo el proyecto de investigación . . . . . 9
2.2. Herramientas de las que hace uso el proyecto . . . . . 11

3.1. Sistemas de geolocalización y seguimiento. A la izquierda, el tradicional método de obtención de la posición de la aeronave vía radar. A la derecha, el uso de tecnologías ADS-B. Fuente: `www.adsb.com` . . . . . 16
3.2. Segmento de dos trayectorias reconstruida con la librería `leaflet` de `R` a partir de las señales ADS-B recibidas . . . . . 19
3.3. Arquitectura de AIRPORTS DL. Fuente: Martínez Prieto et al. [2017] . . . 19
3.4. Alcance de las fuentes de datos más importantes que nutren a AIRPORTS DL. Fuente: Martínez Prieto et al. [2017] . . . . . 21
3.5. Solapamiento de los alcances de las fuentes *Frambuesa*, *OpenSky* y *ADSB-Hub*. Fuente: Martínez Prieto et al. [2017] . . . . . 22
3.6. Segmento de una trayectoria Madrid Barajas-Asturias sobre la que se ha aplicado una reordenación con el algoritmo *2-Opt*. Creado con `Leaflet` . . 23

4.1. Grafo $G$ . . . . . 28
4.2. El juego de inteligencia *The Icosian Game* reta al jugador a encontrar ciclos hamiltonianos sobre el grafo del dodecaedro regular bajo ciertas restricciones 30
4.3. Ilustración de un 2-change . . . . . 39
4.4. Ilustración de un 3-change . . . . . 40
4.5. Ilustración de un *2-change* durante la ejecución del *2-Opt* sobre una instancia con 30 ciudades. En negro y amarillo, aristas y nodos afectados respectivamente. Creado con `Leaflet` . . . . . 40
4.6. Ilustración de un *3-change* durante la ejecución del *3-Opt* sobre una instancia con 30 ciudades. En negro y amarillo, aristas y nodos afectados respectivamente. Creado con `Leaflet` . . . . . 41
4.7. Ejemplo de camino alterno . . . . . 42
4.8. Combinación de un camino alterno $P$ y una ruta $R$ sobre el grafo del octógono regular . . . . . 43

4.9. Ilustración de una mejora durante la ejecución del algoritmo *Lin-Kernighan* sobre una instancia con 30 ciudades. En naranja y negro, el camino alterno encontrado . . . . . 46
4.10. Ilustración de una mejora con longitud de cadena 3 durante la ejecución del algoritmo *Or-Opt* (ver Algoritmo 3) sobre una instancia con 30 ciudades. En negro y naranja la cadena de vértices, y en blanco la arista que se sustituye por la cadena . . . . . 48
4.11. Ruta de partida para la ejecución de los algoritmos de búsqueda local expuestos en esta sección, cuyos resultados finales se ilustran en la Figura 4.12 . . . . . 49
4.12. Rutas finales obtenidas de la ejecución de los cuatro algoritmos ilustrados sobre una misma instancia con 30 ciudades y con la misma ruta de partida, que muestra la Figura 4.11 . . . . . 49

5.1. Arbol de características de la aplicación en R . . . . . 55
5.2. Partes de la interfaz de la aplicación . . . . . 57
5.3. Widget mapas seleccionado . . . . . 58
5.4. Widget tabla seleccionado . . . . . 58
5.5. Widget pruebas seleccionado . . . . . 59
5.6. Ejemplo de reordenación de una trayectoria a través de su descomposición, con $\tau = 4$ y $\sigma = 2$ . . . . . 62
5.7. Representación de los mensajes ADS-B de cada trayectoria sobre un mapa creado con `Leaflet` . . . . . 63
5.8. Ejemplo de corrección de los timestamps tras haber ejecutado una reordenación sobre un vuelo, con los registros ordenados conforme al nuevo orden de los mensajes obtenido . . . . . 68
5.9. Ejemplo de corrección de los timestamps tras haber ejecutado una reordenación sobre un vuelo, vista de la gráfica *Distancia vs Tiempo* del *widget Mapas* . . . . . 68
5.10. Ampliación de la gráfica que muestra la Figura 5.9 . . . . . 69
5.11. Gráfica Distancia vs Tiempo al representar la altura de cada punto a través de un gradiente de colores . . . . . 69

6.1. Diagrama de barras: Longitud mejorada (en km) registrada con respecto a la ruta de partida inicial . . . . . 72
6.2. Diagrama de barras: Longitud mejorada (en km) registrada con respecto a la ruta de partida inicial obtenida por los métodos no constructivos . . . . 73
6.3. Diagrama de cajas: Efectividad (en %) de mejora en longitud registrada con respecto a la ruta de partida inicial comparando los resultados para cada modo de ejecución . . . . . 77

7.1. Esquema de funcionamiento de MapReduce . . . . . 87
7.2. Análisis de la implementación en *MapReduce*: diagrama de clases . . . . . 89

7.3. Procedimiento de procesado de los mensajes ADS-B por la implementación sobre *MapReduce* . . . . . 91
7.4. Salida de la aplicación en *MapReduce* . . . . . 92

A.1. Contenido del CD . . . . . 101
A.2. Contenido de la carpeta ProyectoR . . . . . 102
A.3. Contenido de la carpeta shiny . . . . . 102
A.4. Estructura de directorios de la implementación escalable . . . . . 103
A.5. Terminal desplegado tras ejecutar el fichero *run.bat* . . . . . 104
A.6. Pantalla de inicio de la aplicación . . . . . 105
A.7. Página principal del sitio web de Cloudera: `https://es.cloudera.com/` . 106
A.8. Página de inicio . . . . . 107
A.9. Controles del menú lateral . . . . . 108
A.10. Controles adicionales al seleccionar el *Simulated Annealing* como algoritmo en el *Selector de algoritmo* . . . . . 109
A.11. Controles adicionales al hacer click sobre el *checkbox Adaptar ventanas a aeropuertos* . . . . . 111
A.12. Controles en la parte central con *widget Mapas* seleccionado . . . . . 112
A.13. Gráficas mostradas al seleccionar cada una de las pestañas del *dashboard* . 113
A.14. Controles en la parte central con *widget Tabla* seleccionado . . . . . 114
A.15. Controles en la parte central con *widget Pruebas* seleccionado . . . . . 116
A.16. Controles adicionales al seleccionar la opción "Utilizar ventanas de un mismo tamaño", en la parte izquierda, o "Utilizar ventanas de distinto tamaño para aeropuertos", en la parte derecha, en el control *opciones de ejecución* 116
A.17. Secuencia de pasos a realizar para llevar a cabo la función *Ejecutar una reordenación* . . . . . 118
A.18. Secuencia de pasos a realizar para llevar a cabo la función *Ejecutar una reordenación* . . . . . 120

# Índice de tablas

5.1. Comparativa de la técnica de las ventanas de ejecución con respecto a su no utilización en longitud de la solución obtenida (Long) en kilómetros y tiempo de ejecución (Tmp) en segundos . . . . . 62

6.1. Comparativa de los algoritmos en longitud de la ruta encontrada (Long) en kilómetros y en tiempo de ejecución (Tmp) en segundos . . . . . 73
6.2. Resultados de la segunda parte del estudio preliminar . . . . . 76

7.1. Comparativa, resultados de la reordenación usando *2-Opt* con ventanas únicas de tamaño 100 y solapamiento 20 en las aplicaciones de `R` y *Java* . . 84
7.2. Comparativa, resultados de la reordenación usando *2-Opt* con ventanas de tamaño 100 y solapamiento 20 en vuelo y de tamaño 25 y solapamiento 5 en aeropuertos, distinguiendo por altura 6000, en las aplicaciones de `R` y *Java* 84

# Capítulo 1

# Introducción

Desde principios de siglo, el tráfico aéreo mundial viene siendo objeto de un crecimiento constante en su concurrencia [Grushka-Cockayne y De Reyck, 2009]. La *gestión del tráfico aéreo*, del inglés *Air Traffic Management* (ATM), pasa a cobrar cada vez más importancia en un entorno en el que el número de agentes es cada vez mayor con el paso del tiempo. A esta dinámica creciente hay que sumarle el aumento de la complejidad de los distintos tipos de actores que intervienen (aeropuertos, aerolíneas, aeronaves, etc) [Alonso Isla et al., 2018], resultando en una demanda en aumento de nuevas tecnologías que se adapten de forma adecuada a las cada vez mayores exigencias del sector.

El organismo encargado de la gestión del tráfico aéreo en Europa es Eurocontrol, creado en 1963 por la *International Civil Aviation Organization* (ICAO) con el fin de constituir una única corporación con responsabilidades sobre todo el espacio aéreo europeo [Grushka-Cockayne y De Reyck, 2009]. En sus inicios, los distintos gobiernos nacionales se negaron a renunciar parcialmente a su soberanía en favor de una gestión comunitaria del tráfico aéreo, derivando en un espacio estructurado en bloques o sectores nacionales que dificulta la gestión del tráfico aéreo.

La existencia de estas fronteras estatales, unida a la necesidad de modernizar los sistemas ATM continentales, derivaron en pérdidas estimadas entre 200 y 300 millones de euros anuales con respecto a otras estructuras homólogas, tales como el sistema norteamericano, que parte del proyecto *NextGen* [Alonso Isla et al., 2018]. Para acabar con esta situación, surge en marzo de 2004 la iniciativa *Single European Sky* (SES), de la mano de la Comisión Europea [Grushka-Cockayne y De Reyck, 2009], para abordar una reestructuración del espacio aéreo europeo que establezca una nueva división por bloques funcionales (acordes a los flujos del tráfico), dejando atrás el anterior sistema marcado por las fronteras nacionales, así como para modernizar los sistemas de control del tráfico aéreo o *Air Traffic Control* (ATC).

En este afán renovador, se incluye una de las tecnologías de seguimiento del tráfico aéreo más recientes: las *Automatic Dependent Surveillance - Broadcast* (ADS-B), que

suponen un importante paso hacia delante en esta materia. Mientras que su uso se hizo obligatorio en Europa para algunas aeronaves a partir de 2017, en otras regiones geográficas como Australia, la apuesta por estas tecnologías es aun más intensa, siendo en la actualidad de equipamiento obligatorio para todas las aeronaves que atraviesan su espacio aéreo [Álvarez Esteban et al., 2017].

La tecnología ADS-B se caracteriza por utilizar los sistemas de navegación vía satélite o *Global Navigation Satellite System* (GNSS) como método por el que la aeronave obtiene su posición, que envían a través del aire en forma de paquetes que reciben el nombre de *señales o mensajes ADS-B* [Álvarez Esteban et al., 2017]. Estas comunicaciones son recibidas por una red heterogénea de sensores distribuidos por todo el planeta y conectados con grandes repositorios de datos a los que posteriormente acceden los sistemas ATM.

Cuando una señal es recibida por un receptor, éste le asocia una marca temporal o *timestamp* a fin de poder establecer una ordenación temporal de los mensajes ADS-B asociados a un mismo vuelo o trayectoria. Sin embargo cada mensaje puede llegar a más de un receptor de tierra, cada uno de los cuales le asocia una marca temporal o *timestamp* de acuerdo con su reloj interno. No hay control de las estaciones que reciben cada mensaje ADS-B, lo que unido al hecho de que los mensajes viajan por el aire pudiendo ocasionarse retrasos, produce que un mismo mensaje recibido por varias estaciones receptoras tenga a su vez distintos valores de *timestamp* en lugar de tener el mismo.

Por tanto, y a pesar de las ventajas de ADS-B, entre las que se encuentran la veracidad de sus datos de geolocalización (al obtener la aeronave su posición a través de una red de satélites de comunicaciones), o su alta frecuencia de comunicaciones (de hasta 2 mensajes ADS-B por segundo), esta tecnología cuenta, por un lado, con la aparición de problemas de escalabilidad en los sistemas ATM, obligando a la consideración de tecnologías *bigdata* capaces de hacer frente a las altas cargas de procesamiento diarias producidas por la enorme cantidad de datos a tratar. Por otro lado, la tecnología ADS-B cuenta con el problema anterior de alineamiento temporal en los datos.

En esta línea, el proyecto AIRPORTS (CIEN, 2015), liderado por *Boeing Research & Technology Europe* trata las diferentes formas de fusionar los datos de diversas fuentes (receptores) de manera que la reconstrucción de las trayectorias se haga de la forma más fielmente posible a la realidad que representan.

Este Trabajo Fin de Grado propone un modelo que permita detectar y corregir estos desordenes producidos a causa de esta problemática a partir del conjunto de mensajes ADS-B procedentes de un vuelo. Esta solución pasa por considerar el problema de encontrar esta reordenación correcta de los mensajes ADS-B de un vuelo como una variante del conocido *problema del viajante*.

Este problema, cuyo nombre procede del inglés *Traveling Salesman Problem* (TSP) trata la búsqueda de caminos de mínima longitud que permitan visitar una serie de ubicaciones determinadas comenzando y terminando en una de ellas. Para reordenar los mensajes ADS-B asociados a una trayectoria es necesario introducir una pequeña variante

al problema consistente en buscar caminos con las propiedades anteriores no cerrados, esto es, que comiencen y terminen en diferentes ubicaciones. Como ubicaciones o nodos se consideran las coordenadas geográficas que acompañan a cada mensaje ADS-B.

Este modelado del problema requiere de un análisis detallado del problema del viajante a modo de estado del arte, que recoja sus características más importantes así como las diferentes técnicas y métodos que lo abordan. Dada la gran variedad de algoritmos existentes, este trabajo profundizará sobre el funcionamiento y características de un tipo muy conocido de estos métodos, las *heurísticas de mejora local*, los cuales, dada una primera solución, buscan mejorarla (obtener otra mejor) a través de modificaciones sobre ella que derivan en nuevas soluciones de mejor calidad.

La propuesta será analizada y posteriormente implementada de forma escalable, para poder formar parte de la lógica de un gran sistema de gestión del tráfico aéreo, que por tanto admita ejecuciones en paralelo sobre un sistema de altas prestaciones.

Además, este trabajo incorpora un modelo de corrección de las marcas temporales asociadas a los mensajes ADS-B de una trayectoria, que, tras haber sido reordenados, existirán inconsistencias entre sus marcas temporales (mensajes con *timestamp* posterior pueden quedar dispuestos antes que otros con *timestamp* anterior). En este sentido, se busca la implementación de este modelo para que actúe tras una reordenación reasignando los *timestamps* de los mensajes de manera que se asemejen lo más fielmente posible al instante temporal en que fueron emitidos.

## 1.1. Motivación

Los nuevos sistemas ATM en Europa han adoptado las nuevas tecnologías de seguimiento ADS-B, que, a pesar de requerir una infraestructura más barata, están desbancando a las tradicionales comunicaciones vía radares secundarios [Martínez Prieto et al., 2017].

Estos nuevos mecanismos de seguimiento se basan, como ya se ha mencionado, en la emisión periódica por parte de una aeronave de señales o mensajes ADS-B conteniendo una serie de datos que permiten el seguimiento del vuelo. Entre ellos se incluyen ubicación actual (en coordenadas geográficas: latitud y longitud), altura, velocidad y *callsign* (indicador de vuelo) [Álvarez Esteban et al., 2017].

Los mensajes ADS-B son recibidos por distintas redes de receptores, que, como hemos visto, asocian el *timestamp* conforme a su reloj interno. Posteriormente, los mensajes son almacenados en grandes bases de datos a las que acceden los sistemas ATM para operar con ellos. Entre otras aplicaciones, estos datos permiten la posterior reconstrucción de los trayectos realizados. Para ello, se requiere conocer el orden temporal de los mensajes ADS-B que se corresponden a un mismo vuelo, que a su vez se obtienen por lo general de

varias fuentes de datos no sincronizadas. Por tanto, se repercute en el problema temporal descrito anteriormente, generando errores en estas reconstrucciones.

Este trabajo tiene como objetivo resolver esta problemática, aumentando así el grado de exactitud de las posteriores reconstrucciones de las trayectorias con respecto a las realizadas realmente, obteniendo datos de longitud de la trayectoria (o distancia recorrida por la aeronave) más próximas a la realidad. Reduciendo las tasas de error en esta materia es posible un control más preciso del espacio aéreo, que entre otras consecuencias tiene el registro y estudio minucioso de los sucesos acontecidos en el espacio aéreo continental, derivando causas y responsabilidades de una forma más efectiva.

## 1.2. Objetivos

El presente Trabajo Fin de Grado estudia, como ya se ha mencionado, los problemas derivados del mal alineamiento temporal de los mensajes ADS-B con el fin de desarrollar una solución que contrarreste sus efectos sobre el procesamiento de los datos por parte de los sistemas de gestión del tráfico aéreo. En esta línea, se identifican los siguientes objetivos a cubrir:

OB-1 Desarrollo de una implementación escalable de un algoritmo que permita la reconstrucción de trayectorias de aeronaves a partir del conjunto de mensajes ADS-B asociados.

- OB-1.1 Formulación de un modelo de reordenación temporal de los mensajes ADS-B que aproxime lo máximo posible el verdadero orden en que son lanzados.
- OB-1.2 Implementación del modelo teórico que propone el subobjetivo OB-1.1 sobre alguna tecnología escalable que permita su integración a la lógica interna de un sistema de gestión del tráfico aéreo.

OB-2 Abordar la corrección de las marcas temporales o *timestamp* de los mensajes ADS-B asociados a una trayectoria tras haber sido reordenados

- OB-2.1 Creación de un modelo matemático de corrección de las marcas temporales, dada una reordenación de los mensajes.
- OB-2.2 Implementar la propuesta que plantea el subobjetivo [OB-2.1], a fin de que pueda ser puesta en práctica sobre trayectorias reales.

OB-3 Elaborar un compendio bibliográfico del problema del viajante, enfocando el estudio sobre las heurísticas de mejora local.

- OB-3.1 Identificar las heurísticas de mejora local más relevantes a fin de ser recogidas y estudiadas en esta memoria.
- OB-3.2 Analizar el funcionamiento de estos métodos y las técnicas en que se basan.

El proyecto a abordar por este trabajo, lleva así asociada una buena labor de investigación, con objeto de encontrar las mejores técnicas y estrategias que sustenten los modelos de reordenación y corrección a desarrollar. Para ello, se requiere analizar de qué manera los algoritmos que resuelven el problema del viajante son aplicables en este contexto en base a su comportamiento, coste computacional, procedimientos que utiliza, etc. Con este objetivo, a lo largo de la memoria se incluyen varias pruebas cuyos resultados serán de utilidad para determinar el nivel de adecuación de las distintas estrategias existentes en diferentes situaciones.

## 1.3. Organización de la memoria

El presente documento se divide en capítulos que tratan diferentes aspectos del proyecto. En primer lugar, se desglosa la metodología de trabajo en el Capítulo 2, donde se identifican los entregables del proyecto (ver Sección 2.1), se listan las fases que componen su ciclo de vida (ver Sección 2.2) y que marcan la estructura posterior de la memoria, y se listan las herramientas y técnicas identificadas (ver Sección 2.3).

El Capítulo 3 introduce el contexto que motiva el presente Trabajo Fin de Grado, que involucra la iniciativa de *Boeing Resarch & Technology Europe* y la colaboración de la Universidad de Valladolid en la construcción de una plataforma *bigdata* que habilite el estudio de métricas de eficiencia asociadas a la reconstrucción de trayectorias de aeronaves, en el marco del proyecto AIRPORTS, impulsando una renovación en los sistemas de gestión del tráfico aéreo europeo.

En el Capítulo 4 se analiza el estado del arte del proyecto. Este esfuerzo abarca los fundamentos teóricos del problema del viajante (TSP), historia, aplicaciones prácticas y algoritmos de resolución, de entre los cuales se analizan en profundidad un tipo especial de ellos, que reciben el nombre de *heurísticas de mejora local*. Estos algoritmos parten de un camino inicial dado e iteran sobre él realizando modificaciones sucesivas para obtener otros caminos mejores (más cortos). Para ello, cada heurística usa distintas técnicas sobre las que es necesario profundizar y analizar su comportamiento a la hora de ser adaptadas para poder aplicarse a la reordenación de los mensajes ADS-B de una trayectoria. Conviene destacar que, en el contexto de estas reordenaciones, se dispone de un camino inicial dado por el *timestamp* de los datos de seguimiento. Partiendo de esta ordenación, las heurísticas de mejora local pueden iterar para un nuevo orden que de como resultado una trayectoria reconstruida más corta.

A continuación, el Capítulo 5 expone los puntos más destacados del desarrollo de una aplicación destinada a albergar un estudio que permita establecer una primera comparación entre los algoritmos más destacados recogidos previamente en el análisis del estado del arte. No es objeto de esta aplicación implementar los algoritmos, sino el de reutilizar implementaciones libres de los mismos. Se opta por el lenguaje `R` para desarrollar esta aplicación, que cuenta con librerías como `TSP` y `stats` que abarcan buena parte de los

algoritmos más destacados para resolver el problema del viajante. En la Sección 5.5 se plantea el modelo de corrección de las marcas temporales de los mensajes ADS-B en base a diferentes parámetros incluidos en ellos como la velocidad, altura o geolocalización de la aeronave. Los resultados del estudio que pretende albergar esta aplicación se incluyen en el Capítulo 6.

El Capítulo 7 expone las fases posteriores del trabajo, incluyendo la implementación en *Java* de una de las heurísticas estudiadas en base a su adecuación a la reordenación de los mensajes, así como su integración a una nueva aplicación sobre *MapReduce* que permita la reordenación simultánea de varias trayectorias y la posterior corrección de las marcas temporales de sus mensajes, a fin de poder ser ejecutada sobre un clúster e integrado a un sistema de gestión del tráfico aéreo.

El Capítulo 8 incluye las conclusiones finales del proyecto, así como el trabajo futuro a realizar.

# Capítulo 2

# Metodología de trabajo

Una de las primeras tareas de todo proyecto es la elección y posterior configuración de las técnicas, procedimientos y estrategias aplicables a fin de llevar a cabo las diferentes actividades del proyecto. Se desarrolla, a fin de cuentas, una metodología de trabajo orientada a llevar a cabo las labores de investigación previas a las posteriores implementaciones que permitan cubrir los diferentes objetivos del proyecto, que han sido presentados en el Capítulo 1.

De esta forma, el presente capítulo expone la metodología planteada para abordar este trabajo, que requiere de una gran labor investigadora, comúnmente caracterizada por el desconocimiento de sus resultados finales, impidiendo así vislumbrar en los instantes iniciales del proyecto el punto final al que se llegará en el marco de este Trabajo Fin de Grado (pensemos que en el caso de resultar inadecuado el presente modelo de reordenación de los mensajes ADS-B en base al TSP, los trabajos posteriores para poner en práctica la propuesta no tendrán sentido). Esta incertidumbre dificulta el establecimiento de una planificación detallada de las actividades en los instantes iniciales o el desarrollo de una línea de base, así como una estimación del esfuerzo del proyecto; tareas imprescindibles en la gestión de cualquier proyecto de desarrollo software. Dados estos obstáculos iniciales, se opta por fijar una metodología en la que los objetivos de cada fase o etapa del proyecto se marquen en función de los resultados obtenidos en las anteriores.

A lo largo del proyecto, y especialmente al final de cada etapa, se han ido estableciendo sucesivas reuniones con los tutores que han tenido como objetivo poner en común los resultados obtenidos en la fase previa para, a partir de éstos, establecer los hitos posteriores así como los pasos necesarios para alcanzarlos.

## 2.1. Entregables del proyecto

En relación al conjunto de objetivos que se presentan en el Capítulo 1. Se identifican los siguientes entregables para el proyecto:

- Un **compendio bibliográfico del problema del viajante**, en línea con el objetivo [OB-3]. Su desarrollo es a su vez una tarea necesaria para poder abordar el objetivo [OB-1]. Se encuentra expuesto en el Capítulo 4 de la presente memoria.

- Una **aplicación que albergue un estudio comparativo de métodos**: a raíz del previo análisis del estado del arte. Se identifican una serie de métodos aplicables al problema del viajante, que análogamente, tienen aplicación en la reordenación de los datos de seguimiento de trayectorias de aeronaves. Con el fin de analizar la adecuación de cada uno de ellos a esta problemática, es oportuno llevar a cabo un estudio comparativo que permita encontrar la mejor de estas técnicas. Con este objetivo, se requiere el desarrollo de una aplicación que incluya la creación de un *dashboard* que ilustre los resultados obtenidos, así como de una lógica interna que soporte las ejecuciones del estudio. Esta memoria dedica el Capítulo 5 a esta aplicación.

- Un **informe de resultados** del análisis comparativo anterior que incluya los resultados de una extensa batería de pruebas expresados a través de una serie de métricas globales que permitan extraer de forma sencilla conclusiones acerca de la adecuación de cada algoritmo estudiado a la reordenación de los datos de una trayectoria.

- Una **implementación escalable que aplique el modelo de reordenación de trayectorias y corrección de marcas temporales propuesto**: del informe anterior se extraerá la eficacia de cada método. Para este entregable se escogerá el mejor de los algoritmos para ser implementado de manera que aborde la reordenación de trayectorias reales haciendo uso de un modelo de computación paralelizable. En esta línea, y conforme al [OB-1], se estudia el modelo de programación *MapReduce*, que a través de una serie de librerías en *Java*, brinda una serie de métodos ejecutables de forma paralela sobre un hardware adecuado como puede ser un cluster formando parte de un sistema gestor del tráfico aéreo.

- **Implementación de un modelo de corrección de *timestamps***: una vez queda probada la adecuación de la aplicación escalable, es posible abordar, en línea con el objetivo [OB-2], la creación y posterior implementación de un modelo que permita corregir las marcas temporales de los mensajes ADS-B asociados a un vuelo tras haberse ejecutado una reordenación. El objetivo será implementar este modelo en la aplicación escalable como segunda fase del procedimiento de reordenación. Por otro lado, este modelo también se implantará en la aplicación que albergue el estudio comparativo de métodos, pues gracias a sus componentes gráficos será posible analizar su eficacia así como visualizar los resultados de su ejecución.

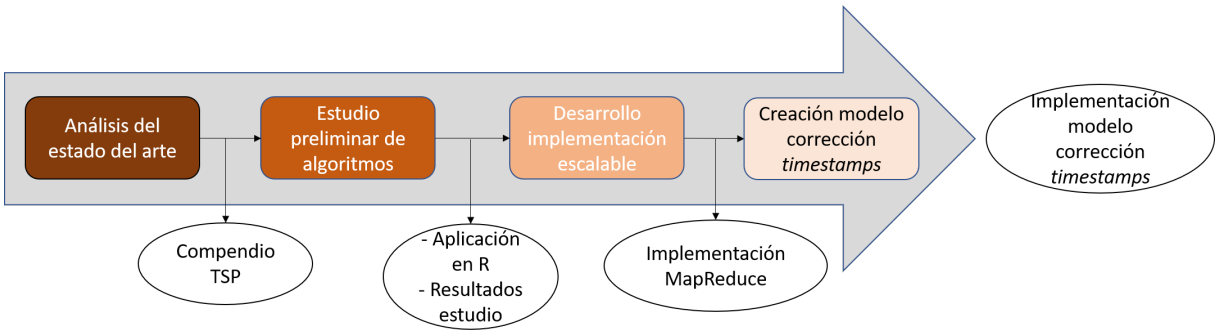



Figura 2.1: Gráfico ilustrativo de la metodología desarrollada para llevar a cabo el proyecto de investigación

## 2.2. Ciclo de vida del proyecto

Se puede establecer una clara división temporal del trabajo realizado en torno a las siguientes etapas:

1. **Estudio del estado del arte**: El proyecto inicialmente requiere la realización de una labor de documentación y estudio del problema del viajante o *Traveling Salesman Problem* (TSP). En el Capítulo 1, hablábamos de que el problema de la reordenación de los mensajes ADS-B de una trayectoria, objeto de estudio este TFG, puede ser modelado como una variante del problema del viajante en la que el punto inicial y el final de la ruta son distintos. Esto obliga a identificar y analizar los algoritmos más destacados que resuelven el problema del viajante. Son además objeto específico de investigación de este trabajo las *heurísticas de mejora local* (objetivo [OB-3]), entre las que encontramos los algoritmos *k-Opt* y *Lin-Kernighan*, que analizaremos en profundidad en la Sección 4.5. Todos los resultados del trabajo realizado en esta fase se hallan recogidos en el Capítulo 4. Este epígrafe conforma el único entregable de esta fase del proyecto.

2. **Estudio preliminar del comportamiento de los algoritmos**: tras una primera fase de documentación previa acerca del problema del viajante y algunos de los algoritmos más conocidos que lo resuelven, se establece una fase de estudio inicial que tiene como fin obtener unas primeras conclusiones acerca de la adecuación de los algoritmos a la reordenación de trayectorias. Para ello se hizo uso de las librerías `TSP` y `stats` de `R`, que incluyen gran parte de estos algoritmos. Este análisis abarca el estudio del tiempo de ejecución y de la longitud de la ruta encontrada en cada una de las ejecuciones realizadas, todas ellas en condiciones lo más semejantes posibles (mismo ordenador, misma carga del procesador, etc). El objetivo principal de esta etapa es la selección de los dos o tres algoritmos que mejores resultados hayan reportado según los criterios de estudio establecidos (tiempo de ejecución más bajo y longitud de la ruta encontrada más corta). El trabajo realizado a lo largo de esta fase se expone en el Capítulo 5, y los resultados y conclusiones obtenidas, fruto

del análisis citado en este párrafo, se incluyen en el Capítulo 6. Además de este entregable, se obtiene la aplicación que alberga el estudio, fruto de esta fase.

3. **Desarrollo de una implementación escalable del modelo de reordenación**: los resultados obtenidos del estudio de la fase previa permitirán determinar la bondad del comportamiento de cada uno de los algoritmos estudiados, así como su grado de adecuación o no al problema de reordenación de trayectorias. El siguiente paso consiste en el desarrollo de una implementación escalable de la mejor de las heurísticas de mejora local estudiadas en la fase anterior que sea aplicable como método de reordenación de trayectorias en un clúster *bigdata*. Para ello se utiliza el modelo de computación *MapReduce*, desarrollando la aplicación sobre una máquina Cloudera que simule el hardware de altas prestaciones requerido. De esta fase se obtiene como entregable esta implementación, en línea con el objetivo [OB-1].

4. **Implementación del modelo de corrección de *timestamps***: tras finalizar la implementación escalable del modelo de reordenación de los datos de seguimiento asociados a un vuelo, se añade una última característica que efectúe la posterior corrección de las marcas temporales de los mensajes ADS-B tras haber sido reordenados. Esta corrección es necesaria, pues al establecer un nuevo orden en los mensajes ADS-B, los *timestamps* de aquellos que han visto alterada su posición serán incoherentes con el resto de mensajes. Es por ello que se sustituye el valor de este campo para los mensajes en estas condiciones, de manera que el nuevo orden de los mensajes se corresponda con aquel que resultaría de ordenarlos ascendentemente por su marca temporal.

## 2.3. Herramientas y técnicas

A continuación se listan las herramientas identificadas para ser utilizadas a lo largo del proyecto (ver Figura 2.2):

- **Entorno RStudio**: a fin de desarrollar la aplicación en `R` que de cabida al estudio de la fase 2. *RStudio* presenta una gestión sencilla de las librerías de programación y *scripts*, así como una interfaz amigable para el programador que permite agilizar el trabajo a realizar.

- **Entorno NetBeans**: escogido para realizar la implementación en *Java*, debido a mi previa experiencia trabajando con este IDE. Permite la creación de proyectos *Java* de diversas condiciones, así como la gestión de las clases, librerías externas, integración con repositorios en la red como *GitHub*, y varios modos de compilación y *debugging*.

- **Cloudera**: el desarrollo de una implementación escalable requiere de software que soporte el elevado coste computacional de la computación paralela. Se trabajará

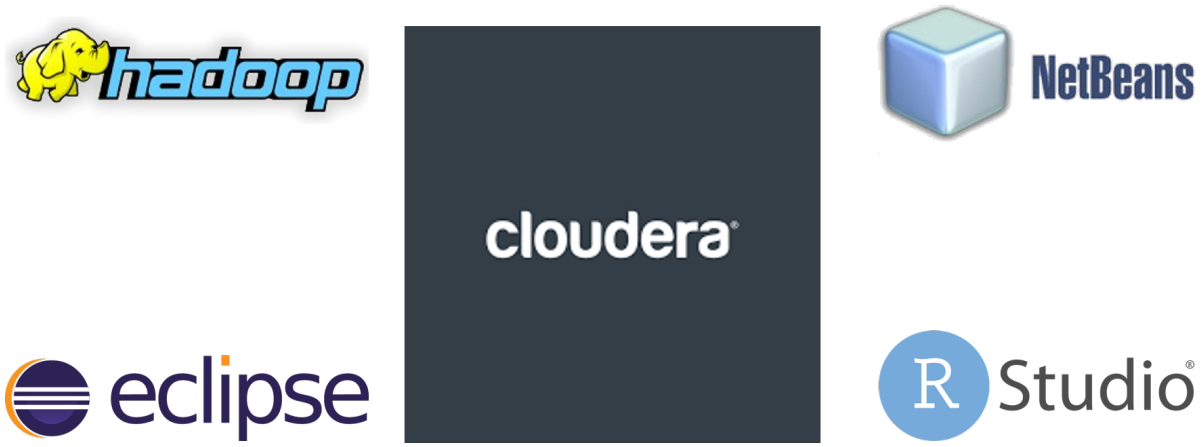



Figura 2.2: Herramientas de las que hace uso el proyecto

con una máquina virtual *Cloudera* que simule las condiciones de ejecución sobre un clúster.

- **MapReduce**: conjunto de librerías en *Java* dentro de Apache Hadoop que permiten el desarrollo de aplicaciones de computación paralela aptas para sistemas distribuidos. El objetivo es utilizar esta librería para desarrollar la aplicación sobre la máquina virtual *Cloudera*, que simule el hardware requerido para soportar el alto coste computacional que implica su ejecución.
- **Entorno Eclipse Luna**: para ejecutar las librerías de Hadoop el entorno de ejecución más adecuado es *Eclipse* teniendo en cuenta la buena gestión de librerías externas que permite.

# Capítulo 3

# Gestión del Tráfico Aéreo

El presente capítulo tiene como finalidad establecer el contexto o marco de este Trabajo Fin de Grado. En línea con lo establecido en el Capítulo 1, se plantean algunos de los elementos más importantes que dan sentido a la elaboración de este proyecto. Para comenzar se expone la problemática de la gestión del tráfico aéreo en Europa en la Sección 3.1. A continuación, se trata la iniciativa *SES* o *Single European Sky*, que como se ha venido comentando, persigue reestructurar el espacio aéreo europeo; así como el proyecto SESAR que pretende llevar *SES* a cabo (ver Sección 3.2). Por otro lado encontramos las tecnologías de seguimiento ADS-B, que como ya se ha anunciado en el capítulo anterior, son de reciente incorporación en Europa y cuentan con una gran ventaja competitiva frente a otras tecnologías existentes (ver Sección 3.3). A continuación, se introduce el proyecto AIRPORTS, en el que se enmarca el presente trabajo (ver Sección 3.4). El capítulo finaliza con la exposición formal del problema que se genera en la reconstrucción de trayectorias por la mala ordenación temporal de las señales ADS-B ya anunciada en el capítulo previo (ver Sección 3.5).

## 3.1. Introducción

Como se ha mencionado en el Capítulo 1, el tráfico aéreo internacional ha sufrido un gran aumento en el número de aeronaves en circulación. Este hecho es debido fundamentalmente al avance de las tecnologías aeronáuticas y a los mayores niveles de seguridad que es posible garantizar en el tráfico de pasajeros y mercancías. Esta circunstancia explica que haya cobrado especial relevancia lo que hoy se denomina como **gestión del tráfico aéreo** o *Air Traffic Management* (ATM), que consiste en la necesidad de regular el tráfico aéreo existente a través de una serie de sistemas de asistencia y seguimiento a las aeronaves desde su despegue en un determinado aeródromo hasta su aterrizaje en otro.

Este sistema de ATM se desarrolla en un entorno cada vez más complejo, compuesto de distintos elementos tales como aerolíneas, aeronaves o aeropuertos que cohabitan entre

sí [Alonso Isla et al., 2018]. A su vez, este medio se ve afectado por algunos factores como la meteorología, la estructura de los espacios aéreos alrededor de los aeropuertos o la diversidad de procedimientos de control que se aplican.

En el contexto europeo se reproduce esta misma tendencia. Eurocontrol prevé subidas anuales en el tráfico aéreo de $1,7\,\%$ hasta 2023, alcanzando ese mismo año la cifra de $11,6\,(\pm 1,2)$ millones de desplazamientos aéreos en ese año, lo que significa, un $14\,\%$ más que en 2016 [Eurocontrol, 2017]. Como ya hemos mencionado en capítulos anteriores, es necesaria una renovación en los sistemas ATM europeos, así como una reestructuración del espacio aéreo continental, hasta ahora dividido en bloques nacionales debido a la negativa de los distintos estados comunitarios a ceder sus competencias en esta materia en favor de Eurocontrol, la institución europea creada para llevar a cabo la gestión del tráfico aéreo continental.

A fin de deshacer esta tradicional división que venía caracterizando el espacio aéreo europeo, se plantea una nueva organización del mismo basada en bloques funcionales (en relación a los flujos de tráfico). Para materializar esta propuesta, nace la iniciativa *Single European Sky (SES)*, que propone coordinar el diseño y la regulación del *espacio aéreo común europeo* o *European Common Aviation Area* (ECAA).

## 3.2. SESAR

Con el objetivo de crear el *Single European Sky*, nace el *Single European Sky ATM Research (SESAR)* [Grushka-Cockayne y De Reyck, 2009] a modo de proyecto colaborativo que persigue unificar la gestión de todo el tráfico aéreo europeo. Nace en el año 2004 como continuación de un esfuerzo menor llevado a cabo por Eurocontrol denominado SESAME, del cual parte la iniciativa SES y que se desarrolla, a su vez, en paralelo con el actual modelo de unificación del espacio aéreo norteamericano.

El proyecto SESAR está concebido en tres fases:

- **Definición (2004-2008)**: que consistió en la entrega de un plan maestro para la ATM en el que se defina el contenido, desarrollo y creación de planes de producción para la próxima generación de sistemas ATM.
- **Desarrollo (2008-2013)**: en la cual se estableció la producción de la nueva generación de sistemas y componentes tecnológicos definidos en la fase de definición. Esta etapa contó con un presupuesto de 2.1 billones de euros y fue gestionada por la asociación *SESAR Joint Undertaking*, que engloba compañías tanto públicas como privadas.
- **Producción (2014-2020)**: con el fin de construir las nuevas infraestructuras que formarán el ATM, con la interoperabilidad necesaria entre sus componentes que

garantice un alto rendimiento asociado a las actividades de transporte aéreo en Europa.

El objetivo perseguido a través de SESAR se basa en una serie de nuevas funcionalidades clave tales como un plan de operaciones de la red que asegure una visión común de la situación actual, una integración completa de las operaciones aeroportuarias como parte de la gestión del tráfico aéreo, una gestión de las trayectorias que reduzca al mínimo las restricciones sobre el espacio aéreo y el desarrollo de nuevos modos de separación entre aeronaves en vuelo incrementando la seguridad y gestión de la información en todo el sistema o *System-Wide Information Management (SWIM)* y permitiendo la comunicación y la compartición de datos entre todos los stakeholders. Se considera al ser humano como centro de la toma de decisiones siendo asistido por nuevas funciones automáticas que faciliten su carga de trabajo a lo largo del proceso.

## 3.3. Los sistemas ADS

Las tecnologías *Automatic Dependent Surveillance* consisten en sistemas de seguimiento cuya función principal es habilitar el intercambio automático de mensajes entre varios sistemas de control del tráfico aéreo o *Air Traffic Control* (ATC). Estos mensajes incluyen datos tales como el *callsign* o identificador de vuelo, coordenadas de la posición de la aeronave, altura o velocidad, los cuales pueden ser mandados de forma periódica o bajo petición [Juricic et al., 2002].

Existen dos variantes de la misma: la tecnología ADS-C (*Automatic Dependent Surveillance-Contract*) y ADS-B (*Automatic Dependent Surveillance-Broadcast*), la última de las cuales se considera una versión mejorada de ADS, en tanto que permite la emisión omnidireccional de mensajes.

En el contexto de evolución de los sistemas de gestión del tráfico aéreo (ATM), ADS-B es un pilar fundamental de los nuevos sistemas ATC, al permitir el seguimiento de las trayectorias de vuelo que, como ya se ha mencionado en el capítulo introductorio, hace uso del sistema de navegación global vía satélite o *Global Navigation Satellite System* (GNSS), en detrimento de las tradicionales comunicaciones vía radar. La tecnología ADS-B proporciona una mayor seguridad, capacidad y eficiencia sobre los espacios aéreos que la utilizan.

ADS-B, al igual que la tecnología *contract* (ADS-C), está destinada a proveer, de forma automática, los datos referidos a la posición actual de las aeronaves sobre el espacio aéreo. Se caracteriza por

i) ser *automática*, al no requerir la intervención humana,

ii) ser *dependiente*, debido a que la generación de datos está sujeta al sistema de a bordo de la aeronave,

iii) proveer *datos de seguimiento*, similares a los enviados anteriormente vía radar, tanto a controladores en tierra como a otras aeronaves.

Aunque los sistemas se encuentran actualmente en un periodo de transición para incorporar la tecnología ADS-B, los equipamientos asociados a la misma son ya obligatorios para algunas aeronaves en Europa desde finales de 2017. Estos vehículos han sido equipados con transmisores de mensajes ADS-B que contienen información acerca de la trayectoria seguida, y que mandan mensajes que comprenden una serie de datos como el código hexadecimal de 24 bits asociado a la aeronave, el código identificativo del vuelo o *callsign*, datos de geolocalización, altitud o velocidad.

La principal ventaja de la tecnología ADS-B es que provee *control del tráfico aéreo* (ATC) con la posición en tiempo real obtenida mediante el sistema de navegación que, por lo general, es más preciso que el sistema basado en radares. Además, funciona a bajas altitudes y en tierra, por lo que es posible hacer un seguimiento de las maniobras realizadas por el piloto durante el aterrizaje.

La infraestructura necesaria para el despliegue de ADS-B queda conformada por dos tipos de componentes:

i) ADS-B Out, que emite los datos de la aeronave (e.g. identificación, posición, altitud, velocidad).

ii) ADS-B In, que recibe información de utilidad para el piloto, como datos del tráfico, u otra información transmitida mediante el servicio de información sobre previsiones de vuelo o *Flight Information Service-Broadcast* (FIS-B), como previsiones meteorológicas, restricciones temporales, etc.

El despliegue de la tecnología ADS-B requiere a su vez de dos componentes aeronáuticos: un sistema de navegación junto a un enlace de datos con la aeronave, y una estación receptora que obtiene las señales ADS-B.

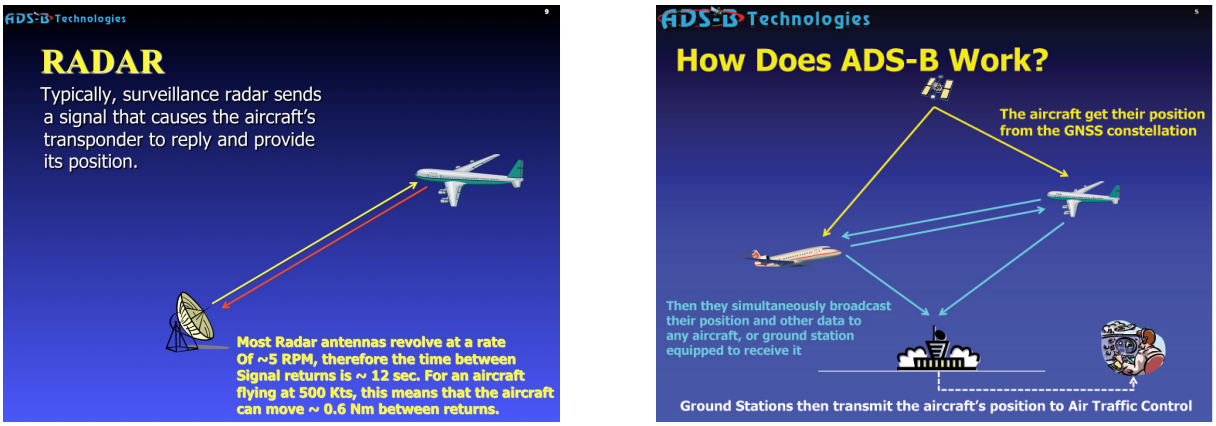



Figura 3.1: Sistemas de geolocalización y seguimiento. A la izquierda, el tradicional método de obtención de la posición de la aeronave vía radar. A la derecha, el uso de tecnologías ADS-B. Fuente: `www.adsb.com`

En la Figura 3.1 se ilustra el mecanismo de seguimiento que sigue la tecnología ADS-B frente al efectuado por sistema de comunicación de la posición vía radar. Mediante el sistema ADS-B, cada aeronave obtiene su geolocalización vía GNSS (*Global Navigation Satellite System*) a través de una red de satélites de comunicaciones situados en la órbita terrestre. A continuación, la aeronave, una vez ha determinado su posición, la emite en forma de mensaje ADS-B que será recibido por cualquier estación receptora terrestre cercana, que la almacena y procesa. A su vez, otras aeronaves pueden recibir e interpretar los mensajes ADS-B emitidos por otras permitiendo así determinar la distancia a la que se encuentran para, en caso de aproximarse en exceso, desviar su trayectoria evitando posibles colisiones o interferencias.

La frecuencia de transmisión de mensajes ADS-B por cada aeronave es de 2 por segundo, lo que resultan en una enorme cantidad de datos asociados a un mismo vuelo que se reciben, almacenan y procesan por los sistemas ATM. Esto hace que la escalabilidad sea un requisito indispensable de estos sistemas, demandando así el uso de tecnologías *bigdata* que puedan llevar a cabo las operaciones necesarias para procesar tal cantidad de datos.

Este hecho es lo que motiva la actual colaboración de la Universidad de Valladolid con *Boeing Research and Technology Europe*, en la que se establecen varias líneas de investigación que abarcan la construcción de una plataforma *bigdata* para ATM, que actúe como centro de datos y que englobe los procesos desde la ingesta de datos "en crudo" emitidos por cada vuelo hasta el refinamiento de los mismos que de lugar a información con valor para el usuario final. La descripción de la arquitectura de esta plataforma puede encontrarse en Álvarez Esteban et al. [2017], y a continuación se describe en líneas generales.

## 3.4. El proyecto AIRPORTS

En los últimos años, la optimización aplicada al campo de la gestión del espacio aéreo se ha convertido en un tema de interés especialmente motivado por el aumento continuo del número de vuelos. Esta optimización recae en el análisis de grandes cantidades de datos procedentes de diferentes actores del sistema ATM, lo que convierte este procesamiento en un problema de *bigdata* en términos de *volumen*, *velocidad* y *variedad* de los datos. Sin embargo, la industria aeronáutica no dispone de gran influencia para conseguir integrar este tipo de tecnologías como sí que disponen otras como la sanitaria o la financiera [Martínez Prieto et al., 2017].

En este contexto, nace el proyecto AIRPORTS como un consorcio formado por un conjunto de empresas y *Organismos Públicos de Investigación* (OPI) liderados por *Boeing Research and Technology Europe*, con el propósito de desarrollar nuevas tecnologías aplicables a la aeronáutica que mejoren la eficiencia en las operaciones de ATM. De esta forma,

una de las contribuciones más importantes del proyecto es la mejora de estas operaciones aplicando las tecnologías *bigdata* más recientes.

El objetivo principal de AIRPORTS es estudiar cómo los datos de seguimiento procedentes de diferentes proveedores pueden ser combinados con otros datos similares mejorando la calidad (veracidad) y siendo más relevantes como entrada de las herramientas de soporte a la toma de decisiones empleados en los sistemas ATM.

Los datos de seguimiento permiten la reconstrucción de las trayectorias de los vuelos desde que la aeronave se dirige a la pista de despegue hasta que se apagan los motores tras haber aterrizado. La continua actualización de la posición de cada aeronave en vuelo supone una fuente de datos clave para este propósito, dotando al sistema de información sobre su posición, altitud o velocidad, que es recogida durante el vuelo. Sin embargo, la gestión correcta de estos datos solo es posible mediante tecnología *bigdata*, y de hecho, la cantidad de estos datos es mucho mayor cuando se hace uso de tecnologías ADS-B, pues las señales emitidas por cada aeronave son mucho más frecuentes. Más aún, se está acrecentando la extensión del área terrestre sobre la cual son aplicables las tecnologías ADS-B, al estar siendo cubierta por una infraestructura de satélites de comunicaciones.

Sin embargo, tal y como hemos comentado en el capítulo introductorio, a día de hoy existe un problema de veracidad en los datos de seguimiento procedentes de ADS-B debido a que el *timestamp* (marca temporal) del mensaje, es asignado por la estación de destino en el momento en que lo recibe, y no por el sistema emisor cuando lo envía. Esto deriva en errores en la reconstrucción de las trayectorias al poder producirse desórdenes puntuales en los datos de seguimiento (dos mensajes enviados de forma consecutiva pueden llegar a estaciones receptoras en orden inverso al que fueron enviados), que se propagará al procesamiento y análisis de la información del vuelo, incluyendo la posterior reconstrucción de la trayectoria efectuada. En la Figura 3.3 se ilustran dos ejemplos de segmentos de la reconstrucción de dos trayectorias en donde se observan los efectos de esta problemática. La línea roja une los distintos datos de seguimiento por el orden que marca su *timestamp*.

Finalmente, cabe destacar como una de las contribuciones más importantes del proyecto AIRPORTS la creación de un *data lake* que recibe el nombre de AIRPORTS DL [Martínez Prieto et al., 2017]. Siendo uno de los conceptos más novedosos asociado al *bigdata*, un *data lake* permite la ingesta y almacenamiento de datos *en crudo* (aquellos que aun no han sido procesados por haberse obtenido recientemente) para ser tratados posteriormente. AIRPORTS DL está basado en sistemas de ficheros HDFS (*Hadoop Distributed File System*) y el modelo de computación *MapReduce* para trabajar con grandes volúmenes de datos de seguimiento ADS-B procedentes de una gran variedad heterogénea de fuentes de datos aéreos.

AIRPORTS DL se compone de los siguientes elementos [Martínez Prieto et al., 2017]:

- *Fuentes de datos*: abarca un amplio conjunto de servicios de tiempo real y bases de datos de las cuales AIRPORTS DL obtiene los datos en crudo.


(a) Vuelo Madrid Barajas-Palma de Mayorca

(b) Vuelo Madrid Barajas-Asturias

Figura 3.2: Segmento de dos trayectorias reconstruida con la librería `leaflet` de `R` a partir de las señales ADS-B recibidas

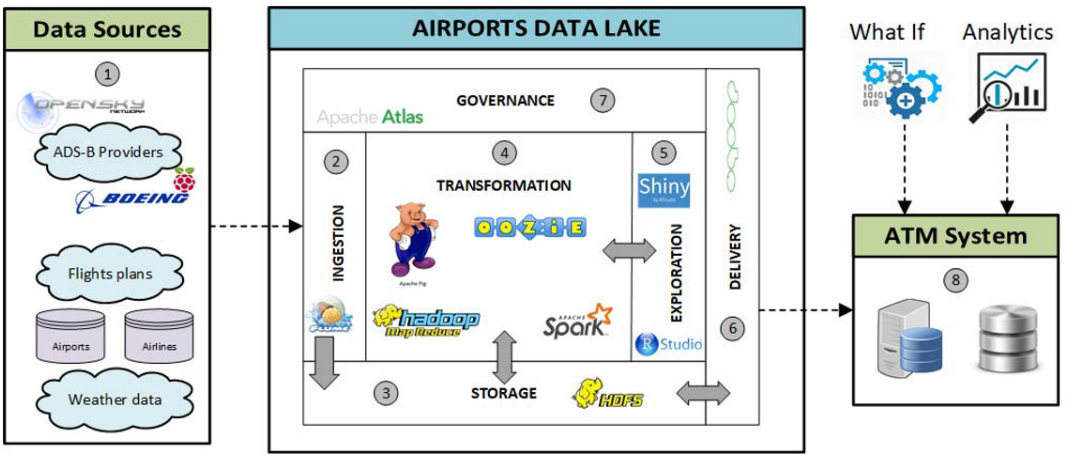



Figura 3.3: Arquitectura de AIRPORTS DL. Fuente: Martínez Prieto et al. [2017]

- *Ingestión*: componente encargado de la extracción de los datos procedentes de cada uno de los proveedores anteriores.
- *Almacenamiento*: diseñado de forma escalable y para garantizar la persistencia de grandes ficheros de datos a un bajo coste.
- *Transformación*: responsable de la manipulación y transformación de los datos. En estos procesos entra en juego el modelo de computación de *MapReduce* para llevar a cabo el procesamiento simultáneo de varios conjuntos de datos.
- *Exploración*: sirve a modo de interfaz de usuario. Permite analizar, así como visualizar los datos contenidos en el sistema.
- *Producción*: lleva a cabo la carga de datos previamente tratados a modo de salida para otros componentes o sistemas fuera de AIRPORTS DL.
- *Governanza*: permite la monitorización de los procesos de manipulación de datos, preservando suficiente información para poder trazarlos.
- *Sistema ATM*: es externo al *data lake* pero comunica con él. El *data lake* procesa los datos que necesita el sistema ATM para llevar a cabo las diferentes operaciones ATM que se le requieren.

## 3.5. El problema que motiva este trabajo

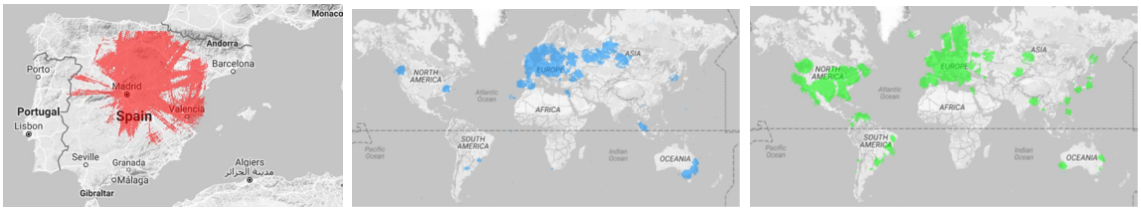
Las tecnologías ADS-B, de reciente aparición, van cobrando cada vez más importancia en el entorno de la gestión del tráfico aéreo a nivel mundial gracias a las mejoras que incorpora en comparación a los sistemas de seguimiento existentes previamente. Entre sus ventajas ya mencionadas está la mayor cantidad de información que es posible enviar por parte de una aeronave durante el vuelo mediante paquetes de información que en este trabajo se denotan por *señales o mensajes ADS-B*, con una frecuencia de hasta 2 por segundo. Además de aumentar la frecuencia de las comunicaciones, la determinación de la posición vía satélite aumenta la precisión con la que la ubicación es determinada.

Sin embargo, y tal y como se menciona en la sección anterior, existe un problema de veracidad en los datos de seguimiento ADS-B derivados de su proceso de comunicación. Esta problemática consiste en una ordenación temporal incorrecta de las señales ADS-B recibidas para ser posteriormente procesadas con respecto al orden en que fueron emitidas. Esto se debe a la producción de alteraciones que afectan al orden de llegada de los mensajes a una determinada fuente de datos, que es donde se asigna el *timestamp*, o instante temporal a los mensajes.

La mayor parte de estos problemas se deben a la falta de sincronización entre redes de sensores distribuidos por todo el planeta. No todos los mensajes correspondientes a un mismo vuelo son recibidos por las mismas estaciones receptoras, y así, dos mensajes

emitidos consecutivamente que son captados por diferentes sensores pueden ver intercambiado su orden de recepción, bien debido a la mayor proximidad del avión al segundo receptor en comparación con el primero, bien porque, aun habiéndose recibido antes el primer mensaje, el *timestamp* que se asigna al primero es posterior debido a que el reloj interno del sensor que lo captó está más atrasado que el del sensor que captó el segundo mensaje, etc. Estas situaciones se dan más acusadamente en el caso de vuelos de larga distancia o los que atraviesan océanos, cuyas señales son captadas por muchos receptores de múltiples organizaciones a lo largo del vuelo, situados en husos horarios diferentes.

De la fusión (*merge*) de los datos de distintas fuentes pueden producirse a su vez descuadres en el procesamiento. En el caso del *data lake* AIRPORTS DL, que se describe al final de la sección anterior, son varios los *datasets* que nutren de *datos en crudo* al sistema. En la Figura 3.4 se ilustra el alcance que reportan las fuentes de datos de seguimiento más importantes que aportan datos al *data lake*.




(a) Frambuesa (b) ADSBHub (c) OpenSky

Figura 3.4: Alcance de las fuentes de datos más importantes que nutren a AIRPORTS DL. Fuente: Martínez Prieto et al. [2017]

Nótese que muchos mensajes ADS-B son recibidos por sensores de varias de estas fuentes de datos (como se observa en la Figura 3.5), lo que obliga a llevar a cabo ese proceso de fusión de datos tras ser recibidos por el sistema, con el fin de eliminar duplicados y corregir los datos de *timestamp*, que presumiblemente serán diferentes en cada mensaje duplicado debido a la no sincronización que se indicó previamente.

A partir de esta problemática expuesta, este trabajo plantea una definición formal para el problema de reordenar los mensajes ADS-B asociados a una misma trayectoria, en el Problema 1. Su enunciado, requiere la especificación del concepto de *permutación*, presente en la Definición 3.1.

**Definición 3.1** *Una* ***permutación*** $\pi$ *sobre un conjunto* $N = \{1, 2, ..., n\}$ *con* $n \in \mathbb{N}$ *es una función biyectiva que va de* $N$ *en si mismo. Se dice además que* $\pi$ *es* ***circular*** *si para todo* $E \subsetneq N$ *no vacío, existe* $e \in E$ *de manera que* $\pi(e) \notin E$.

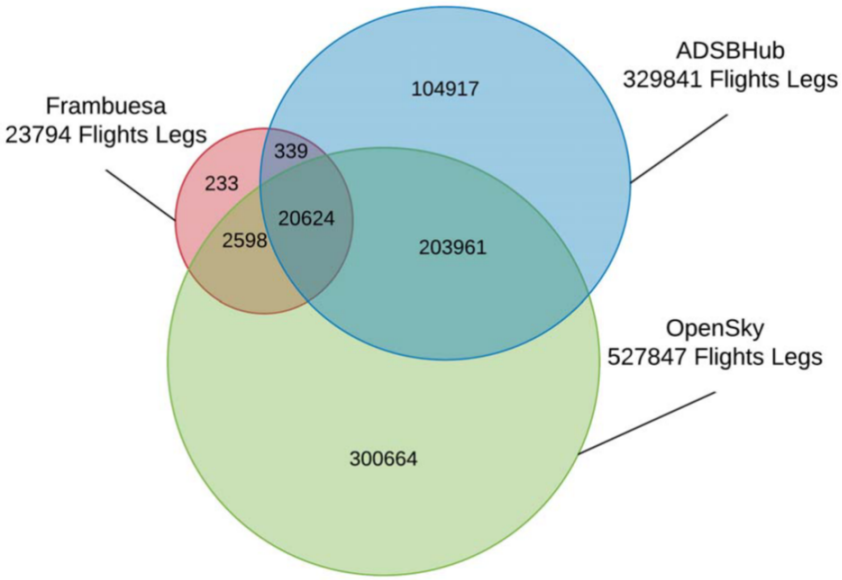


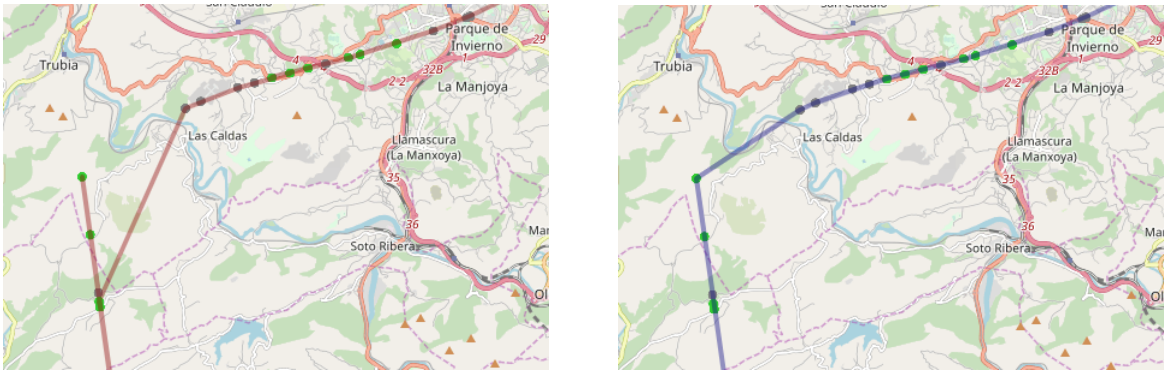

Figura 3.5: Solapamiento de los alcances de las fuentes *Frambuesa*, *OpenSky* y *ADSBHub*. Fuente: Martínez Prieto et al. [2017]

**Problema 1: Problema de reordenación de las trayectorias**

Se supone un avión que efectúa una trayectoria desde un aeródromo $A$ hasta otro aeródromo $B$, enviando $n$ mensajes ADS-B en el transcurso de su realización que son recibidos por diversos receptores y etiquetados con *timestamps* $t_1 < t_2 < ... < t_n$, siendo $t_i$ el *timestamp* que se asocia al mensaje $i$ para todo $1 \leq i \leq n$.

**Problema**: encontrar una permutación circular $\pi$ sobre $\{1, ..., n\}$ tal que para cada par de mensajes $1 \leq i < j \leq n$ se tiene que el mensaje $\pi(i)$ fue enviado antes que el mensaje $\pi(j)$.

Nótese que, por lo general, cualquier desorden generado (i.e. dos mensajes $i, j$ tales que aunque $i$ se envió antes que $j$ sus respectivos *timestamp* al recibirse verifican que $t_j < t_i$) produce una trayectoria más larga que la realmente recorrida por la aeronave. Es por ello que en la mayoría de los casos (salvo excepciones relativas a maniobras de los pilotos en los aeropuertos esperando permisos desde el control), la solución a este problema coincide con la reordenación de los mensajes que se corresponda con la trayectoria más corta, por lo que parece razonable considerar esta suposición, que nos brinda además la posibilidad de modelar el problema a través del renombrado *problema del viajante*, sobre el que profundizaremos en el Capítulo 4.




(a) Ruta sin ordenar: 511,99 km

(b) Ruta ordenada: 452,95 km

Figura 3.6: Segmento de una trayectoria Madrid Barajas-Asturias sobre la que se ha aplicado una reordenación con el algoritmo *2-Opt*. Creado con `Leaflet`

La Figura 3.6 (a) ilustra la reconstrucción de un segmento de trayectoria formado a partir de mensajes ADS-B mal ordenados temporalmente. Por su parte, en la Figura 3.6 (b) se observa el mismo segmento tras haberse aplicado una reordenación sobre el conjunto de mensajes ADS-B asociado al vuelo basada en este modelo teórico. Nótese además la enorme reducción que se obtiene en la longitud de la trayectoria realizada.

# Capítulo 4

# El Problema del Viajante

En el presente capítulo se recogen los fundamentos del ya mencionado *problema del viajante*. Este problema de optimización combinatoria pertenece a la clase de los problemas $\mathcal{NP}$-*difíciles* (o $\mathcal{NP}$-*hard*), que significa que no existe ningún algoritmo en la actualidad que garantice encontrar la solución óptima al mismo de forma computacionalmente razonable (en concreto, con un coste computacional de orden polinomial). Por tanto, será necesaria una máquina con gran capacidad de cómputo para resolver supuestos del problema de tamaño considerable.

En la actualidad, son varias las líneas de investigación establecidas a fin de desarrollar algoritmos con las propiedades antes mencionadas, o al menos, mejorar las técnicas ya existentes, algunas de las cuales se vienen utilizando desde hace ya varias décadas. Este capítulo, así como el propio trabajo, se centra, con motivo del Objetivo [OB-3], en una de las corrientes más destacadas en términos de efectividad de los métodos que abarca a la hora de abordar el problema del viajante: las *heurísticas de mejora local*. Estos algoritmos parten de una ruta que, como veremos a continuación, no es más que un camino que atraviesa una, y sólo una vez, cada uno de los nodos (también llamados *ciudades*) del grafo de la instancia partiendo y terminando sobre uno de ellos. Esta ruta inicial trata de mejorarse cada iteración realizando sobre ella continuas modificaciones basadas en una serie de técnicas específicas que permiten la obtención de nuevas rutas más cortas. Cada algoritmo implementa sus propias técnicas que lo diferencian del resto.

La estructura del capítulo es como sigue. En primer lugar, se define, en la Sección 4.1, el problema del viajante en los términos en que será tratado a lo largo del trabajo. A continuación, en la Sección 4.2, se presentan los detalles históricos más relevantes asociados al TSP. El problema del viajante presenta en la actualidad una gran cantidad de aplicaciones prácticas, algunas de las cuales se describen en la Sección 4.3. El capítulo cierra presentando los algoritmos de resolución del problema más destacados (ver Sección 4.4), centrando el foco sobre las heurísticas de mejora local, cuyo funcionamiento se describe en la Sección 4.5.

## 4.1. Introducción al problema del viajante

El problema del viajante, del inglés *Traveling Salesman Problem* (TSP), es uno de los problemas más estudiados en las últimas décadas. Presenta una gran cantidad de aplicaciones prácticas y multitud de algoritmos que lo resuelven a través de un amplio conjunto de estrategias. Las numerosas investigaciones del problema a lo largo del tiempo han venido motivadas por la notable dificultad que este problema presenta para obtener la solución óptima. Esta es, aquella que se refiere al camino de mínima longitud que recorra un conjunto de puntos o ciudades comenzando y terminando en uno mismo de ellos. Hasta hoy, se han ido desarrollando una gran variedad de algoritmos cada vez más eficientes aplicables al TSP. Este hecho, unido a la mejora constante de los sistemas informáticos, con cada vez mayor capacidad de cálculo, permite la resolución de instancias cada vez más grandes del TSP.

En esta sección, se define el problema del viajante en base a una terminología que se apoya sobre algunos de los conceptos más básicos de la teoría de grafos, los cuales se presentan a continuación.

Para comenzar, la Definición 4.1 comprende el concepto de *grafo*.

**Definición 4.1** *Se define un* ***grafo*** $G = (V, E)$ *como un conjunto* $V$ *de puntos* $v \in V$ *denominados* ***vértices*** *o también* ***nodos*** *y un conjunto* $E \subseteq V \times V$ *de conexiones* $e = (v_i, v_j) \in E$ *que unen pares de vértices* $v_i, v_j \in V$*, denominadas* ***aristas****. Se dice que el grafo es* ***no dirigido*** *si para cada* $(v_i, v_j) \in E$ *se tiene que* $(v_j, v_i) \in E$*. En caso contrario, se dice que el grafo es* ***dirigido****. Un grafo se denomina* ***completo*** *si* $\{(v_i, v_j) \in V \times V \,|\, i \neq j\} \subseteq E$*.*

En el contexto del problema del viajante, los grafos que vamos a considerar van a ser completos (y por tanto no dirigidos). El hecho de la no completitud del grafo supondría que la ruta encontrada no pudiese atravesar determinados pares de ciudades de forma consecutiva, al no existir la arista que las une. Esta restricción no resulta de interés en la exposición que se hace del problema del viajante a lo largo de la memoria, por lo que de ahora en adelante asumiremos, en su defecto, que todos los grafos de instancias del problema del viajante referidos a lo largo de la exposición son completos.

**Definición 4.2** *Sea* $G = (V, E)$ *un grafo. Se dice que* $C$ *es un* ***camino*** *sobre* $G$ *si,* $C = (v_1, v_2, ..., v_r) \in V^r = V \times \overset{r}{\cdots} \times V$*, donde* $r \in \mathbb{N}$*,* $r \geq 2$ *se dice que es la* ***longitud del camino*** $C$*. Si además* $C$ *cumple que* $v_r = v_1$ *se dice que* $C$ *es un* ***camino cerrado****,* ***ciclo*** *o* ***circuito*** *sobre* $G$*. En caso contrario, se dice que* $C$ *es un* ***camino abierto****.*

**Notación 4.3** *Sea* $G$ *un grafo y sea* $C = (v_1, v_2, \ldots, v_r)$ *un camino sobre* $G$*, se denotará por* $E(C) = \{(v_i, v_{i+1}) : i = 0, \ldots, r-1\}$ *al conjunto de aristas que conforman el camino* $C$*, y por* $V(C) = \{v \in V : \exists i \text{ tal que } v_i = v,\ 1 \leq i \leq r\}$ *el conjunto de vértices que visita.*

Todo el trabajo realizado sobre grafos se va a centrar en encontrar circuitos con determinadas propiedades. En concreto, son de nuestro interés un tipo concreto de circuitos denominados *hamiltonianos*.

**Definición 4.4** *Sea $G = (V, E)$ un grafo y sea $C = (v_1, v_2, \ldots, v_r)$ un camino sobre $G$. Se dice que $C$ es un* ***camino hamiltoniano*** *sobre $G$ si $v_i \neq v_j$ para cada $i, j$ con $1 \leq i < j \leq r$ y $r = |V|$. Todo camino cerrado y hamiltoniano recibe el nombre de* ***ciclo hamiltoniano****.*

Debido a sus propiedades, no todos los grafos admiten un ciclo hamiltoniano sobre él. Aquellos grafos que sí lo hacen se denominan a su vez *hamiltonianos*.

**Definición 4.5** *Se dice que un grafo es* ***hamiltoniano*** *si es posible construir un ciclo hamiltoniano sobre el mismo.*

De forma trivial, todo grafo completo es hamiltoniano, al estar cualquier par de vértices del grafo unidos por una arista. Sin embargo, si eliminamos la condición de completitud sobre el grafo de una instancia del TSP, la condición necesaria para que exista al menos una solución es que el grafo sea hamiltoniano.

Estos términos de *grafo hamiltoniano* y *ciclo hamiltoniano* se atribuyen al famoso *The Icosian Game* de W.R. Hamilton, quién, a su vez, contribuyó en gran medida al impulso y desarrollo de la teoría de grafos como un campo abierto de estudio a lo largo del siglo XIX.

Con el conjunto de conceptos previamente presentados en esta sección, es posible dar una definición formal del problema del viajante.

**Problema 2: Problema del viajante (TSP)**

Sea $G = (V, E)$ un grafo completo donde $V = \{v_1, v_2, ..., v_n\}$ y $E = \{(v_i, v_j) \,|\, v_i, v_j \in V, i \neq j\}$, sea $C = (c_{ij})$ la matriz $n \times n$ tal que $c_{ij} \geq 0$ es la longitud o coste asociado a la arista $(v_i, v_j)$, y sea $\Pi$ el conjunto de todas las permutaciones circulares sobre el conjunto $\{1, 2, ..., n\}$.

**Problema**: encontrar una permutación circulares $\pi^* \in \Pi$ sobre $\{1, ..., n\}$ tal que:

$$\sum_{i=1}^{n} c_{i\pi(i)^*} = \min_{\pi \in \Pi} \sum_{i=1}^{n} c_{i\pi(i)}. \tag{4.1}$$

Nótese que el hecho de imponer una permutación circular (ver Definición 3.1) se traduce en encontrar un circuito sobre el grafo considerado, que además pasará por todos

y cada uno de sus vértices una, y solo una vez (ciclos hamiltonianos). Todos los circuitos que cumplen las condiciones anteriores reciben el nombre de *rutas*. El valor de (4.1) recibe el nombre de *longitud o coste de la ruta*.

A lo largo de la memoria aparece repetidas veces el término *instancia*, entendido como supuesto o especificación concreta del modelo formal que plantea un problema. Se dice así que el problema es la abstracción que se concreta a través de sus instancias. En el caso del TSP, éstas quedan caracterizadas en la Definición 4.6.

**Definición 4.6** *Una* ***instancia $\mathcal{I}$ del TSP*** *se define como un par $(G, C)$ donde $G = (V, E)$ es un grafo completo con $V = \{v_1, v_2, \ldots, v_n\}$ como conjunto de vértices y $E = \{(v_i, v_j) \,|\, v_i, v_j \in V, i \neq j\}$ como conjunto de aristas; y donde $C = (c_{ij})_{1 \leq i,j \leq n}$ es una matriz verificando que $c_{ij} \geq 0$ es la longitud o coste asociado a la arista $(v_i, v_j), i \neq j$, y $c_{ii} = 0$ si $1 \leq i \leq n$.*

**Ejemplo 4.7** *El siguiente ejemplo considera el grafo $G$ ilustrado en la Figura 4.1 y la matriz de distancias $C$ dada por* (4.2).

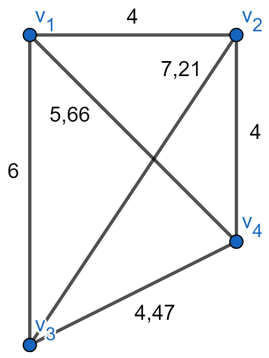



Figura 4.1: Grafo $G$

$$C = \begin{pmatrix} 0 & 4 & 6 & 5,66 \\ 4 & 0 & 7,21 & 4 \\ 6 & 7,21 & 0 & 4,47 \\ 5,66 & 4 & 4,47 & 0 \end{pmatrix} \tag{4.2}$$

*El camino $P = (v_1, v_2, v_4, v_3, v_1)$ es una ruta de mínima longitud, y viene dada por la*

*permutación*

$$\begin{array}{rcl} \pi^* : \{1,2,3,4\} & \longrightarrow & \{1,2,3,4\} \\ \pi(1) & = & 2 \\ \pi(2) & = & 4 \\ \pi(3) & = & 1 \\ \pi(4) & = & 3 \end{array}$$

Se distinguen dos casos en función de si la matriz de costes es o no simétrica. En caso de serlo se dice que el problema es *simétrico*, pues para cualquier par de ciudades $i$, $j$ se tendrá que $c_{ij} = c_{ji}$, es decir, el coste de ir de $i$ a $j$ es el mismo que el de ir de $j$ a $i$. Por otro lado, si la matriz no es simétrica se dice que el problema es *asimétrico*, y que por tanto, existirá un par de ciudades $i$, $j$ tales que el $c_{ij} \neq c_{ji}$. Al primero de estos dos casos es al que nos referiremos a lo largo de la memoria.

El TSP es aplicable a algunos problemas reales surgidos principalmente en entornos empresariales y de investigación. Algunas de las aplicaciones prácticas del problema más destacadas se desglosan en la Sección 4.3.

En el contexto del problema de reordenación de trayectorias (véase Sección 3.5), se consideran las coordenadas de los mensajes asociados a un determinado vuelo como ciudades, y obteniendo las distancias dos a dos que hay entre cada una de esas ubicaciones, el problema de obtener la reordenación correcta de los mensajes ADS-B puede modelarse en términos del TSP considerando una nueva ciudad sobre el grafo tal que conecte con todas las demás ciudades a través de aristas de coste cero, de manera que su inserción no afecte a la solución final del problema. El propósito de la introducción de la ciudad artificial es unir el primer mensaje ADS-B con el último, para que el problema de ordenar estos mensajes se corresponda con el problema del viajante.

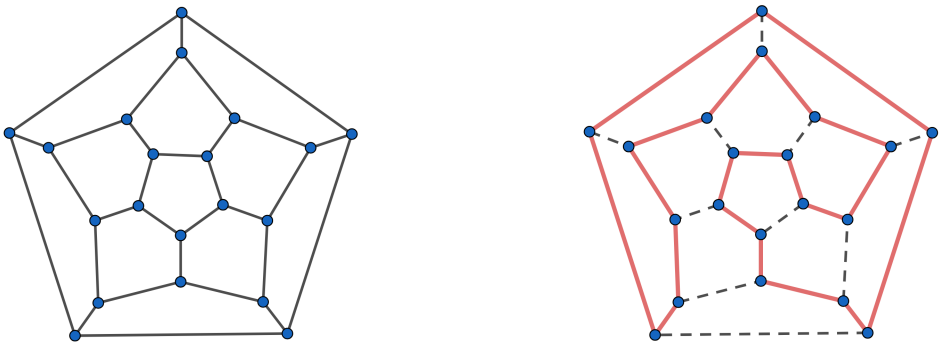
Sin embargo, con este planteamiento del problema la solución que obtendríamos para el problema de reordenación de trayectorias sería el camino abierto más corto que pase por todas las ubicaciones de los mensajes ADS-B una, y solo una vez. Este camino puede no coincidir con el orden temporal natural de los mensajes, aunque parece razonable que lo haga de forma general (se exceptúan por ejemplo vuelos en los que el piloto debe realizar complejas maniobras de aterrizaje implicando en ocasiones rodear repetidas veces el aeropuerto de destino hasta que se recibe la orden de aterrizar por parte del control aéreo del aeródromo). Por otro lado, suponer como solución al problema la dada por el camino antes descrito resulta en una simplificación razonable del problema sin reducir con ello la calidad de la solución final.

## 4.2. Historia del TSP

En esta sección se resumen, en líneas generales, los aspectos más importantes que caracterizan al TSP, siguiendo el enfoque de Lawler et al. [1992].

El problema del viajante, cuya traducción literal del inglés es *problema del vendedor ambulante*, recibe popularmente su nombre de la necesidad que se le presenta a estos profesionales de visitar un determinado conjunto de ciudades por cuestiones de trabajo para después retornar a su lugar de partida, con un claro interés por hacer este trayecto invirtiendo el menor tiempo posible. Suponiendo longitud y tiempo de viaje proporcionales, esta aspiración se traduce en encontrar el camino de mínima distancia que atraviese todas las ubicaciones prefijadas. Esta idea no es más que el objetivo final del TSP.

Los primeros antecedentes del TSP parten del desarrollo de la teoría de grafos a lo largo del siglo XIX, de la mano de matemáticos como Kirkman, quién en 1856 fue el primero en considerar los ciclos hamiltonianos en un contexto general, estableciendo condiciones para su existencia sobre grafos poliédricos. Ese mismo año, Sir William Rowan Hamilton descubrió un álgebra no conmutativa, que denominó *The Icosian Calculus*, cuyos elementos son las acciones o posibles movimientos sobre los vértices del dodecaedro regular, y sobre el cual plantea el problema de encontrar ciclos hamiltonianos bajo ciertas restricciones, como por ejemplo, estableciendo su paso obligado por cinco vértices determinados de forma consecutiva. A raíz de su teoría, Hamilton confeccionó un juego de inteligencia que denominó *The Icosian Game* en el que desafiaba al jugador a encontrar un camino con determinadas propiedades sobre el grafo del dodecaedro regular (ver Figura 4.2).



(a) Grafo del dodecaedro regular

(b) Ciclo hamiltoniano sobre el grafo del dodecaedro regular

Figura 4.2: El juego de inteligencia *The Icosian Game* reta al jugador a encontrar ciclos hamiltonianos sobre el grafo del dodecaedro regular bajo ciertas restricciones

En 1930, Menger dio mayor importancia al coste del camino por ser la medida de la longitud de curvas en el espacio uno de sus temas de estudio favoritos. Surge entonces el denominado *problema del mensajero* o *messenger problem*, que consiste en encontrar el camino abierto más corto que una todos los vértices de un grafo (pensemos en un

mensajero que debe entregar paquetes en varias direcciones prefijadas en el menor tiempo posible). Se diferencia sutilmente del TSP en que no se requiere que las ciudades de partida y de llegada coincidan.

En 1954, George Dantzig, Ray Fulkerson y Selmer Johnson, miembros del grupo de matemáticos de la *RAND Corporation*, recogen el testimonio de Both Flood, quien atestigua que en 1937 Hassler Whitney, en una de sus conferencias, introdujo el nombre de *Traveling Salesman Problem* para denominar al problema igualmente nombrado. Dantzig, Fulkerson y Johnson presentan además uno de los primeros algoritmos de resolución del problema: el algoritmo de *planos cortantes*, basado en programación lineal entera.

Durante esa década el TSP se popularizó en América surgiendo nuevas líneas de investigación. La *RAND Corporation* se convirtió en el principal núcleo investigador y anunció el pago de importantes cantidades económicas por la resolución de instancias concretas del problema a fin de fomentar el desarrollo de nuevos métodos.

Posteriormente, tres matemáticos: Cook en 1971, Karp en 1972 y Lewis en 1973; desarrollaron sendos trabajos en los que indagaban sobre la dificultad de resolución de una serie de problemas, catalogados como $\mathcal{NP}$*-hard* o *hard non-deterministic polynomial time problems*, caracterizados por ser su resolución computacionalmente costosa, no estando probada la existencia de un algoritmo con complejidad computacional polinomial que garantice encontrar una solución (óptima) del problema. De sus estudios se comienza a entrever la equivalencia de todos estos problemas, pues un algoritmo polinomial que resolviese alguno de ellos también resolvería el resto. Además, fue posible probar que ciertos casos concretos del TSP eran $\mathcal{NP}$*-hard*, entre ellos el TSP euclídeo, mencionado en la sección anterior.

La aparente dificultad del TSP motiva la aparición de dos corrientes de algoritmos. La primera persigue alcanzar la solución óptima (una ruta de longitud mínima) aun a riesgo de no encontrarla en un tiempo razonable. La segunda, sin embargo, prioriza el coste computacional a la calidad de la solución, conformándose con soluciones que simplemente se aproximen a la más corta, sin llegar a ser óptimas. En la primera opción se encuentran los algoritmos de *ramificación y acotación* (o *branch and bound*) , así como la programación dinámica, que solo sirvió para resolver pequeñas instancias del problema. Para la segunda opción, surgen los métodos basados en *heurísticas de mejora* (cuyo análisis es objeto de este Trabajo Fin de Grado), que dada una primera solución inicial realizan modificaciones sobre ella con el fin de obtener nuevas soluciones más cortas.

En 1980, el TSP con 318 ciudades fue resuelto por Crowder y Padberg mediante la división del problema en 3 fases. La primera consistió en el uso del *algoritmo de Lin-Kernighan* (que se introduce en la Sección 4.5.2) para encontrar una ruta de partida y se ejecuta un procedimiento basado en *símplex* sobre un problema de programación lineal con restricciones obteniendo una cota inferior de la longitud de la solución óptima. En la segunda fase se fija un subconjunto de variables para, en la tercera fase, hacer, de nuevo, uso de programación lineal. Se sabe que esta tercera fase conllevó aproximadamente 6

minutos de tiempo de CPU.

Más recientemente, destacan los supuestos que el grupo de investigación formado por David Applegate, Robert Bixby, Vasek Chvátal y William Cook logró resolver a través de un algoritmo de creación propia que denominaron *Concorde*. Mediante este método han obtenido la solución de instancias de tamaño no resuelto hasta entonces, siendo la más reciente una con 85.900 ciudades, en el año 2006 [Cook, 2012].

## 4.3. Aplicaciones generales del TSP

El TSP es un problema de interés científico debido esencialmente a la complejidad computacional que requiere su resolución. Pero además, es ampliamente estudiado por sus múltiples utilidades prácticas.

En realidad, muchos supuestos pueden ser modelados a través del TSP (i.e. planteados como instancias del problema de cuya solución se deriva la del supuesto modelado). Así, existen muchas aplicaciones prácticas del problema del viajante, una de las cuales es la reordenación de los mensajes ADS-B asociados a una trayectoria.

Las aplicaciones más claras del TSP, tal y como fue concebido, están ligadas al enrutamiento de vehículos, donde se dispone de un conjunto de ubicaciones que deben ser visitadas por un determinado vehículo recorriendo la menor distancia posible. Pensemos por ejemplo en una compañía con servicio de venta por Internet que debe entregar una serie de paquetes en un conjunto de direcciones prefijadas a través de uno de sus vehículos.

Existen otras aplicaciones del problema del viajante entre las que se encuentran las siguientes [Laporte, 1992]:

1. El problema del *cableado de ordenador*. Algunos sistemas informáticos pueden describirse en términos de módulos que se unen entre sí por medio de cables. Se desea unir todos los módulos (ciudades) por medio de cables (aristas) de mínima longitud y de tal manera que cada módulo quede unido a otros dos.

2. El problema del *papel de pared*. Se quieren cortar $n$ hojas de un rollo de papel tapiz en el cual se repite un patrón de longitud 1. Para cada hoja $i$, se denota $a_i$ y $b_i$ como los puntos del patrón donde comienza y finaliza la hoja. Entonces, si se corta la hoja $j$ justo después que la hoja $i$ se obtiene un desperdicio $c_{ij}$ dado por

$$c_{ij} = \begin{cases} a_j - b_i & si \quad b_i \leq a_j, \\ 1 + a_j - b_i & si \quad b_i > a_j. \end{cases}$$

   Se pretende por tanto obtener las $n$ hojas del rollo con el menor desperdicio posible. Para definir este problema en términos del TSP se consideran las hojas como ciudades y el gasto de papel de cortar la hoja $j$ justo después que la hoja $i$ como $c_{ij}$

(longitud del camino que une la ciudad $i$ con la ciudad $j$). A mayores, sería necesario introducir una hoja artificial $n+1$ a modo de unión entre la primera y la última hoja cortadas tal que $c_{ij} = 0$ si $i = n+1$ ó $j = n+1$. De esta forma se tendrá que acabar en la ciudad de partida.

3. El problema de la *secuenciación de trabajos*. Se supone que una máquina debe realizar $n$ trabajos de forma secuencial y que $c_{ij}$ es el tiempo que la máquina invertiría en pasar del trabajo $i$ al trabajo $j$. Entonces de nuevo, para minimizar el tiempo invertido en cambiar de un trabajo a otro (o *change-over time*) es posible formular el problema en términos del TSP introduciendo un trabajo artificial de coste 0, de forma análoga al problema anterior.

4. En *cristalografía*, algunos experimentos consisten en tomar un gran número de medidas sobre cristales a través de un detector. Cada muestra debe montarse en un aparato que debe ser posicionado para poder hacer la medición, invirtiendo tiempo en cambiar la posición del detector desde la que disponía para realizar la última medición hasta la requerida para realizar la actual. Como en los anteriores, añadiendo una ciudad artificial de coste 0 que una la primera muestra con la última obtenemos una instancia del TSP donde las ciudades son las muestras a examinar y el coste de las aristas que las unen es el tiempo invertido por el detector para cambiar su posición.

## 4.4. Algoritmos de resolución

Tal y como se adelanta en la Sección 4.2, los algoritmos que tratan de resolver el TSP se clasifican, según garanticen o no encontrar la solución óptima del problema al ser ejecutados, en *algoritmos exactos* y *algoritmos de aproximación*. En esta sección se mencionan los más importantes de cada uno de estos dos tipos.

### 4.4.1. Los algoritmos exactos

Los *algoritmos exactos* son aquellos que buscan obtener la solución óptima del problema independientemente del coste computacional requerido. Estos algoritmos aseguran la obtención de la solución óptima, lo que implica un alto coste computacional ligado a las propiedades del TSP. Es por ello que su uso es preferible sobre instancias pequeñas, cuando el tiempo de ejecución de estos algoritmos aun no es desmedido.

Dentro de estos algoritmos encontramos el rudimentario *ataque por fuerza bruta* que prueba todas las permutaciones circulares posibles sobre los nodos del grafo, quedándose con aquella que minimiza la expresión (4.1). La complejidad computacional de este algoritmo es por tanto $\mathcal{O}(n!)$ siendo $n$ el número de ciudades de la instancia, asumible sobre

instancias pequeñas, pero impensable sobre otras más grandes (con $n = 20$ este número es ya del orden de $2 \cdot 10^{18}$).

Una de las familias de algoritmos exactos más exitosas a lo largo de la historia del problema es la de los métodos *branch and bound*, o de *ramificación y acotación*, que consisten en dividir la instancia en otras más pequeñas, calculando cotas de la longitud de la ruta más corta de cada parte, y que se obtienen de "relajar" las hipótesis del problema. Tras obtener una solución para cada subinstancia, el algoritmo obtiene la ruta de la instancia principal, que resulta ser una solución óptima.

Los algoritmos *branch and bound* se basan en programación lineal entera, que parte de una formulación del problema a resolver. Para el TSP, Dantzig, Fulkerson y Johnson desarrollaron en 1954 una de las primeras [Lawler et al., 1992], y que consiste en que dada una instancia $\mathcal{I} = (G, C)$ donde $G$ es un grafo de $n$ vértices y $C = (c_{ij})_{1 \leq i,j \leq n}$ es la matriz de costes, se quiere minimizar la expresión

$$\sum_{i \neq j} c_{ij} x_{ij} \quad \text{con } x_{ij} \in \mathbb{Z}, \quad \text{para todo } i, j \text{ en } 1 \leq i, j \leq n \tag{4.3}$$

sujeto a

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1, ..., n; \tag{4.4}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1, ..., n; \tag{4.5}$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1; \tag{4.6}$$

$$S \subset V, \qquad 2 \leq |S| \leq n - 2; \tag{4.7}$$

$$x_{ij} \in \{0, 1\}, \qquad i, j = 1, \ldots, n, \ i \neq j. \tag{4.8}$$

Es necesario también citar el algoritmo de *Held-Karp*, que aparece en 1962 de la mano de Michael Held y Richard Karp, y que se basa en programación dinámica [Cook, 2012]. Capaz de encontrar la solución óptima de cualquier instancia del TSP, este algoritmo fija una ciudad como punto de partida, a partir de la cual va construyendo caminos formados por cada vez más vértices y de los cuales va almacenando su longitud. De esta forma, dado un vértice de destino, y un subconjunto de vértices intermedios, el algoritmo halla cual es la forma más corta de recorrer todos los nodos intermedios partiendo desde el primero fijado al comienzo hasta el último señalado. El método itera hasta abarcar todos los vértices. A pesar de garantizar la solución óptima, el coste computacional de este algoritmo es $\mathcal{O}(n^2 2^n)$ lo que lo inhabilita para ser utilizado sobre instancias grandes.

En la actualidad, *Concorde* es uno de los algoritmos más destacados. Implementado por David Applegate, Robert Bixby, Vasek Chavátal y William Cook, consiste en uno de los desarrollos más recientes del *método de ramificación y acotación* sobre árboles de

búsqueda. Este algoritmo, cuyo código está disponible en la red, fue utilizado para obtener la solución óptima a instancias del problema de tamaño *record* hasta la fecha. Entre ellas se encuentran supuestos de: 3.038 ciudades en 1992, 13.509 en 1998, 24.978 en 2004 y 85.900 en 2006 Cook [2012].

### 4.4.2. Los algoritmos de aproximación

Por otro lado se encuentran los *algoritmos de aproximación*, que son aquellos que dan prioridad a un tiempo de ejecución razonable frente a encontrar la solución óptima.

Mientras que ejecutar un algoritmo exacto para el problema del viajante puede requerir un coste computacional inasumible para instancias de gran tamaño, un algoritmo de aproximación, aunque no encuentre la solución óptima, es mucho menos complejo computacionalmente, por lo que es posible ejecutarlos sobre instancias de mayor tamaño.

Los más sencillos de estos métodos son las *heurísticas constructivas*, que engloban una serie de métodos cuya mecánica consiste en construir una ruta partiendo de un nodo al que van uniendo sucesivamente ciudades en cada iteración del algoritmo. Algunos de los más destacados son los siguientes [Gutin y Punnen, 2007]:

- **Método de vecinos más próximos** (o *nearest neighbor*): es el método constructivo más simple. Consiste en tomar el conjunto $V$ de vértices del grafo de una instancia y elegir un nodo $v_1 \in V$ arbitrario como punto de partida. Para cada iteración $i = 2, 3, \ldots, n$, siendo $n = |V|$, se elige un nuevo nodo $v_i \in V \setminus \{v_1, v_2, \ldots, v_{i-1}\}$ tal que $c_{i-1,i}$ sea el mínimo posible (siendo $c_{i-1,i}$ el coste de la arista $(v_{i-1}, v_i)$), es decir, a cada paso se añade el nodo más cercano, de entre los no escogidos aun, al último añadido.
- **Método de vecinos más próximos iterativo** (o *repetitive nearest neighbour*): variante algo más compleja del método de vecinos más próximos que consiste en comenzar este método desde cada vértice del grafo, eligiendo como ruta la obtenida de menor longitud. Su complejidad computacional por lo general es $\mathcal{O}(n^2)$.
- **Greedy** (*codicioso*): se comienza ordenando de menor a mayor coste todas las aristas. A continuación, se contruye una ruta, haciendo uso de esta ordenación, comenzando con la arista más corta y añadiendo una nueva arista $(v, v')$ siempre y cuando $v$ o $v'$ no son alcanzados por dos aristas de la ruta en construcción y además la nueva arista no completa un ciclo con menos de $n$ vértices siendo $n$ el número de vértices del grafo de la instancia. Lleva asociado un orden de complejidad $\mathcal{O}(n^2 log(n))$.
- **Inserción más barata** (o *cheapest insertion*): consiste en insertar, a cada iteración, un nuevo vértice entre dos nodos de la ruta en construcción de la forma que menos incremente su longitud, esto es, si $T_m = (v_1, v_2, \ldots, v_{m-1}, v_m)$ es el camino en construcción con $m > 1$ y $(v_i, v_{i+1}) \in E(T_m)$ es la arista en el camino actual que minimiza $d(v_i, v_k) + d(v_k, v_{i+1}) - d(v_i, v_{i+1})$, donde $d$ es la función distancia (coste)

entre cada par de vértices del grafo, entonces se obtiene un nuevo camino $T_{m+1}$ a partir de $T_m$, eliminando la arista $(v_i, v_{i+1})$ y añadiendo $(v_i, v_k)$ y $(v_k, v_{i+1})$ en el lugar que ésta ocupaba, esto es, $T_{m+1} = (v_1, v_2, \ldots, v_i, v_k, v_{i+1}, \ldots, v_m)$.

- **Inserción más cercana** (o *nearest insertion*): consiste en iniciar el algoritmo con una ciudad y su vecino más próximo. A cada iteración se añade la ciudad no perteneciente a la ruta en construcción cuya distancia con respecto a cualquiera de las ciudades de la ruta en construcción sea mínima.
- **Inserción más lejana** (o *farthest insertion*): como en los anteriores, consiste en comenzar con un nodo y su vecino más cercano. Se itera añadiendo a la ruta en construcción $P$ el nodo $v$ tal que maximice el valor de

$$\text{mín}\{d(v, v') : v' \in V(P)\}, \tag{4.9}$$

  donde $d$ es la función distancia (coste) entre cada par de vértices del grafo.
- **Inserción arbitraria** (o *arbitrary insertion*): difiere del anterior en que en cada iteración se añade a la ruta un nodo aleatorio no perteneciente a la ruta en construcción.

Una de las familias de algoritmos de aproximación más importantes son los *búsqueda local*, también llamados *heurísticas de mejora local*, cuyo análisis es objeto de este Trabajo Fin de Grado y que se profundizan en la Sección 4.5). Estos métodos parten de una ruta inicial (solución factible) que van modificando a cada paso para obtener otras rutas más cortas. Se caracterizan por la técnica de modificación que utilicen, que restringe la cantidad de rutas que es posible obtener desde una dada en una sola iteración.

Los más conocidos son los algoritmos de la familia *k-Opt*, que en cada iteración buscan una nueva solución reemplazando $k$ aristas de la ruta actual por otras $k$ nuevas mediante un procedimiento de modificación que recibe el nombre de *k-change* (ver Definición 4.8). Los métodos más conocidos de esta familia son el *2-Opt* y el *3-Opt*. Por otro lado, encontramos una implementación más sofisticada de los algoritmos *k-Opt* en el *algoritmo de Lin-Kernighan*, que a diferencia de los anteriores, varía el valor de la $k$ a lo largo de su ejecución con el fin de adaptarse al tipo de modificaciones que resultarán más efectivas a cada instante. Por otro lado, encontramos el algoritmo *Or-Opt* que basa su procedimiento en la escisión de cadenas de vértices de la ruta actual para unirlas en otra posición.

Otros algoritmos de aproximación con aplicación al TSP son los siguientes:

- *Simulated annealing* (*enfriamiento simulado*): se introduce a través de uno de los trabajos de Kirkpatrick, Gelatt y Vecchi en 1983 como método de resolución del TSP con unos resultados bastante prometedores. Mientras que un algoritmo típico de búsqueda local sigue la estrategia *hill-climbing*, consistente en realizar solo movimientos hacia soluciones "vecinas" (alcanzables en una iteración a través de una técnica de modificación fija) mejores que la actual, el *simulated annealing* realiza saltos arbitrarios sobre el espacio de estados con una cierta frecuencia a fin de no

quedar atrapado en máximos locales. La frecuencia y la distancia de estos saltos va disminuyendo con el paso de las iteraciones (se produce el *enfriamiento*).

- Los *algoritmos de redes neuronales*: basados en el funcionamiento del cerebro humano. Sus componentes básicos son unidades de cómputo simples que reciben el nombre de *neuronas artificiales* que se comunican entre si a través de una red de conexiones que simulan el procesamiento en paralelo que ocurre en el cerebro humano. Este modelo fue aplicado al TSP por Hopfield y Tank en 1985.
- Los *algoritmos genéticos*: trabajan con un conjunto de soluciones del problema. En cada iteración, se eligen las mejores soluciones (*selección*), las cuales se combinan para obtener otras nuevas (*cruce*), finalmente se alteran (*mutación*) y se repite el proceso tomando estas nuevas soluciones como entrada de la siguiente iteración.

## 4.5. Heurísticas de mejora local

En línea con lo expuesto hasta ahora, existe una gran colección de algoritmos aplicables a la resolución del problema del viajante. No obstante, este trabajo se enfoca en un subconjunto de ellos, denominados *heurísticas de mejora local*. Estos algoritmos destacan por obtener por lo general buenas soluciones requiriendo su ejecución un coste computacional generalmente moderado. En esta sección se ilustran los más importantes.

Los *algoritmos de búsqueda local* o *heurísticas de mejora local* son considerados en Korte y Vygen [2006] como la técnica más adecuada para obtener buenas soluciones a gran parte de los supuestos del problema del viajante. La mecánica de todas ellas consiste en iniciar su ejecución con una ruta de partida (un ciclo hamiltoniano, también llamado ruta, sobre el grafo de la instancia), obtenida a través de cualquier otra técnica (lo aconsejable es que sea un método constructivo, por su sencillez), e intentar "mejorarla" (encontrar una ruta más corta) mediante la realización de pequeñas modificaciones sobre ella a cada iteración. Es en la forma de realizar esas modificaciones (técnicas de las que hace uso) donde una heurística difiere sobre las demás.

Distinguimos dos clases de estos algoritmos en función de los mecanismos de mejora utilizados, de las cuales nos centraremos en la primera de ellas:

- **Algoritmos de intercambio de aristas (EE: *edge exchanges*)**: el procedimiento de modificación realizado en estos algoritmos consiste en eliminar a cada iteración un conjunto de aristas de la ruta actual reemplazándolas por otras de manera que formen una nueva ruta. Los métodos de este tipo más conocidos son los de la familia *k-Opt* o el *algoritmo de Lin-Kernighan*.
- **Algoritmos de intercambio de cadenas (CE: *chain exchanges*)**: en cada iteración, se considera una cadena de vértices de la ruta que son desligados del

resto. A continuación se vuelven a unir a la ruta en otra posición diferente mediante la eliminación de una arista. Un ejemplo de estos algoritmos es el *Or-Opt*.

### 4.5.1. Los algoritmos de la familia *k-Opt*

Dentro de las heurísticas de mejora local, los algoritmos de implementación más sencilla son los de la familia *k-Opt*. Estos métodos se caracterizan por construir, a cada iteración, una nueva ruta mediante un procedimiento que recibe el nombre de *k-change*. Se compara la nueva ruta con la ruta actual, y en caso de ser la nueva ruta más corta, sustituyen la ruta actual por la nueva. El procedimiento de modificación *k-change*, del que hacen uso estos algoritmos, consiste en suprimir $k$ aristas de la ruta actual reemplazándolas por otras $k$ dando como resultado una nueva ruta. Esta nueva ruta será considerada solo si es más corta que la anterior. El valor de $k$ es por tanto fijo para cada iteración del algoritmo siendo éste dependiente de la heurística de la familia *k-Opt* utilizada. Así, para el *2-Opt*, $k = 2$; para el *3-Opt*, $k = 3$; y así sucesivamente.

Más formalmente, un *k-change* viene definido de la forma siguiente:

**Definición 4.8** *Dada una instancia $\mathcal{I} = (G, C)$ donde $G = (V, E)$ es un grafo de $n$ vértices, y sea $R$ una ruta sobre $G$. Una modificación de $R$ que da como resultado otra ruta $R'$ sobre $G$ se dice que es de tipo* ***k-change*** *si a partir de $R$, dado por*

$$R = (v_1^1, v_2^1, \ldots, v_{n_1}^1, v_1^2, v_2^2, \ldots, v_{n_2}^2, \ldots, v_1^k, v_2^k, \ldots, v_{n_k}^k, v_1^1),$$

*se obtiene la ruta $R'$ dada por*

$$R' = (v_1^1, v_2^1, \ldots, v_{n_1}^1, v_{n_2}^2, v_{n_2-1}^2, \ldots, v_1^2, v_{n_3}^3 \ldots, v_1^{k-1}, v_{n_k}^k, v_{n_k-1}^k, \ldots, v_1^k, v_1^1).$$

Los algoritmos de la forma *k-Opt* son muy utilizados en la práctica, al obtener de forma general buenas soluciones a instancias del problema del viajante sin requerir para ello un elevado coste computacional. Sin embargo, el estudio que incluye Englert et al. [2007] permite concluir la existencia de instancias del problema del viajante para las que el algoritmo *2-Opt* puede alcanzar una complejidad computacional de orden exponencial, y de la misma forma, en Chandra et al. [1999] se generaliza esta tesis para cualquier algoritmo de la familia *k-Opt*.

En Korte y Vygen [2006] se presentan los algoritmos *k-Opt* en pseudo-código tal y como se muestra en el Algoritmo 1.

**Algoritmo 1** Algoritmo k-Opt

**Entrada:** Una instancia del TSP $\mathcal{I} = (G, C)$, con $G = (V, E)$.
**Salida:** Una ruta $R$

```
1: Sea R una ruta
2: Sea S la familia de subconjuntos de E(R) con k elementos
3: para S ∈ S y toda ruta R' con E(R') ⊇ E(R) \ S hacer
4:    si Σ_{(v_i,v_j)∈E(R')} c_{i,j} < Σ_{(v_i,v_j)∈E(R)} c_{i,j} entonces
5:       Asignar R := R'
6:       Sea S la familia de subconjuntos de E(R) con k elementos
7:    fin si
8: fin para
9: devolver R
```

A pesar de hacer uso de una mecánica similar, cada uno de los algoritmos de la familia *k-Opt* se comportan de distinta forma y por tanto se predispone que alcanzarán soluciones de características diferentes requiriendo para ello distintos órdenes de complejidad en función del valor de $k$ elegido y, por supuesto, del número de vértices de la instancia y la localización de los mismos, esto es, los valores de la matriz de costes asociada.

El algoritmo *2-Opt*, a cada iteración prueba un *2-change* sobre la ruta actual. Un ejemplo de un *2-change* se ilustra en la Figura 4.3, donde podemos apreciar que para obtener la nueva ruta simplemente se han suprimido dos aristas $(v_1, v_3)$ y $(v_2, v_4)$, que han sido reemplazadas por $(v_1, v_2)$ y $(v_3, v_4)$. Las modificaciones de tipo *2-change* son las de menor alcance (que afecta a menos aristas de la ruta), pero también las más finas, en cuanto a que permiten mejorar rutas muy próximas a una solución óptima. Esto implica que el algoritmo funcionará peor cuando la ruta actual diste mucho de la solución optima (en cuyo caso es preferible un algoritmo *k-Opt* con un valor de $k$ más elevado), pero va mejorando a medida que se aproxima a una solución óptima pues es entonces cuando se requieren modificaciones más pequeñas, y que por tanto afecten a menos aristas.

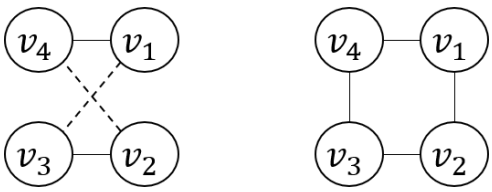



Figura 4.3: Ilustración de un 2-change

En la Figura 4.4 se ilustra una modificación de tipo *3-change*. En la línea de lo expuesto en el párrafo anterior, un *3-change* realiza modificaciones de mayor alcance, pero a su vez menos refinadas que las que puede obtener un *2-change*, lo que predispone a este último como menos preferible en caso de disponer de una ruta actual más próxima a la solución óptima en términos de longitud.

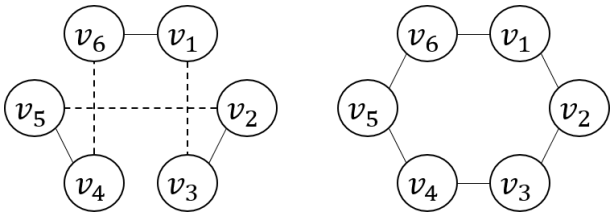
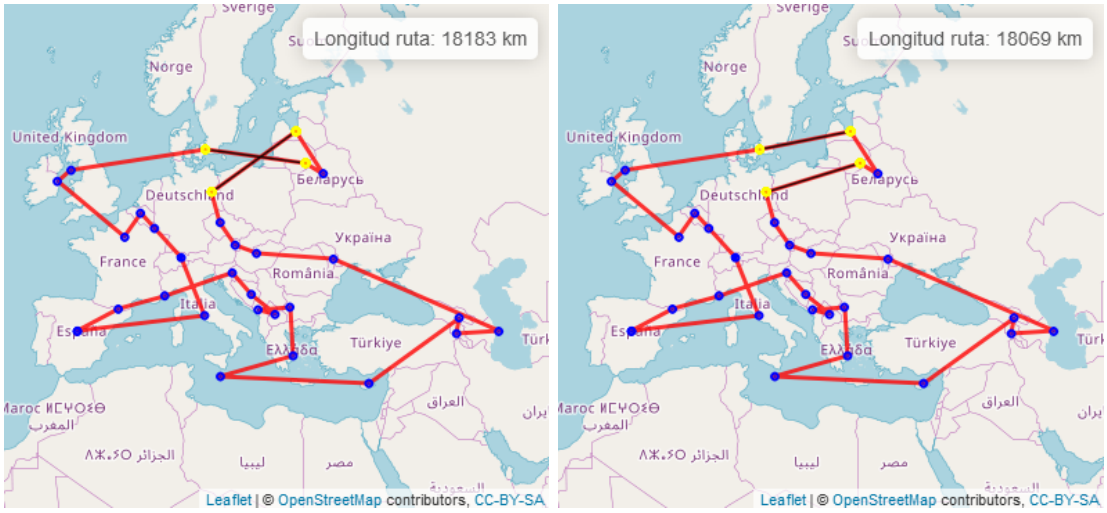



Figura 4.4: Ilustración de un 3-change

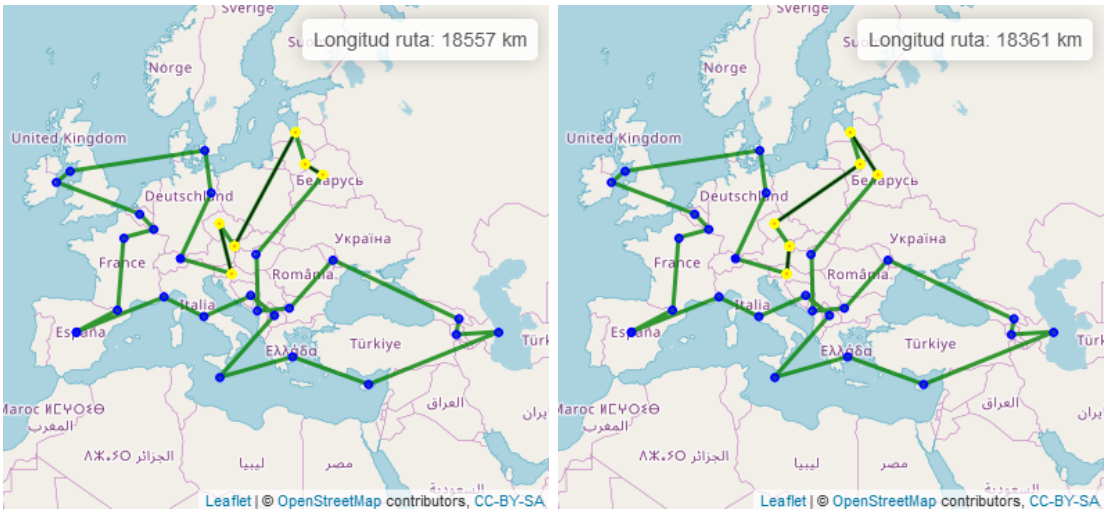



(a) Ruta actual (b) Ruta mejorada

Figura 4.5: Ilustración de un *2-change* durante la ejecución del *2-Opt* sobre una instancia con 30 ciudades. En negro y amarillo, aristas y nodos afectados respectivamente. Creado con `Leaflet`

Las Figuras 4.5 y 4.6 ilustran una mejora realizada por los algoritmos *2-Opt* y *3-Opt* respectivamente sobre un mismo supuesto del TSP con 30 ciudades situadas en Europa.

En conclusión, los algoritmos de la familia *k-opt* muestran un comportamiento estático, en el sentido de que admiten un solo tipo de modificación que involucra a un número fijo de aristas de la ruta actual, que son eliminadas y reemplazadas por otras nuevas en cada iteración. Sin embargo, como se ha visto, no en todas las situaciones se requieren modificaciones que involucren al mismo número de aristas, sino que en ocasiones será preferible la realización de cambios de mayor alcance mientras que a medida que la longitud de las rutas encontradas va disminuyendo se prefieren cambios más finos. Persiguiendo esto, el algoritmo de *Lin-Kernighan* propone que estas modificaciones involucren a un número variable de aristas a cada iteración, de manera que sea el propio algoritmo el que se adapte de forma dinámica al alcance de la modificación requerido.




(a) Ruta actual (b) Ruta mejorada

Figura 4.6: Ilustración de un *3-change* durante la ejecución del *3-Opt* sobre una instancia con 30 ciudades. En negro y amarillo, aristas y nodos afectados respectivamente. Creado con `Leaflet`

### 4.5.2. El algoritmo de Lin-Kernighan

Para poder hablar del funcionamiento de este algoritmo es necesario introducir los resultados teóricos que hay debajo del mismo. Nos guiaremos del enfoque dado en Korte y Vygen [2006], que centra su exposición en el concepto de *camino alterno*. Un **camino alterno** sobre un grafo puede entenderse como un camino tal que, dada una ruta, el camino alterna aristas de la ruta con aristas no pertenecientes a ésta. Más formalmente, un camino alterno puede definirse de la forma siguiente:

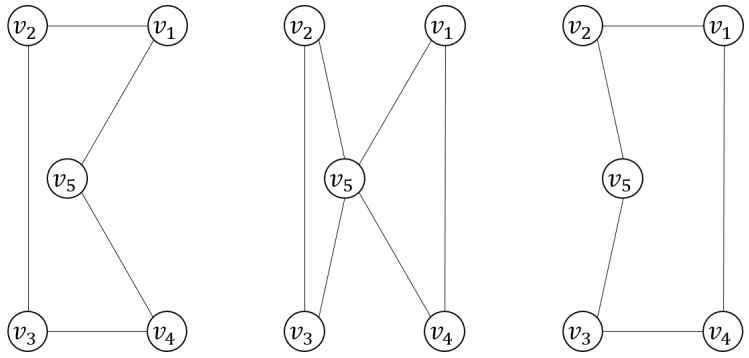
**Definición 4.9** *Sea $\mathcal{I} = (G, C)$ una instancia del TSP, con $C = (c_{ij})_{i,j=1}^{n}$ y sea $R$ una ruta sobre $G$. Un* ***camino alterno*** *$P = (v_1, v_2, \ldots, v_{2m+1})$ sobre $G$ es aquel que verifica $(v_i, v_{i+1}) \neq (v_j, v_{j+1})$ para todo $1 \leq i < j < 2m+1$, y $(v_i, v_{i+1}) \in E(R)$ si, y sólo si $i$ es impar.*

*La* ***ganancia de P*** *viene dada por*

$$g(P) = \sum_{i=0}^{m-1} c_{2i+1,2i+2} - c_{2i+2,2i+3}$$

*Se dice que $P$ es* ***adecuado*** *si $g(v_1, v_2, \ldots, v_{2i+1}) > 0$ para todo $i \in \{1, \ldots, m\}$.*

**Notación 4.10** *Sean $A$ y $B$ dos conjuntos, se denota por $A \triangle B$ a la diferencia simétrica de los conjuntos $A$ y $B$. Esto es, $A \triangle B = (A \setminus B) \cup (B \setminus A)$.*




(a) Ruta $R$ sobre el grafo

(b) Camino alterno $P$

(c) Nueva ruta con aristas $E(P) \triangle E(R)$

Figura 4.7: Ejemplo de camino alterno

La Figura 4.7 ilustra la principal aplicación de los caminos alternos al TSP. Dada una ruta (imagen (a)), es posible encontrar un camino alterno (imagen (b)), que combinado con la ruta, de como resultado otra ruta más corta que la anterior (imagen (c)). Esta idea, es la que lleva a la práctica el *algoritmo de Lin-Kernighan*.

El Lema 4.12 establece el pilar teórico fundamental sobre el que se asienta el funcionamiento del algoritmo *Lin-Kernighan*.

**Notación 4.11** *Sea $\mathcal{I} = (G, C)$ una instancia del TSP, sea $G = (V, E)$ y $X \subseteq E$ no vacío. Se denota por $Coste(X) = \sum_{(v_i,v_j)\in X} c_{i,j}$ .*

**Lema 4.12 (Lin & Kernighan, 1973)** *Sea $R$ una ruta. Si existe un camino alterno $P$ con $g(P) > 0$, entonces:*

*(a) $Coste(E(R) \triangle E(P)) = Coste(E(R)) - g(P)$,*

*(b) existe un camino alterno adecuado $Q$ tal que $E(Q) = E(P)$.*

Nótese que, si se encuentra un camino alterno cerrado y adecuado $P$ sobre la ruta $R$, tal que $E(R) \triangle E(P)$ forman una nueva ruta sobre el grafo, por (a) es posible concluir que la nueva ruta es más corta que $R$.

El algoritmo de *Lin-Kernighan*, basándose en esta lógica, prueba a cada iteración un camino alterno sobre la ruta actual, y aplica el razonamiento anterior para determinar si una nueva ruta encontrada es más corta o no que la actual. La Figura 4.8 ilustra este procedimiento. Considerando el grafo de 8 vértices mostrado, describiendo un octógono regular, $R$ es la ruta sobre el grafo, ilustrada por la imagen (a), dada por

$$R = (v_1, v_2, v_7, v_6, v_5, v_8, v_4, v_3, v_1),$$

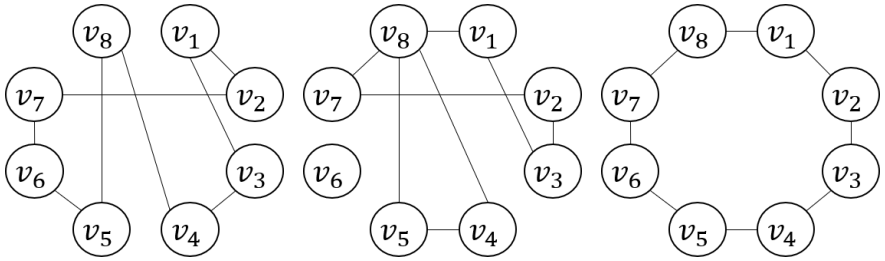
mientras que $P^*$ es el camino alterno cerrado, mostrado en la imagen B), dado por el siguiente conjunto de aristas:

$$P^* = (v_1, v_3, v_2, v_7, v_8, v_5, v_4, v_8, v_1).$$

Nótese la alternancia entre la pertenencia y no pertenencia de las aristas del camino a la ruta $R$. Al hacer la diferencia simétrica entre ambos conjuntos de aristas se obtiene una nueva ruta

$$E(R) \triangle E(P^*) = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_5), (v_5, v_6), (v_6, v_7), (v_7, v_8), (v_8, v_1)\},$$

la cual se ilustra en la imagen (c) y, suponiendo que el coste de las aristas es proporcional a la distancia euclídea entre los vértices que unen y sabiendo que la longitud de los lados de cualquier octógono regular es menor que cualquier arista que una dos vértices no consecutivos del mismo, parece claro que la nueva ruta $E(R) \triangle E(P^*)$ es más corta que $E(R)$.




(a) Ruta $R$ (b) Camino alterno $P^*$ (c) Ruta $E(R) \triangle E(P)$

Figura 4.8: Combinación de un camino alterno $P$ y una ruta $R$ sobre el grafo del octógono regular

Nótese que, basar las modificaciones en la búsqueda de caminos alternos cerrados y adecuados dota al algoritmo de una mayor adaptabilidad a las necesidades concretas de ejecución, pues un camino alterno puede alterar un número variable de aristas de la ruta actual. Así, para la ruta

$$R = (v_1, v_3, v_2, v_4, v_5, v_6, v_7, v_8, v_1),$$

no existe ningún camino alterno $P$ con 8 aristas tal que $E(R) \triangle E(P)$ sea la solución óptima. En esta situación se requiere un refinamiento algo más preciso, y el camino alterno cerrado y adecuado

$$P^* = (v_1, v_2, v_4, v_3, v_1),$$

sí logra que $E(R) \triangle E(P^*)$ sea la solución óptima.

El algoritmo de *Lin-Kernighan* por tanto, combina las ventajas de cada uno de los algoritmos de la familia *k-Opt*, y en la práctica ha demostrado un buen comportamiento en instancias pequeñas y medianas del problema del viajante (en términos del número de vértices de éstas).

**Algoritmo 2** Algoritmo de Lin-Kernighan

**Entrada:** Una instancia $\mathcal{I} = (G, C)$, con $G = (V, E)$ del TSP. Dos parámetros $p_1, p_2 \in \mathbb{N}$
**Salida:** Una ruta $R$

1: Sea $R$ una ruta.
2: $V_0 := V$, $i := 0$ y $g^* := 0$.
3: **si** $V_i = \varnothing$ y $g^* > 0$ **entonces**
4: $\quad E(R) := E(R) \vartriangle E(P^*)$ y volver a 2.
5: **fin si**
6: **si** $V_i = \varnothing$ y $g^* = 0$ **entonces**
7: $\quad i := \text{mín}\{i - 1, p_1\}$
8: $\quad$ **si** $i < 0$ **entonces**
9: $\quad\quad$ Fin de ejecución.
10: $\quad$ **si no**
11: $\quad\quad$ Volver a 3.
12: $\quad$ **fin si**
13: **fin si**
14: Sea $v_i \in V_i$, $V_i := V_i \setminus \{v_i\}$.
15: **si** $i$ impar, $i \geq 3$, $E(R) \vartriangle E((v_0, v_1, \ldots, v_{i-1}, v_i, v_0))$ es ruta y $g((v_0, v_1, \ldots, v_{i-1}, v_i, v_0)) > g^*$ **entonces**
16: $\quad P^* := (v_0, v_1, \ldots, v_{i-1}, v_i, v_0)$, $g^* := g(P^*)$.
17: **fin si**
18: **si** $i$ impar **entonces**
19: $\quad V_{i+1} := \{v \in V \setminus \{v_0, v_i\} : (v_i, v) \notin E(R) \cup E((v_0, v_1, \ldots, v_{i-1})), g((v_0, v_1, \ldots, v_{i-1}, v)) > g^*\}$.
20: **fin si**
21: **si** $i$ par, $i \leq p_2$ **entonces**
22: $\quad V_{i+1} := \{v \in V : (v_i, v) \in E(R) \setminus E((v_0, v_1, \ldots, v_i))\}$.
23: **fin si**
24: **si** $i$ par, $i > p_2$ **entonces**
25: $\quad V_{i+1} := \{v \in V : (v_i, v) \in E(R) \setminus E((v_0, v_1, \ldots, v_i)),$
$\quad (v, v_0) \notin E(R) \cup E((v_0, v_1, \ldots, v_i))$, $E(R) \vartriangle E((v_0, v_1, \ldots, v_i, v, v_0))$ es una ruta$\}$.
26: **fin si**
27: $i = i + 1$, volver a 3.
28: **devolver** $R$

En estas condiciones, se define el *algoritmo de Lin-Kernighan*, el cual se focaliza en buscar un camino alterno cerrado y adecuado $P$. En cada iteración comprueba todas las posibilidades hasta encontrarse un camino con las propiedades anteriores o hasta que se de la condición de parada, dependiente de los dos parámetros $p_1$ y $p_2$. Una versión en pseudo-código del *algoritmo de Lin-Kernighan*, presente en Korte y Vygen [2006], se ilustra en el Algoritmo 2. El código de este algoritmo puede dividirse en cinco partes, de manera que cada parte puede implementarse como un procedimiento con entidad propia que actúa sobre las distintas variables del algoritmo.

- En una primera parte (línea 1) se genera una ruta de partida. El autor da libertad a la manera en que se obtiene esa ruta, por lo que una buena forma de generar ésta podría ser haciendo uso de un método constructivo (ver Capítulo 4.1). Sin embargo en el contexto del problema de reordenación de trayectorias la ruta inicial viene dada por el orden creciente de su *timestamp* asociado.

- En la segunda parte del código (línea 2) se inicializan las variables del algoritmo para pasar a continuación a la tercera parte.

- En la tercera parte (líneas 3-13) se configura la asignación de una nueva ruta cuando esta sea mejor que la ruta actual $T$. Así mismo, configura los condicionales que regulan un decremento en el valor de la variable $i$ así como la condición de parada del algoritmo (cuando $i$ alcanza el valor -1).

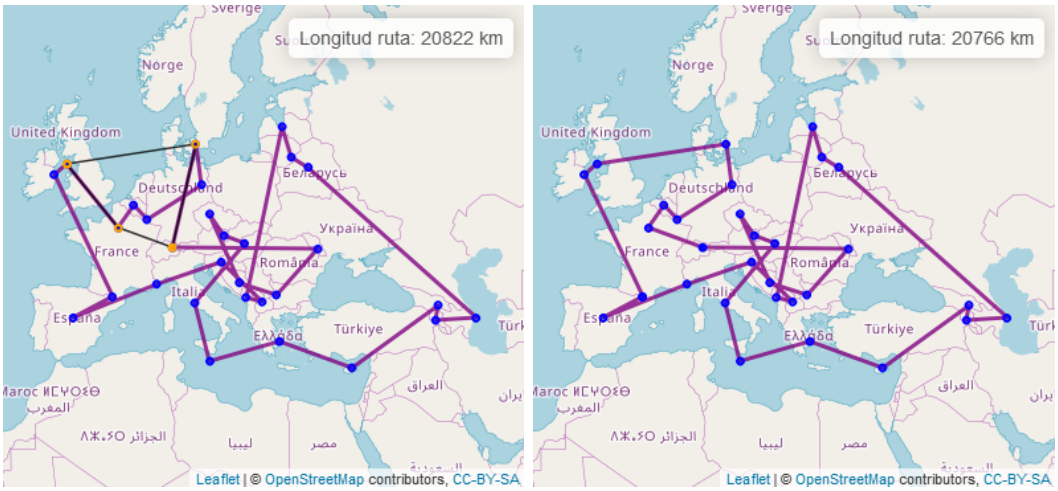
- La cuarta parte del programa (líneas 14-17) presenta la elección o no de un nuevo camino alterno siempre que este sea lo suficientemente bueno.

- La quinta parte (líneas 18-27) construye las variables $X_{i+1}$, que aglutinan un conjunto de ciudades que regulan la construcción de los caminos alternos que a cada iteración se prueban para mejorar la ruta actual.

El texto Korte y Vygen [2006] concluye destacando la efectividad del *algoritmo Lin-Kernighan*, superando con creces la del *3-Opt*. Mientras que el *algoritmo Lin-Kernighan* es al menos tan bueno como el *3-Opt*, el tiempo de ejecución esperado del mismo tomando los parámetros $p_1 = 5$ y $p_2 = 2$ también es favorable, pues reporta una complejidad en tiempo de ejecución de $\mathcal{O}(n^{2,2})$. El problema radica en que la complejidad temporal deja de ser exponencial en el peor caso posible.

La Figura 4.9 muestra una mejora efectuada por el algoritmo de Lin-Kernighan sobre el mismo supuesto de 30 ciudades ilustrado en las Figuras 4.5 y 4.6.

Casi todos los algoritmos de búsqueda local se basan en este algoritmo debido a que las soluciones obtenidas por el mismo suelen aproximarse lo suficiente a la óptima a la vez que el tiempo que requiere su ejecución es razonable en la mayoría de los casos. Otros algoritmos funcionan mejor que Lin-Kernighan en el peor caso, por ejemplo, el *algoritmo de Christofides*. Se remite al lector a Korte y Vygen [2006] para más información sobre este algoritmo.




(a) Ruta actual (b) Ruta mejorada

Figura 4.9: Ilustración de una mejora durante la ejecución del algoritmo *Lin-Kernighan* sobre una instancia con 30 ciudades. En naranja y negro, el camino alterno encontrado

### 4.5.3. Algoritmo Or-Opt

El algoritmo *Or-Opt* es uno de los algoritmos más conocidos de la clase CE (*chain exchanges*). A diferencia de los métodos *k-Opt* y *Lin-Kernighan*, que basan sus técnicas en la eliminación de aristas de la ruta que son reemplazadas por otras dando como resultado una nueva ruta, el algoritmo *Or-Opt* selecciona una cadena de vértices consecutivos de la ruta y una arista de la ruta, y construye una nueva ruta posicionando la cadena de vértices entre los dos nodos que unía la arista eliminada [Babin et al., 2007].

A lo largo de la ejecución del algoritmo, se prueba con diferentes longitudes de cadena determinadas al comienzo. Una implementación en pseudo-código del algoritmo *Or-Opt* se presenta en la Figura 3.

En Babin et al. [2007] se presenta a su vez una variante del algoritmo que intercambia los bucles. El algoritmo fija primero un vértice y sobre él va iterando con distintas longitudes de cadena hasta encontrar una arista que al introducir la cadena en la posición donde se encontraba la arista eliminada, se obtenga una nueva ruta más corta que la anterior. El pseudo-código de esta variante se presenta en el Algoritmo 4.

Estos algoritmos dotan por tanto al usuario de un mayor control en el procedimiento respecto a los algoritmos *k-Opt* o *Lin-Kernighan* vistos anteriormente, ya que es el usuario quién decide las diferentes longitudes de cadena que el algoritmo irá probando a lo largo del proceso, así como el orden en que se probarán. De esta forma, en caso de ser conocedor de la capacidad de mejora que tiene la ruta de partida, podrá elegir longitudes más grandes al principio con el fin de intentar alcanzar mejoras mayores en los primeros instantes de la ejecución, y posteriormente tamaños de cadena más pequeños para refinar la ruta

**Algoritmo 3** Algoritmo Or-Opt

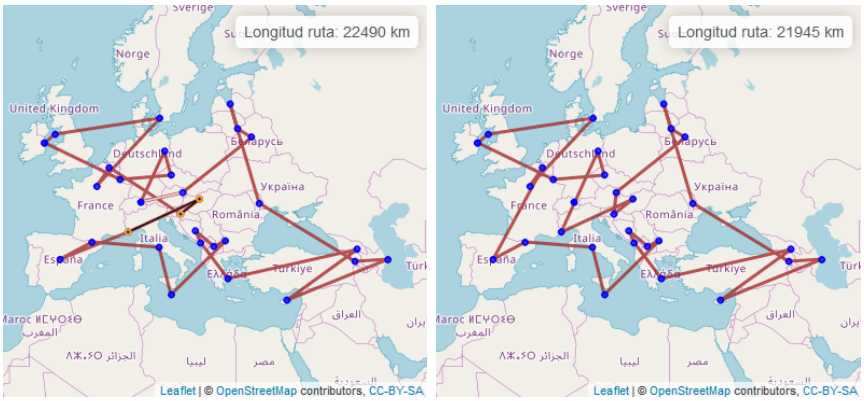
**Entrada:** Una instancia $\mathcal{I} = (G, C)$, con $G = (V, E)$ del TSP. Parámetros: longitudes de cadena $l_1, l_2, \ldots, l_s \in \mathbb{N}$
**Salida:** Una ruta $R$

Sea $R$ una ruta
**para** $l_t$ **hacer**
  **para** $v_k \in V(R)$ **hacer**
    **para** $e \in E(R)$ **hacer**
      Asignar a $R'$ la ruta resultante de eliminar $e$ y reconectar en su lugar la cadena de vértices de $R$ que conforman $v_k$ y los $(l_t - 1)$ siguientes.
      **si** $\sum_{(v_i,v_j)\in E(R')} c_{i,j} < \sum_{(v_i,v_j)\in E(R)} c_{i,j}$ **entonces**
        $R := R'$
        Salir del bucle sobre $v_k$
      **fin si**
    **fin para**
  **fin para**
**fin para**
**devolver** $R$

**Algoritmo 4** Variante algoritmo Or-Opt: *vertex first*

**Entrada:** Una instancia $\mathcal{I} = (G, C)$, con $G = (V, E)$ del TSP. Parámetros: longitudes de cadena $l_1, l_2, \ldots, l_s \in \mathbb{N}$
**Salida:** Una ruta $R$

Sea $R$ una ruta
**para** $v_k \in V(R)$ **hacer**
  **para** $l_t$ **hacer**
    **para** $e \in E(R)$ **hacer**
      Asignar a $R'$ la ruta resultante de eliminar $e$ y reconectar en su lugar la cadena de vértices de $R$ que conforman $v_k$ y los $(l_t - 1)$ siguientes.
      **si** $\sum_{(v_i,v_j)\in E(R')} c_{i,j} < \sum_{(v_i,v_j)\in E(R)} c_{i,j}$ **entonces**
        $R := R'$
        Salir del bucle sobre $l_t$
      **fin si**
    **fin para**
  **fin para**
**fin para**
**devolver** $R$




(a) Ruta actual (b) Ruta mejorada

Figura 4.10: Ilustración de una mejora con longitud de cadena 3 durante la ejecución del algoritmo *Or-Opt* (ver Algoritmo 3) sobre una instancia con 30 ciudades. En negro y naranja la cadena de vértices, y en blanco la arista que se sustituye por la cadena

resultante e intentar aproximarla a la solución óptima. Del análisis de ambos códigos se obtiene un coste computacional asociado de orden $\mathcal{O}(n^2)$, siendo $n$ el número de nodos de la instancia.

Entre las ventajas del *Or-Opt* está su gran flexibilidad, capaz de resolver rápidamente instancias cuya ruta de partida difiere mucho de la solución óptima, así como su reducida complejidad computacional siendo recomendable su aplicación sobre instancias grandes. Como principal desventaja se encuentra la necesaria elección correcta de las longitudes de cadena, pues en caso contrario el algoritmo puede ver altamente reducidas sus prestaciones.

La Figura 4.10 muestra una mejora efectuada por el algoritmo *Or-Opt*. En ella se ilustra en negro (aristas) y naranja (vértices) la cadena de vértices que se escinde para colocarla en la posición en la que situaba la arista señalada en color blanco. Para hacer esto posible desaparecen de la ruta las dos aristas que unían la cadena con los vértices inmediatamente anterior y posterior.

Para concluir, la Figura 4.12 ilustra las rutas finales que obtienen los cuatro algoritmos expuestos a lo largo de esta sección al ser ejecutados sobre el mismo supuesto con 30 ciudades y comenzando en la misma ruta de partida (ilustrada en la Figura 4.11). Obsérvese que, en línea con lo ya mencionado, la ruta final del algoritmo *3-Opt* se entrecruza consigo misma en la zona de Bélgica, indicando su elevada longitud. Sin embargo el algoritmo no es capaz de mejorarla ya que para romper el entrecruzamiento es necesario eliminar las dos aristas que se cortan entre sí, manteniendo las demás. Esto no es posible con el *3-Opt* ya que siempre busca rutas que se obtengan a partir de la eliminación de exactamente 3 aristas de la ruta actual. Esto vemos que no ocurre con el *2-Opt*, pues sus modificaciones (denominadas *2-changes*) se conciben precisamente para eliminar los entrecruzamientos de la ruta.

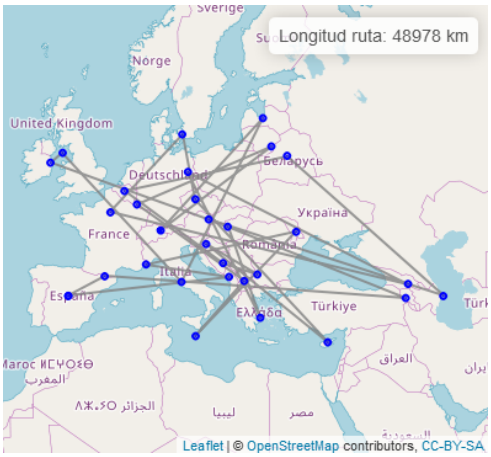
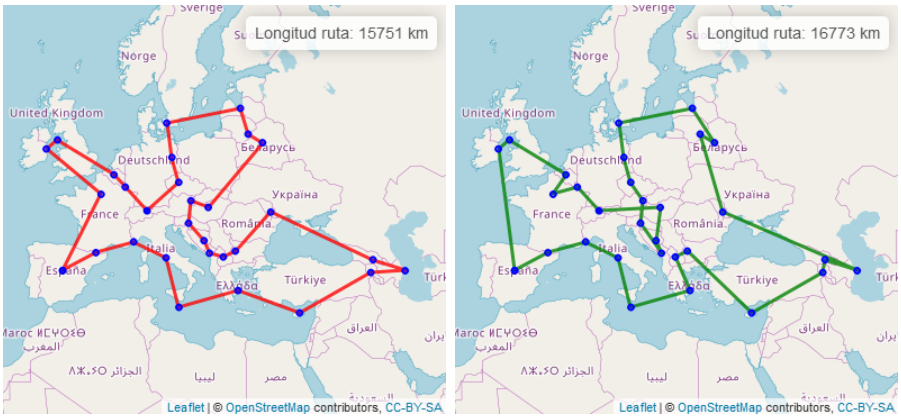
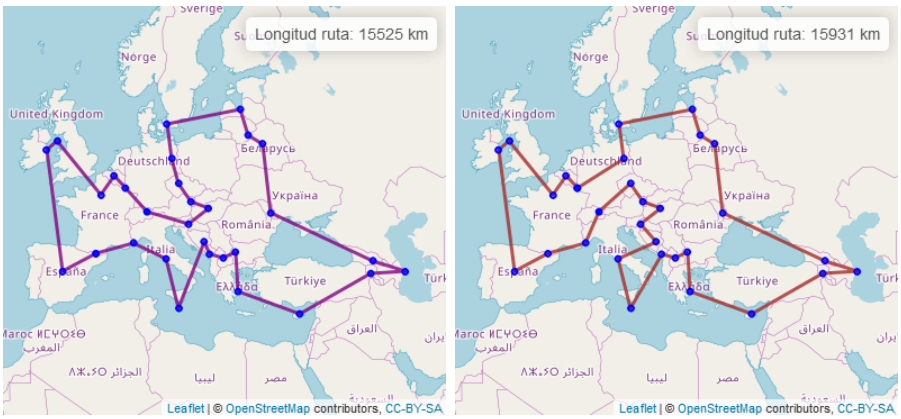



Figura 4.11: Ruta de partida para la ejecución de los algoritmos de búsqueda local expuestos en esta sección, cuyos resultados finales se ilustran en la Figura 4.12




(a) *2-Opt* (b) *3-Opt*




(c) *Lin-Kernighan*, $p_1 = 5$, $p_2 = 2$ (d) *Or-Opt*, $longitudes = (12, 6, 3, 2, 2, 1, 1, 1)$

Figura 4.12: Rutas finales obtenidas de la ejecución de los cuatro algoritmos ilustrados sobre una misma instancia con 30 ciudades y con la misma ruta de partida, que muestra la Figura 4.11

# Capítulo 5

# Estudio preliminar de los algoritmos

Hasta ahora se han recogido los fundamentos del problema del viajante o *Traveling Salesman Problem* (TSP), así como los algoritmos más destacados que abordan el problema, a modo de estado del arte del trabajo. Ahora, es objeto del presente capítulo focalizar en el comportamiento de los algoritmos anteriores al ser aplicados a la reordenación de trayectorias aéreas, llevando a cabo un estudio comparativo que permita obtener unas primeras conclusiones sobre su adecuació en este contexto.

Por tanto, para esta fase se fijan los siguientes objetivos:

- Aplicación de los algoritmos a la reordenación de colecciones de trayectorias concretas de diferentes características.
- Establecimiento de una comparativa entre los resultados obtenidos por los diferentes algoritmos, tras ser ejecutados sobre las mismas trayectorias en las mismas condiciones (mismo ordenador, misma carga del procesador, etc).
- Desarrollo de una aplicación con interfaz gráfica que permita visualizar la reconstrucción de las rutas sobre un mapa terrestre, así como otras gráficas que aporten al estudio información complementaria.
- Analizar las formas más adecuadas de llevar a cabo la corrección de los *timestamps* de los mensajes ADS-B recibidos, de manera que el nuevo orden de éstos sea coherente con el orden real en que fueron enviados por la aeronave.

Primeramente, se fijó el lenguaje de programación que podría resultar más adecuado para llevar a cabo el estudio preliminar fijado. Para este caso, se estableció que `R` era la mejor opción por su amplia colección de librerías que incluyen la implementación de múltiples algoritmos para resolver distintos problemas complejos, incluido el TSP.

## 5.1. Introducción a R

El lenguaje R es un conjunto integrado de programas orientados a la manipulación de datos, cálculo y gráficos. En González y González [2000] se identifican cinco características principales de este lenguaje:

- Almacenamiento y manipulación efectiva de datos
- Operadores para cálculo sobre variables indexadas (Array), en particular matrices
- Una amplia, coherente e integrada colección de herramientas para análisis de datos
- Posibilidades gráficas para análisis de datos
- Un lenguaje de programación simple y efectivo que incluye condicionales, ciclos, funciones recursivas y posibilidad de entradas y salidas.

Surge a modo de una nueva implementación del lenguaje S, desarrollado en AT&T por Rick Becker, John Chambers y Allan Wilks.

Aunque R no es un lenguaje con aplicación exclusiva al campo de la estadística, es ampliamente utilizado en esta materia. Algunas de estas funcionalidades se encuentran incluidas en el entorno base de R y otras se acompañan en forma de *bibliotecas* (*packages*). Existen ocho bibliotecas incluidas en R, y que reciben el nombre de bibliotecas estándar. Otras muchas, se encuentran disponibles en CRAN a través de Internet.

El lenguaje R es *orientado a objetos*, interpretado y no compilado [Ahumada, 2002], por lo que los comandos escritos se ejecutan directamente sin necesidad de la construcción de ejecutables que los incluyan. Destaca por la simplicidad de su sintaxis, que permite la ejecución de funciones complejas como regresiones lineales a través de comandos sencillos.

El amplio conjunto de procedimientos que pueden llevarse a cabo con R incluyen cálculos sencillos, operaciones vectoriales, generación de sucesiones, gestión de distintas clases de objetos, trabajo con matrices, listas y *dataframes*, lectura y escritura de archivos con diferentes formatos, así como una gran variedad de cálculos estadísticos.

## 5.2. Selección de librerías

Tras la elección del lenguaje R como medio para desarrollar la aplicación objeto de esta fase del trabajo, se realiza una búsqueda de librerías que incluyan implementaciones de las heurísticas de mejora local más destacadas, así como otros algoritmos aplicables a la resolución del problema del viajante. Así, se descarta la posibilidad de implementar los algoritmos desde cero en esta fase, pues supone un elevado esfuerzo y hay muchas posibilidades de que el algoritmo implementado no satisfaga las expectativas.

En esta búsqueda exhaustiva de librerías se identifican las siguientes:

- `TSP`: la librería `TSP`, que incluye implementaciones de un gran conjunto de algoritmos destinada a la resolución de instancias del problema del viajante. Estos algoritmos son: *2-Opt*, *vecinos más próximos*, *vecinos más próximos repetitivo*, *inserción más cercana*, *inserción más barata*, *inserción más lejana*, *inserción arbitraria*, *Lin-Kernighan* y *Concorde*. La descripción de todos estos algoritmos se incluye en la Sección 4.4.
- `stats`: la librería `stats`, de propósito más general, es un optimizador orientado a encontrar el valor mínimo de una función objetivo de un problema, recorriendo el espacio de estados del mismo. Para ello, esta librería recoge una serie de algoritmos de búsqueda informada, entre los que encontramos el *Simulated Annealing*.

A continuación, se requiere encontrar un segundo conjunto de librerías que permitan la elaboración de la interfaz de usuario.

- `Shiny`: sostiene el desarrollo de aplicaciones sobre `R`, proporcionando una API que permite el desarrollo del código tanto del lado cliente como del lado servidor.
- `ShinyDashboards`: contiene un conjunto de métodos y funciones que permiten la creación de *dashboards* y cuadros de mando.
- `Leaflet`: permite el renderizado de mapas, así como una serie de funciones para representar distintos elementos sobre éste como puntos, líneas, marcadores, etc. Estos elementos nos proporcionan las herramientas necesarias para poder representar los distintos mensajes ADS-B asociados a una determinada trayectoria, así como unirlos a través de una línea para representar el orden de éstos según su *timestamp*.
- `Plotly`: librería para la elaboración de gráficos estadísticos de diferente tipología.

## 5.3. Análisis

A continuación se incluye el análisis asociado al desarrollo de la aplicación. Una vez queda patente la viabilidad de la aplicación, se identifican las características básicas de la aplicación. Estas son las siguientes:

CAR-1 **Exploración de los datos**: contempla la creación de un *dashboard* sobre el que puedan visualizarse los datos de los vuelos sobre diferentes formatos (gráficas, mapas, tablas, etc), así como los resultados de ejecutar reordenaciones.

CAR-1.1 **Ejecutar una reordenación**: el sistema debe permitir al usuario ejecutar reordenaciones sobre el vuelo, de entre los almacenados en el sistema, y con el algoritmo, de entre los listados en la Sección 5.2, que elija el usuario.

CAR-1.2 **Interacción *dashboard***: engloba aquellas funcionalidades que permiten al usuario manipular el *dashboard*.

CAR-1.2.1 **Seleccionar vuelo**: el sistema debe permitir al usuario seleccionar el vuelo, de entre los almacenados en el sistema, que desea visualizar en el *dashboard*.

CAR-1.2.2 **Acotar rango de puntos visualizados**: el sistema permitirá al usuario trabajar con un subconjunto de los datos asociados a un vuelo.

CAR-1.2.3 **Filtrar fuentes de datos**: el usuario permitirá al usuario representar y eliminar del *dashboard* los datos pertenecientes a las fuentes de datos que el usuario elija, así como visualizar la fuente de la que procede cada dato representado.

CAR-1.2.4 **Visualizar datos mensaje ADS-B**: el sistema mostrará al usuario todos los datos que almacene de cada mensaje ADS-B en el sistema, así como filtrar los mensajes de acuerdo a un patrón introducido por el usuario.

CAR-2 **Pruebas**: contempla la realización de baterías de pruebas en las que se ejecuten varias reordenaciones sobre varios vuelos y usando diferentes algoritmos.

CAR-2.1 **Ejecutar una batería de pruebas**: el sistema permitirá al usuario ejecutar varias reordenaciones sobre los vuelos que seleccione y los algoritmos elegidos.

CAR-2.2 **Descargar resultados**: los datos presentados al finalizar una batería de pruebas serán descargables con el fin de poder ser posteriormente conservados.

CAR-3 **Corrección de timestamps**: la aplicación incorporará también un modelo de corrección de los timestamps tras una reordenación, a fin de que la marca temporal de cada mensaje se asemeje lo más fielmente posible al instante temporal en que la aeronave envió verdaderamente el mensaje.

CAR-3.1 **Visualizar relación distancia recorrida y nuevo timestamp con respecto a la altura**: el usuario podrá observar esta relación tras la ejecución de una reordenación [CAR-1.1] con el fin de comprobar cómo se ha llevado a cabo la corrección de los *timestamps* de los mensajes.

De la lista anterior de características, los requisitos de usuario serán aquellas que no tengan subcaracterísticas por debajo de ellas. De estos podemos derivar los distintos requisitos funcionales, que describen la totalidad de la funcionalidad a cubrir. Los requisitos funcionales identificados son los siguientes:

RF-01 El sistema ejecutará una reordenación sobre los datos del vuelo que el usuario haya elegido, así como haciendo uso del algoritmo y parámetros que el usuario haya introducido [CAR-1.1].

RF-02 El sistema mostrará un mensaje de error si, al ordenarse una reordenación, alguno de los parámetros introducidos no es válido [CAR-1.1, CAR-2.1].

RF-03 El sistema repintará el *dashboard* tras finalizar una reordenación para mostrar por pantalla la información asociada a la misma (distancia de la nueva ruta encontrada,

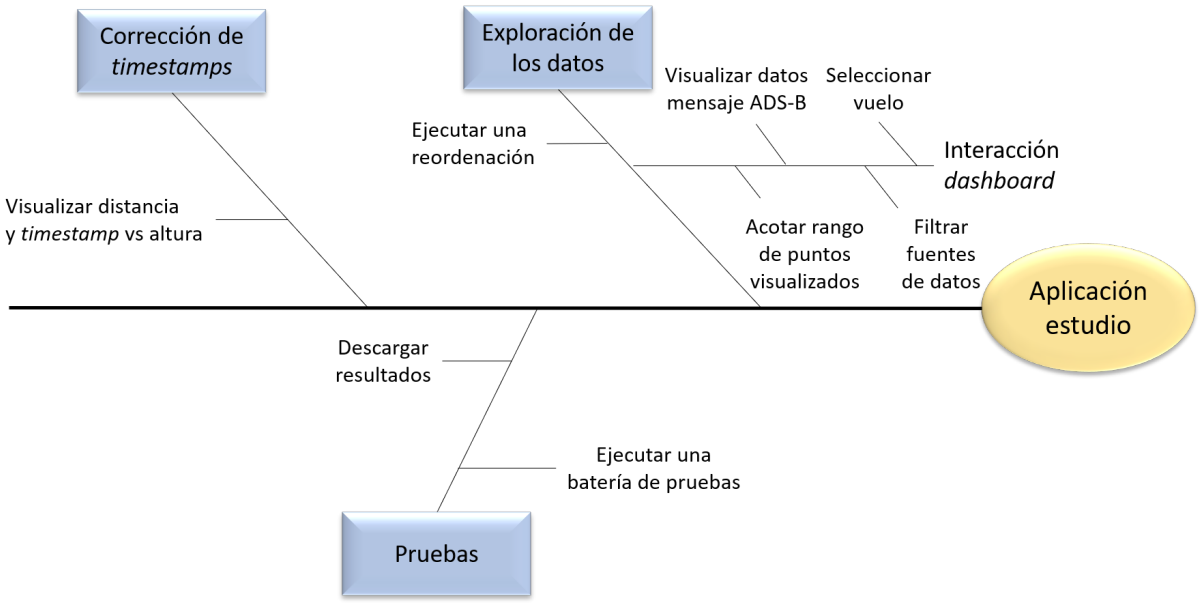



Figura 5.1: Arbol de características de la aplicación en R

puntos cuyo orden haya sido modificado, etc) [CAR-1.1].

RF-04 El sistema leerá todos los ficheros *csv* de la carpeta *Vuelos* del proyecto y permitirá al usuario elegir cual de ellos desea que se represente en el *dashboard* [CAR-1.2.1].

RF-05 El sistema cargará un vuelo en el *dashboard* tras haber sido seleccionado por el usuario [CAR-1.2.1].

RF-06 El sistema cargará de nuevo el *dashboard* cada vez que el usuario ordene modificar el rango de puntos a representar [CAR-1.2.2].

RF-07 El sistema hará visible sobre el *dashboard* la fuente de datos de la que procede cada mensaje ADS-B [CAR-1.2.3].

RF-08 El sistema presentará sobre el *dashboard* aquellos mensajes ADS-B procedentes de fuentes de datos seleccionadas por el usuario [CAR-1.2.3].

RF-09 El sistema presentará todos los datos asociados a cada mensaje ADS-B de un vuelo elegido [CAR-1.2.4].

RF-10 El sistema mostrará al usuario todos los registros en los que alguno de sus datos se corresponda con el patrón por el que el usuario haya ordenado filtrar [CAR-1.2.4].

RF-11 El sistema permitirá al usuario ordenar la ejecución de una batería de pruebas.

RF-12 El sistema presentará tres modos de ejecución de una batería de pruebas: sin ventanas, utilizando un solo tipo de ventana o utilizando dos tipos de ventana para adaptar aeropuertos (ver Subsección 5.4.1) [CAR-2.1].

RF-13 El sistema permitirá al usuario elegir varios vuelos de entre aquellos cuyos datos se encuentren en la carpeta *Vuelos* en formato *csv* [CAR-2.1].

RF-14 El sistema permitirá al usuario elegir varios algoritmos de entre aquellos citados en la Sección 5.2 [CAR-2.1].

RF-15 El sistema mostrará los datos de las ejecuciones en base a métricas de rendimiento, así como los datos individuales de cada ejecución, que incluyan la longitud de la nueva ruta encontrada y la de partida, el tiempo de ejecución y el porcentaje de mejora que se obtiene sobre la ruta de partida en términos de distancia [CAR-2.1].

RF-16 El sistema permitirá al usuario la descarga de los datos resultantes de la ejecución de una batería de pruebas, entre los que se incluyen las longitudes de las rutas de partida y final, el tiempo de ejecución, y el porcentaje de mejora que se obtiene sobre la ruta de partida en términos de distancia [CAR-2.2].

RF-17 El sistema mostrará por pantalla la información que permita al usuario establecer comparación entre la altura de la aeronave y la distancia recorrida desde el comienzo con respecto al tiempo [CAR-3.1].

## 5.4. Desarrollo de la aplicación

En esta sección se describen los aspectos fundamentales que caracterizan el desarrollo llevado a cabo de la aplicación objeto de este capítulo.

Esta aplicación se caracteriza por la presencia importante de una interfaz gráfica elaborada, que responde a la necesidad de visualizar y obtener las conclusiones arrojadas por el estudio que se pretende llevar a cabo en esta fase del proyecto. Estas conclusiones deben ser claras y precisas de tal manera que sea posible determinar lo más fielmente posible el comportamiento de los distintos algoritmos en estudio, identificando las fortalezas y debilidades de cada uno de ellos. Es por esto, que se opta por la realización de un *dashboard*, que aloje al mismo tiempo una serie de gráficas y mapas ilustrativos, claves para la obtención de conclusiones. Tal y como se ha mencionado previamente, la interfaz de usuario de la aplicación hace uso de la librería `shiny` de `R`, que permite la elaboración de *dashboards* y aplicaciones de tipo cliente-servidor, proporcionando funciones que permiten el desarrollo de ambas partes. De esta forma, el código de la aplicación queda dividido en dos partes, una parte cliente, que define todos los componentes que forman parte de la interfaz de usuario, así como su distribución, y una parte servidor, que implementa toda la lógica de negocio, así como el acceso a los datos.

La parte cliente queda configurada a través de la función `shinyUI()`, que conforma en su totalidad la capa de presentación. A través de esta función se determinan y distribuyen los distintos componentes que se muestran en pantalla en cada situación. Por defecto, `Shiny` divide la interfaz de usuario en tres componentes: una cabecera o *hea-*

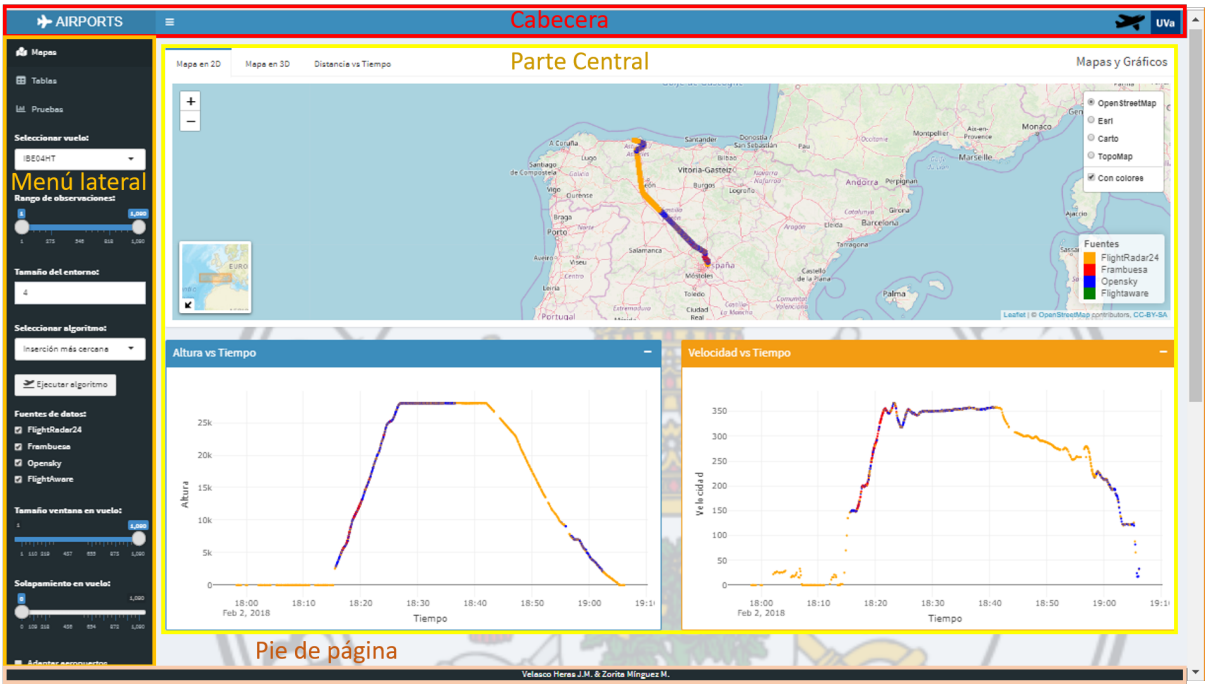



Figura 5.2: Partes de la interfaz de la aplicación

*der*, configurado mediante la función `dashboardHeader()`, una barra lateral, que puede albergar diferentes opciones a modo de menú, así como botones, sliders u otros componentes; mediante `dashboardSidebar()`, y una parte central que contenga los diferentes elementos que componen el *dashboard*, con `dashboardBody()`. A su vez, es posible añadir componentes adicionales a la estructura, como un pie de página.

La Figura 5.2 representa la página de inicio de la aplicación. La barra lateral representa un menú de configuración de los componentes presentes en la parte central. La parte superior de este menú lateral presenta tres *widgets* que permiten al usuario cambiar la vista de la parte central.

- El primero de ellos, denotado como *mapas*, presenta la vista principal (ilustrada en la Figura 5.3), formada por los componentes que componen el *dashboard*. En la parte superior se encuentra un panel con tres pestañas que permite al usuario visualizar tres componentes distintos. El primero de ellos, y visualizado en la Figura 5.3, es el mapa terrestre sobre el cual se pinta la posición de los mensajes ADS-B asociados a un vuelo concreto. El segundo de estos componentes es una gráfica en tres dimensiones que se hacía referencia anteriormente. La tercera es la gráfica que muestra la corrección del *timestamp* asociado a los mensajes ADS-B tras haber ejecutado una reordenación. En la parte inferior del *dashboard* se presentan las gráficas de altura frente al *timestamp* así como de velocidad frente a *timestamp*.
- El segundo de los *widgets* es el que presenta en la parte central una tabla con los datos de mensajes ADS-B asociados a una trayectoria.

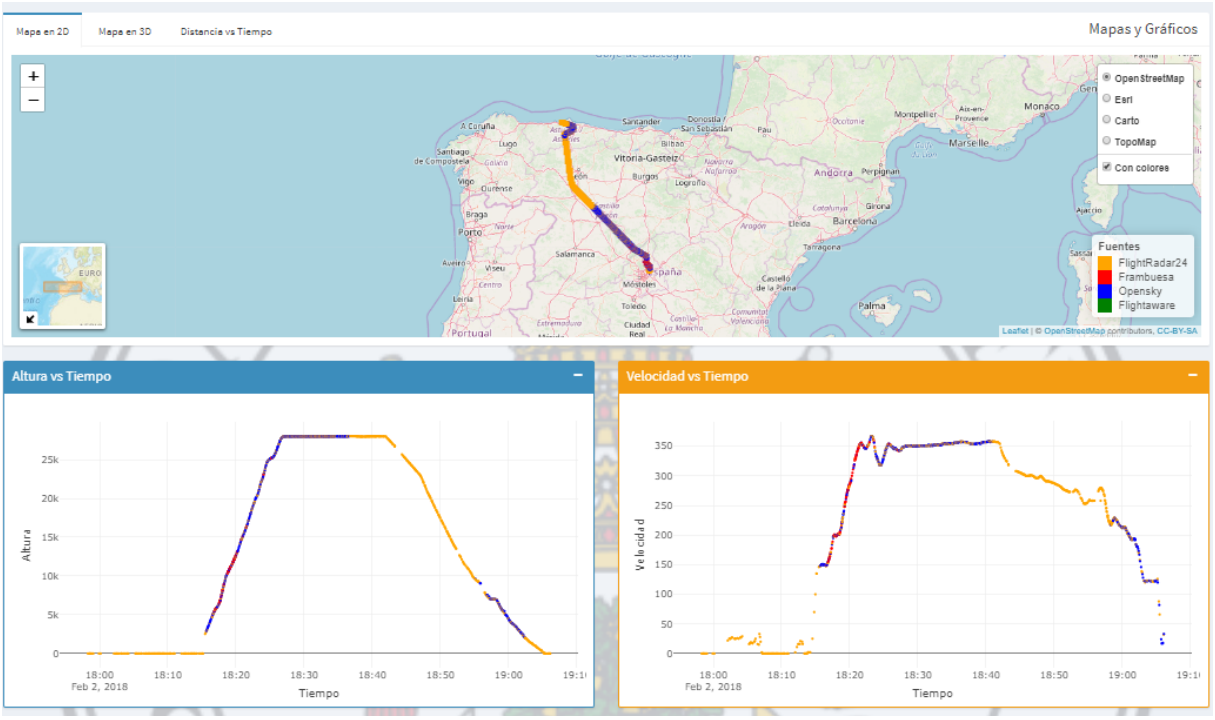



Figura 5.3: Widget mapas seleccionado

Show 10 entries Search:

Tabla de datos del vuelo IBE04HT

| | id | timestamp | latitude | longitude | altitude | speed | vspeed | squawk | track | ground | leg | date |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | fr24-34250C-1517590697 | 1517590697 | 40.48716 | -3.59244 | 0 | 0 | 0 | 0000 | 304 | true | IBE04HT_34250C_1517590697_1517594767 | 2018-02-02 |
| 2 | fr24-34250C-1517590708 | 1517590708 | 40.48718 | -3.59243 | 0 | 0 | 0 | 0000 | 13 | true | IBE04HT_34250C_1517590697_1517594767 | 2018-02-02 |
| 3 | fr24-34250C-1517590712 | 1517590712 | 40.48716 | -3.59243 | 0 | 0 | 0 | 0000 | 194 | true | IBE04HT_34250C_1517590697_1517594767 | 2018-02-02 |
| 4 | fr24-34250C-1517590717 | 1517590717 | 40.48717 | -3.59243 | 0 | 0 | 0 | 0000 | 154 | true | IBE04HT_34250C_1517590697_1517594767 | 2018-02-02 |
| 5 | fr24-34250C-1517590723 | 1517590723 | 40.48716 | -3.59243 | 0 | 0 | 0 | 0000 | 335 | true | IBE04HT_34250C_1517590697_1517594767 | 2018-02-02 |
| 6 | fr24-34250C-1517590733 | 1517590733 | 40.48717 | -3.59243 | 0 | 0 | 0 | 0000 | 154 | true | IBE04HT_34250C_1517590697_1517594767 | 2018-02-02 |
| 7 | fr24-34250C-1517590744 | 1517590744 | 40.48716 | -3.59244 | 0 | 0 | 0 | 0000 | 154 | true | IBE04HT_34250C_1517590697_1517594767 | 2018-02-02 |
| 8 | fr24-34250C-1517590794 | 1517590794 | 40.48716 | -3.59243 | 0 | 0 | 0 | 0000 | 335 | true | IBE04HT_34250C_1517590697_1517594767 | 2018-02-02 |
| 9 | fr24-34250C-1517590809 | 1517590809 | 40.48716 | -3.59244 | 0 | 0 | 0 | 0000 | 335 | true | IBE04HT_34250C_1517590697_1517594767 | 2018-02-02 |
| 10 | fr24-34250C-1517590927 | 1517590927 | 40.48552 | -3.59122 | 0 | 22 | 0 | 0000 | 87 | true | IBE04HT_34250C_1517590697_1517594767 | 2018-02-02 |

Showing 1 to 10 of 1,090 entries Previous 1 2 3 4 5 … 109 Next

Figura 5.4: Widget tabla seleccionado

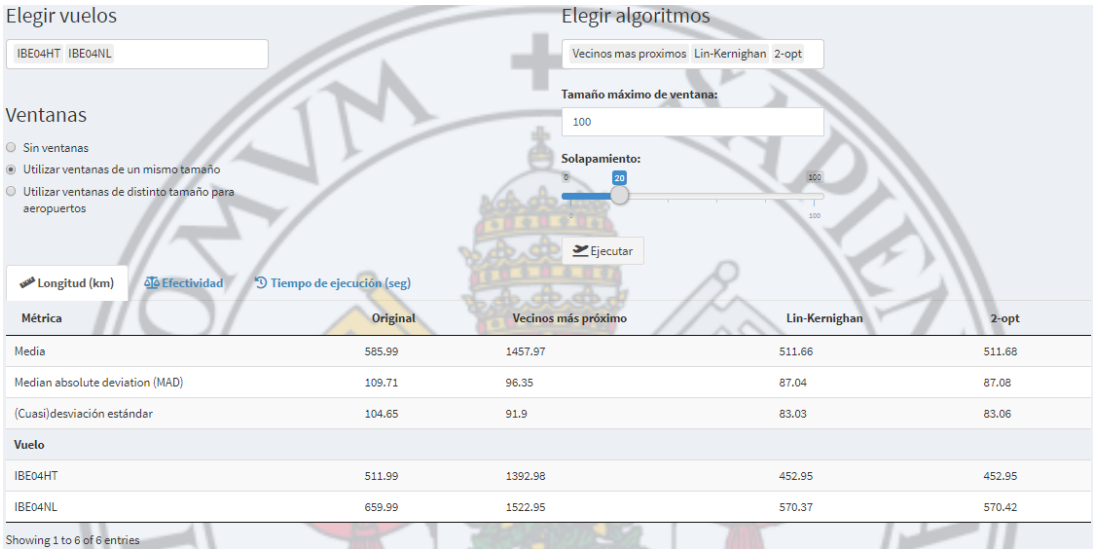



Figura 5.5: Widget pruebas seleccionado

- El último *widget* presenta en la parte central un formulario de configuración para ejecutar baterías de pruebas [CAR-2]. El usuario podrá seleccionar un conjunto de vuelos y un conjunto de algoritmos de tal manera que se aplicará cada algoritmo para reordenar cada uno de los vuelos elegidos. Los resultados de la ejecución se presentan en un conjunto de tablas que indican la longitud de la ruta encontrada, la mejora que representa en relación con la ruta original y el tiempo requerido en la ejecución de cada algoritmo.

La parte de servidor contiene toda la lógica de negocio así como el acceso a los datos. En el caso del segundo, no es objeto de esta fase la implementación de un auténtico acceso a base de datos, sino la simple carga de un conjunto acotado de trayectorias contenidas cada una en ficheros de extensión *.csv*. De este modo, el servidor lee y procesa el fichero asociado al vuelo elegido por el usuario. A lo largo de la ejecución del programa, el usuario puede cambiar el vuelo elegido haciendo uso del selector situado en la barra lateral. En este proceso, el servidor accederá al fichero asociado con el vuelo seleccionado para interpretarlo y generar de nuevo los componentes del *dashboard*.

En cuanto a la lógica de negocio, destaca la inclusión de los algoritmos en estudio, cuya implementación viene dada por las librerías `stats` y `TSP`. El modelo de reordenación implementado consiste en considerar al conjunto de los mensajes ADS-B de la trayectoria seleccionada como ciudades de una instancia del problema del TSP, con la diferencia de que en este caso el camino a encontrar tiene que ser abierto en lugar de cerrado. Para solventar esta pequeña diferencia se añade una ciudad artificial que se una a todas las demás con coste (o longitud) 0, de manera que no afecte a la solución final de la reordenación, que se obtiene eliminando esta ciudad artificial que separará al inicio y al final de la ruta.

A su vez, se programan varios modos de ejecución de las reordenaciones sobre los mensajes ADS-B de una trayectoria, de manera que el usuario pueda elegir entre

- ejecutar el algoritmo de resolución del TSP elegido sobre el conjunto total de mensajes del vuelo, considerando como ciudades del supuesto del TSP a resolver como ciudades o
- dividir el problema en varios supuestos más pequeños que se resuelven paso a paso.

En esta última, se fija un tamaño máximo de los supuestos a resolver (número de mensajes máximo de los subproblemas) y un valor de solapamiento (número de mensajes que coinciden entre un supuesto y el siguiente). Tras resolver uno a uno todos los subproblemas se obtienen rutas para cada uno de ellos que son combinadas entre si para conformar la ruta solución del vuelo que resuelven. Estos subproblemas reciben el nombre de *ventanas de ejecución* al igual que la presente técnica. La Subsección 5.4.1 detalla los fundamentos matemáticos de este procedimiento.

Cabe destacar a su vez, que la aplicación incluye dos submodos de ejecución basados en esta técnica de ventanas de ejecución. La primera consiste en fijar un único tamaño máximo para las ventanas de ejecución mientras que la segunda permite fijar dos tamaños diferentes de ventana. Esta última consiste en dividir la trayectoria en tres partes de manera que uno de los tamaños fijados se aplique para los puntos situados en la primera y tercera parte de la ruta mientras que el segundo de los tamaños se aplique en la parte central del vuelo. Esto está pensado para obtener una mayor adaptabilidad de la técnica a las diferentes partes del vuelo. Así, los mensajes ADS-B que la aeronave emitió antes del despegue así como tras el aterrizaje tendrán datos de posición más cercanos entre sí que cuando la aeronave se encuentra en vuelo debido a la variación de la velocidad. Además, la trayectoria de la aeronave se vuelve más impredecible en las inmediaciones de los aeropuertos, pues es probable que el piloto requiera de hacer maniobras aéreas que involucren, por ejemplo, pasar por la misma zona repetidas veces. Es por ello que es conveniente la aplicación de ventanas de tamaño más reducido en las inmediaciones de los aeropuertos con respecto a la parte central, donde la trayectoria suele ser rectilínea sin apenas cambios de dirección por parte del piloto.

La parte de servidor incluye también un conjunto de *event handlers* o manejadores de eventos asociados a la interacción del usuario con cada uno de los componentes de la interfaz. Entre ellos está el de cambiar el mapa y los gráficos al modificar el vuelo en el selector o el que ejecuta el algoritmo elegido tras hacer click en el botón *Ejecutar*.

### 5.4.1. El modelo de reordenación de trayectorias: la técnica de las ventanas de ejecución

La aplicación de los algoritmos a las trayectorias tiene como principal objetivo obtener de forma rápida la ordenación ascendente de los mensajes ADS-B en función del instante

temporal en que fueron enviados. Para conseguir esto, debe tenerse en cuenta que el TSP es un problema $\mathcal{NP} - hard$, y de la misma forma, los principales algoritmos que atacan el problema pueden alcanzar elevados tiempos de ejecución debido a la alta complejidad del problema. Es por ello que se prefiere la resolución de instancias pequeñas del problema frente a las más grandes en términos de proximidad a la solución óptima y tiempo de ejecución.

Sin embargo, aquellas trayectorias con un elevado número de puntos necesariamente requerirán un alto coste computacional para ser reordenadas. Pensemos en un vuelo que atraviesa Europa desde Madrid hasta Moscú arrojando 2 mensajes ADS-B por segundo. Al final del trayecto, la gran cantidad de datos de seguimiento acumulados hará más difícil su procesamiento, lo cual puede llegar a hacer imposible la ordenación de esta trayectoria en un tiempo razonable, más aun considerando la alta carga que enfrentan por lo general los sistemas de gestión del tráfico aéreo (ATM). Es por eso que se busca un método más eficiente de tratar las instancias más complejas del problema.

En esta línea, se diseña una técnica de ejecución del tipo *divide y vencerás*. Este método consistirá en descomponer cada instancia en instancias más pequeñas del problema con un número máximo de nodos cada una (mensajes ADS-B). La primera fase del procedimiento (dividir) viene dada de la forma siguiente.

Fijados $\tau > 2$ y $\sigma$ tal que $0 \leq \sigma < \tau$ y dada una instancia del problema de reordenación de trayectorias con un grafo $G = (V, E)$ de $n$ vértices asociado, con $V = \{v_1, v_2, \ldots, v_n\}$, se definen los subconjuntos de vértices

$$V_k = \{v_{i_k}, v_{i_k+1}, \ldots, v_{j_k}\}, \text{ con } j_k = \text{máx}\{i_k + \tau, n\},\ i_k = j_{k-1},\ i_1 = 1$$

para todo $k \in \mathbb{N}$. Se construyen entonces nuevas instancias con grafos

$$G_k = (V_k, E_k),\ k \in \mathbb{N}, \quad \text{donde} \quad E_k = \{(v_a, v_b) : v_a, v_b \in V_k\}.$$

De esta forma, los pares $(G_k, C_k)$ para cada $k \in \mathbb{N}$ forman nuevas instancias del problema de reordenación de trayectorias con un máximo de $\tau + 1$ puntos cada una. Al parámetro $\tau$ lo denotaremos por *tamaño de la ventana* mientras que a $\sigma$ lo haremos por *solapamiento*.

De esta forma, se han obtenido subinstancias tratables del problema de reordenación de trayectorias que ahora pueden resolverse (vencer) mediante un algoritmo de resolución del problema del viajante (TSP). Obtenemos así un conjunto de soluciones para las subinstancias anteriores, a partir de las cuales podemos derivar una solución para la instancia de partida. Obsérvese que, debido al solapamiento, cada vértice del problema original puede encontrarse en varias subinstancias, de forma que, igualmente, cada arista puede aparecer en varias de las soluciones obtenidas para cada instancia, estando por ejemplo antes que otra arista en la solución a una instancia, y después que ella en otra. En este caso, se tiene en cuenta el ordenamiento que aparece en la solución de la instancia posterior, es decir, sea $m = \text{máx}\{l : \cap_{i=1}^{l} E_i \neq \varnothing\}$, aparecerán en la solución aquellas aristas de $\cap_{i=1}^{m} E_i$ que aparezcan en la solución de la subinstancia con grafo $G_m$. Este hecho se ilustra en la Figura 5.6.

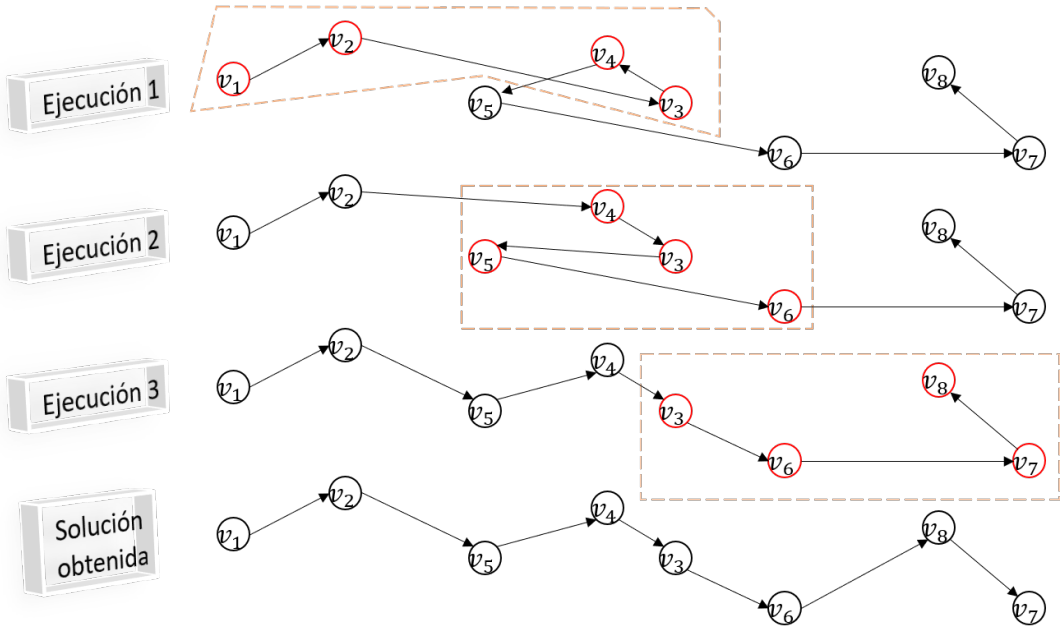



Figura 5.6: Ejemplo de reordenación de una trayectoria a través de su descomposición, con $\tau = 4$ y $\sigma = 2$

| **Algoritmo de resolución** | **IBE04HT** | | **IBE04NL** | | **IBE0519** | | **IBE05DK** | | **RYR9KY_4CA97C** | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Long | Tmp | Long | Tmp | Long | Tmp | Long | Tmp | Long | Tmp |
| Sin aplicar algoritmo | 511,99 | - | 659,99 | - | 682,84 | - | 602,47 | - | 346,98 | - |
| 2-Opt, sin ventanas | 452,95 | 0,42 | 570,42 | 0,86 | 600,38 | 0,66 | 558,69 | 0,36 | 299,99 | 0,04 |
| 2-Opt, $\tau = 100, \sigma = 20$ | 452,95 | 0,04 | 570,42 | 0,00 | 600,38 | 0,00 | 558,69 | 0,00 | 299,99 | 0,00 |

Tabla 5.1: Comparativa de la técnica de las ventanas de ejecución con respecto a su no utilización en longitud de la solución obtenida (Long) en kilómetros y tiempo de ejecución (Tmp) en segundos

La técnica de resolución de instancias del problema del viajante descrita en este epígrafe, se programa en la parte de servidor de la aplicación en R con el fin de estudiar su comportamiento y comprobar si de esta forma es posible reducir notablemente el tiempo de ejecución de los algoritmos sin empeorar la calidad de las rutas encontradas a través de este procedimiento.

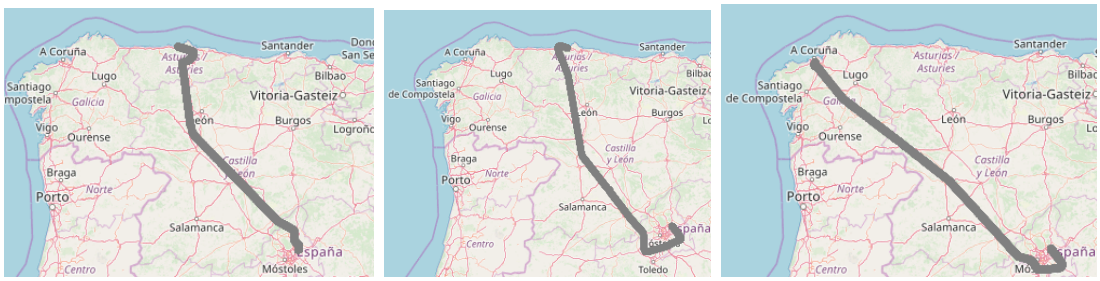
A favor de esta técnica cabe destacar que resulta novedosa en en el entorno aeronáutico, probada exitosamente en el marco del proyecto AIRPORTS para tratar vuelos de grandes dimensiones. Así, se espera que efectivamente, su integración al modelo de reordenación de los mensajes ADS-B de una trayectoria que plantea este Trabajo Fin de Grado sea exitosa.

La Tabla 5.1 muestra los resultados de aplicación de la técnica de *ventanas de ejecución* utilizando el algoritmo *2-Opt* sobre varias trayectorias reales. Obsérvese como, sin aumentar la longitud de la solución obtenida, sí se reduce notablemente el tiempo de ejecución (nunca por encima de la décima de segundo usando la técnica).

Para las ejecuciones, se han tomado como referencia cinco conjuntos de datos, suministrados por *Boeing*, correspondientes a los siguientes vuelos:

- **IBE04HT**: Madrid Barajas-Asturias con 1.090 mensajes ADS-B.
- **IBE04NL**: Asturias-Madrid Barajas con 1.151 mensajes ADS-B.
- **IBE0519**: A Coruña-Madrid Barajas con 1.105 mensajes ADS-B.
- **IBE05DK**: Madrid Barajas-A Coruña con 1.079 mensajes ADS-B.
- **RYR9KY_4CA97C**: Ibiza-Barcelona El Prat con 533 mensajes ADS-B.

La Figura 5.7 muestra una reconstrucción de cada vuelo generada a partir de su conjunto de mensajes ADS-B asociados.



(a) IBE04HT (b) IBE04NL (c) IBE0519

(d) IBE05DK (e) RYR9KY_4CA97C

Figura 5.7: Representación de los mensajes ADS-B de cada trayectoria sobre un mapa creado con `Leaflet`

## 5.5. Modelo de corrección de timestamps

En esta sección se analiza la última de las características de la aplicación en estudio [CAR-3], que aborda la corrección de los *timestamps* de los mensajes ADS-B tras haber sido ejecutada una reordenación.

Como hemos visto en el Capítulo 3, los mensajes ADS-B asociados a un vuelo contienen un conjunto de datos necesarios para el seguimiento. Entre ellos se encuentra una marca temporal o *timestamp* que para cada mensaje representa el instante temporal en que fue recibido por un sensor y que permite establecer un orden en el conjunto de ellos. Sin embargo, como consecuencia de los problemas de alineamiento temporal que presenta la tecnología ADS-B (que se trata en el Capítulo 3), esta ordenación temporal difiere del verdadero orden en que los mensajes son enviados por la aeronave. A lo largo del presente capítulo hemos abordado el problema de reordenación de los mensajes ADS-B asociados a un vuelo mediante la implementación de un modelo que tiene por objeto encontrar esa disposición temporal real de los mensajes ADS-B y que habilita la reconstrucción de la trayectoria seguida por la aeronave tomando en consideración los datos de geolocalización contenidos en estos mensajes. Este desarrollo permite llevar a cabo las reordenaciones anteriores mediante la ejecución de uno de los algoritmos aplicables a resolver supuestos del problema del viajante, problema que, como ya se ha comentado, es la base de este planteamiento teórico.

Sin embargo, tras reordenar los mensajes ADS-B de un vuelo se genera una inconsistencia en los datos de seguimiento, pues el nuevo orden de los mensajes no se corresponde con el que resultaría de ordenar ascendentemente sus marcas temporales. Es por ello que es necesario efectuar correcciones sobre el *timestamp* de ciertos registros de manera que el nuevo orden de los mensajes sea coherente con su *timestamp*.

Este trabajo tiene entre sus aspiraciones abordar este nuevo problema de rectificación o refinamiento de las marcas temporales (en línea con el objetivo [OB-2]), consecuencia del procedimiento de seguimiento que efectúa ADS-B. En primer lugar, se lleva a cabo la creación de un modelo matemático que permita efectuar estas correcciones, así como obtener un nuevo valor de *timestamp* para cada punto a partir de los datos de los mensajes ADS-B inmediatamente anteriores y posteriores.

Dado un vuelo con cierto conjunto de mensajes ADS-B asociado, y una reordenación de estos mensajes, se establece una partición sobre los mensajes de manera que queden divididos en dos subconjuntos en función de si su posición en el orden cambia o no tras la reordenación. Esta división separa los mensajes de la forma siguiente:

1. En el primero de ellos, aquellos mensajes cuya posición antes y después de la reordenación coincide.

2. En el segundo, aquellos mensajes cuya posición antes y después de la reordenación cambia.

Así por ejemplo, el mensaje que ocupaba la posición tercera (existen solo dos mensajes con marcas temporales anteriores) antes de reordenar, estará en el primer grupo si tras la reordenación sigue estando en esa posición, y pertenecerá al segundo grupo en caso contrario.

Una vez se ejecuta la reordenación, quedan establecidas estas dos categorías de men-

sajes, considerando a la primera de ellas como el conjunto de mensajes cuyos datos de *timestamp* son correctos, de manera que se supone que no han sufrido alteraciones sobre su marca temporal. Estos mensajes se toman como referencia en el modelo a la hora de efectuar las correcciones, y por tanto no se rectifican. Por otro lado, para los mensajes que pertenecen a la segunda categoría se considera que sus marcas temporales no representan fielmente el instante temporal en que se enmarcan, por lo que se establece que deben ser rectificadas. En esta corrección se parte de las marcas temporales de los mensajes pertenecientes al primer grupo más próximos en el orden a cada punto del segundo grupo.

El procedimiento de corrección se divide en dos fases. En la primera, se corrigen los puntos primero y último de la ruta en caso de que alguno de ellos pertenezca al segundo grupo. A continuación, la segunda fase consistiría en la corrección del resto de puntos intermedios del segundo grupo.

El modelo matemático aplicado a la corrección de cada punto parte de los datos de velocidad y distancia recorrida por la aeronave de los mensajes ADS-B más próximos. Dado cualquier punto del grupo segundo intermedio (que no sea el primero o el último de la ruta tras la reordenación), se localiza el punto bueno anterior y el posterior más cercanos, y de ellos se toman los datos de *timestamp*, $t_{ant}, t_{pos}$ que se consideran correctos (que no necesitan de corrección). A continuación se calcula la velocidad media de la aeronave $v_{ant}$ en el segmento de trayectoria que va desde el punto bueno anterior más cercano hasta el punto actual, así como la velocidad media $v_{pos}$ que alcanza en el segmento que va desde el punto actual hasta el punto bueno posterior más próximo. Para obtener estos dos datos es necesario disponer de los datos de velocidad de los mensajes ADS-B que, según el nuevo orden, se encuentran entre los puntos buenos previamente mencionados. Más adelante, en esta sección, veremos la correlación aparente entre velocidad y altura de la aeronave. Esto permite establecer una relación de proporcionalidad entre ambos datos para obtener los datos $v_{ant}$ y $v_{pos}$ a partir de la altura, considerando cierta esta correlación.

El nuevo *timestamp* de cada punto dentro del segundo grupo viene dado por la expresión

$$\frac{(v_{pos}d_{pos}+1)t_{ant}+(v_{ant}d_{ant}+1)t_{pos}}{(v_{pos}d_{pos}+1)+(v_{ant}d_{ant}+1)} \tag{5.1}$$

donde

$v_{pos}$ es la media aritmética de las velocidades de los $n$ mensajes del grupo 1 inmediatamente posteriores al actual,

$v_{ant}$ es la media aritmética de las velocidades de los $n$ mensajes del grupo 1 inmediatamente anteriores al actual,

$d_{pos}$ es la distancia del punto actual con respecto al punto del grupo 1 inmediatamente posterior,

$d_{ant}$ es la distancia del punto actual con respecto al punto del grupo 1 inmediatamente anterior,

$t_{ant}$ es el *timestamp* del punto del grupo 1 inmediatamente anterior al actual y

$t_{pos}$ es el *timestamp* del punto del grupo 1 inmediatamente posterior al actual.

La constante 1 que se suma a cada uno de los pesos evita el caso en que ambos pesos son nulos, en el que el cociente sería igual a cero.

La expresión (5.1) no es más que la media ponderada de los *timestamps* de los puntos del primer grupo anterior y posterior más próximos, donde los pesos dependen de la velocidad media de la aeronave y de la distancia recorrida en el segmento que limitan los dos puntos buenos previos. Este modelo, sin embargo, no funciona cuando no existen puntos buenos en alguno de los dos lados del punto actual, cosa que ocurre cuando alguno de los dos extremos de la ruta son objeto de corrección (son del segundo grupo). Es por ello que se realiza la corrección de estos dos puntos previamente a la corrección de los puntos intermedios, para que, una vez corregidos, pasen a ser considerados como puntos del primer grupo evitando la situación anterior.

En el caso de los extremos los datos considerados no cambian: se parte de la velocidad media que alcanza la aeronave en el segmento de trayectoria que va desde el extremo considerado hasta el punto bueno más cercano, así como la distancia recorrida en ese segmento y el *timestamp* del punto del primer grupo. Así, y considerando un movimiento rectilíneo uniforme, se deriva la marca temporal del extremo considerado a partir de la expresión

$$t_0 \pm \frac{d}{v + 0{,}1}$$

donde

$t_0$ es la marca temporal asociada al punto bueno más cercano,

$v$ es la velocidad media de la aeronave en el segmento que define el extremo considerado y el punto bueno más próximo y

$d$ es la longitud del segmento de trayectoria.

El operando $\pm$ que se considera va en función de si el punto a corregir es el primero o el último: será $+$ para tratar el último punto y $-$ para el primero. Por su parte, la constante 0,1 se considera para evitar el caso en que la velocidad media de la aeronave es nula, y por tanto, el cociente igual a cero.

Como en el caso anterior, si no se disponen datos de velocidad se supone proporcional a la altura y se obtiene a partir de este dato.

Por último añadir que tras corregirse un punto pasa a formar parte de los puntos del primer grupo, por lo que en la corrección de los siguientes puntos, los puntos anteriormente corregidos son buenos a la vista del algoritmo.

Este modelo de corrección de marcas temporales se encuentra implementado en la aplicación de la que trata este capítulo, a fin de ser puesto en práctica. Tras ejecutar una reordenación, existen dos formas de visualizar sobre la aplicación la reordenación de los *timestamps* producida.

La primera de estas formas es sobre el *widget Tabla*, donde se muestra una tabla que contiene los datos de los mensajes ADS-B asociados al vuelo. En la tabla que se muestra en esta vista aparecen nuevas columnas que reflejan los resultados de la reordenación y de la corrección de las marcas temporales. Estas columnas son las siguientes:

- *distancia*: indica la longitud en kilómetros del camino que recorre todos los puntos de la trayectoria que se encuentran entre el primero y el punto actual en el orden que establece la nueva reordenación. Este valor de longitud se calcula como la suma de la longitud de los segmentos que unen cada punto con el siguiente, a través de la *Fórmula de Haversine* (7.1).
- *tag_original*: Indica la posición que ocupaba cada mensaje antes de la reordenación.
- *tag_ordenado*: Indica la posición que ocupa el mensaje tras la reordenación ejecutada.
- *timestamp_ordenado*: Indica el nuevo valor de *timestamp* que se asocia al mensaje tras la reordenación.

Por defecto, al acceder a este módulo las filas de la tabla aparecerán ordenadas conforme a la nueva ordenación obtenida, pero el usuario puede, a través de los controles de la tabla, modificar la ordenación de los registros respecto al valor de los mensajes en cualquiera de las columnas. En la Figura 5.8 se observan estos nuevos campos de la tabla. Obsérvese, recuadrado en rojo, como dos puntos cuyo orden se ha intercambiado obtienen nuevos valores de *timestamp* coherentes con su nueva posición.

La segunda forma de comprobar cómo se ha efectuado la corrección de los *timestamps* se observa en la gráfica que asocia la distancia recorrida por la aeronave (campo *distancia* anterior) con el nuevo *timestamp*. A esta componente se accede a través del *widget Mapas* y la pestaña *Distancia vs Tiempo* de la componente central. Sobre esta gráfica bidimensional se observa la correlación entre los distintos puntos que componen el vuelo. La Figura 5.9 ilustra lo que se visualiza para un ejemplo concreto. En verde, se representan los puntos "malos" (aquellos que han visto afectada su posición en la ruta al ejecutar la reordenación), cuyo *timestamp* ha sido modificado, mediante el modelo que representa (5.1). Por otro lado, los puntos en gris representan los puntos "buenos" (que no han cambiado su posición). Éstos mantienen su marca temporal al considerarla el modelo como correcta. En la Figura 5.10 se observa en mayor detalle como se han situado los puntos verdes tras haber sido interpolados, situándose en la línea recta que une los dos puntos grises inmediatamente anterior y posterior al grupo de puntos malos observado, variando su posición en función del valor que tengan en la columna *distancia* y de la velocidad asociada a los $n$ puntos buenos anteriores y posteriores, como ya hemos comentado.

| tag_original | tag_ordenado | distancia | timestamp_ordenado | timestamp |
|---|---|---|---|---|
| 951 | 951 | 413.89 | 1517594295 | 1517594295 |
| 952 | 952 | 414.11 | 1517594295 | 1517594295 |
| 953 | 953 | 414.42 | 1517594298 | 1517594298 |
| 954 | 954 | 414.54 | 1517594299 | 1517594299 |
| 956 | 955 | 415.04 | 1517594304 | 1517594305 |
| 955 | 956 | 415.11 | 1517594305 | 1517594304 |
| 957 | 957 | 415.71 | 1517594310 | 1517594310 |

Figura 5.8: Ejemplo de corrección de los timestamps tras haber ejecutado una reordenación sobre un vuelo, con los registros ordenados conforme al nuevo orden de los mensajes obtenido

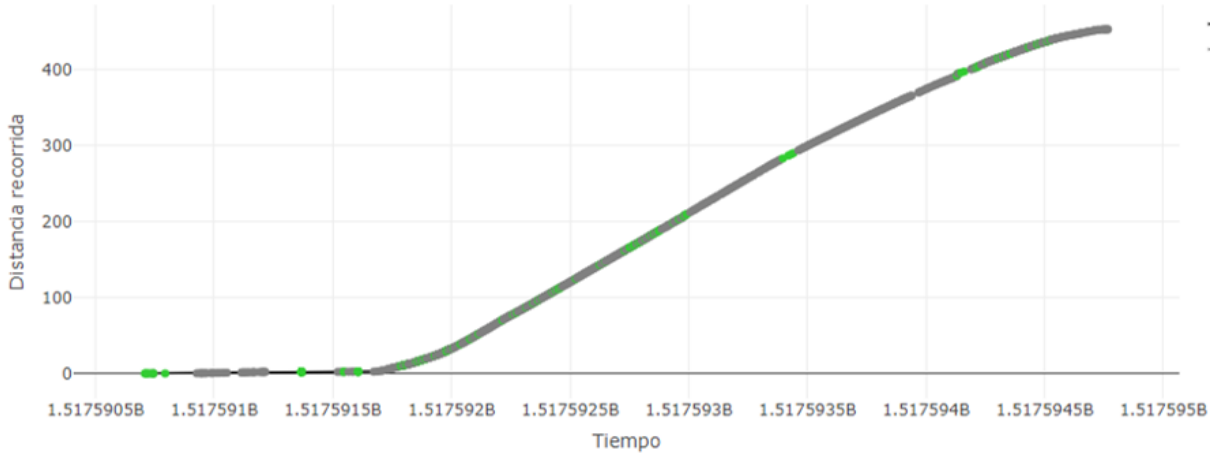



Figura 5.9: Ejemplo de corrección de los timestamps tras haber ejecutado una reordenación sobre un vuelo, vista de la gráfica *Distancia vs Tiempo* del *widget Mapas*

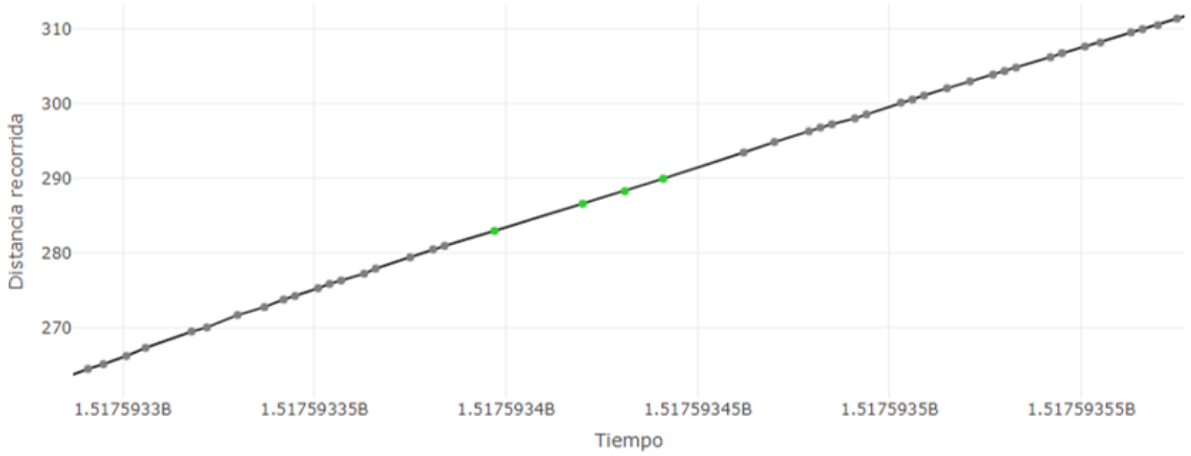



Figura 5.10: Ampliación de la gráfica que muestra la Figura 5.9

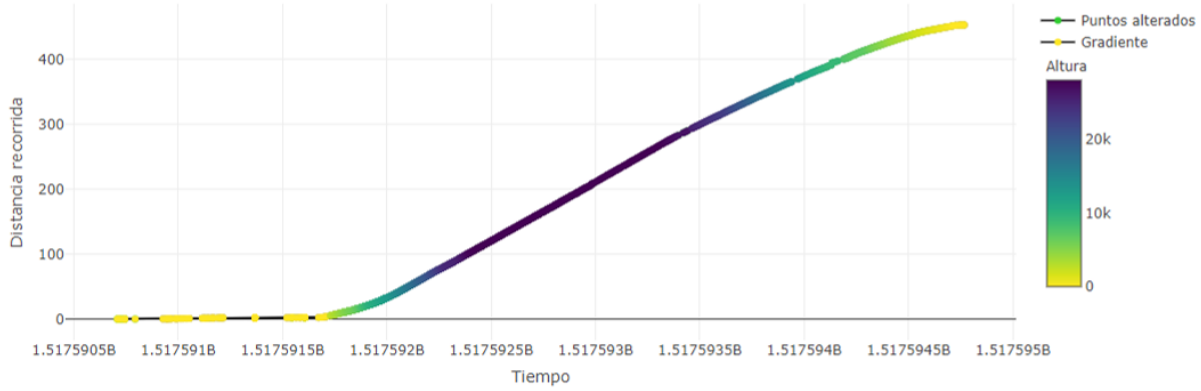



Figura 5.11: Gráfica Distancia vs Tiempo al representar la altura de cada punto a través de un gradiente de colores

Por último, la gráfica anterior puede ilustrar a su vez la relación entre el valor que obtienen los puntos en el campo *distancia*, su nuevo *timestamp* y su altura. En la Figura 5.11 se muestra la gráfica de la Figura 5.9 donde, a través de un gradiente de colores, es posible visualizar globalmente la asociación entre la altura de la aeronave y la distancia recorrida con respecto al tiempo. Obsérvese como la pendiente de la gráfica es más acusada cuando los valores de altura son altos (aeronave en vuelo), y menor cuando ésta no es tan elevada (en aeropuertos, despegando o realizando maniobras de aterrizaje). Este hecho se debe a que altura y velocidad de la aeronave son generalmente proporcionales.

# Capítulo 6

# Análisis de resultados

A lo largo del presente capítulo se presentan los resultados del estudio llevado a cabo a través de la aplicación presentada en el Capítulo 5. Este estudio cuenta con dos fases: en la primera se comparan todos los algoritmos entre sí al ser aplicados a la reordenación de un conjunto de trayectorias reales (ver Sección 6.1), mientras que en la segunda se ejecuta una gran batería de pruebas que incluye 96 vuelos reordenados con el algoritmo *2-Opt* a fin de medir su rendimiento (ver Sección 6.2).

Ambos estudios han sido realizados junto a María Zorita Mínguez en el marco de una beca de colaboración con el Departamento de Informática. Los vuelos que aparecen de aquí en adelante proceden de datos reales suministrados por *Boeing Research & Technology Europe*, sin los cuales la realización de este estudio, así como del presente Trabajo Fin de Grado, no habría sido posible.

## 6.1. Estudio comparativo de los algoritmos de resolución del TSP aplicados a la reordenación de mensajes ADS-B

Para elaborar esta primera parte del estudio, se ha tomado como referencia cinco conjuntos de datos procedentes de mensajes ADS-B, cada uno correspondiente a un vuelo concreto. Estos vuelos son los considerados en la breve comparativa que ilustra la eficacia de la técnica de las ventanas de ejecución en la Tabla 5.1.

Por otra parte, el sistema informático sobre el que se han ejecutado los algoritmos es un *ACER Aspire 5 A515-51G-751G* que incluye entre sus características más destacables un sistema operativo *Windows 10 Home*, procesador *Intel Core i7-7500U con velocidad de entre 2.7 GHz a 3.5 GHz* y memoria RAM de 8GB.

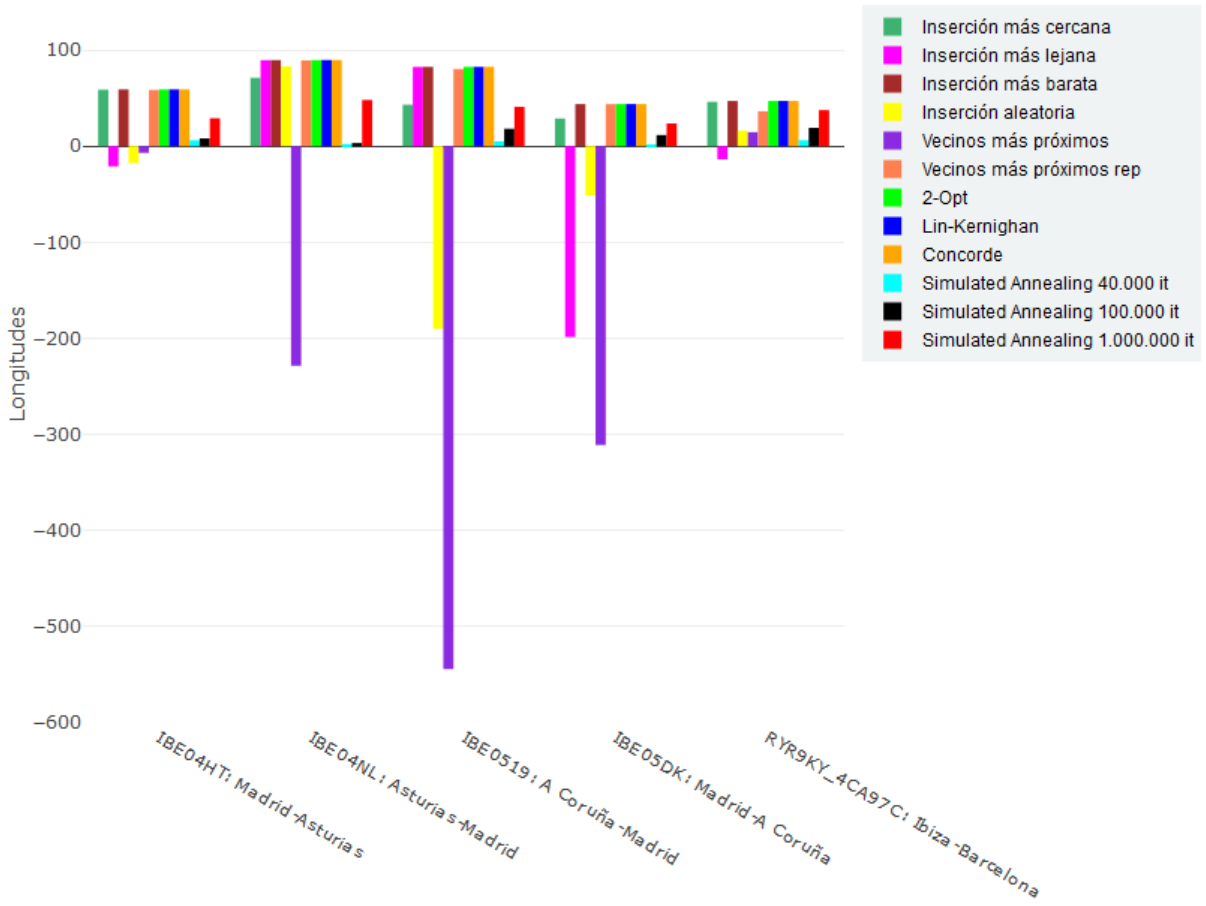



Figura 6.1: Diagrama de barras: Longitud mejorada (en km) registrada con respecto a la ruta de partida inicial

| **Algoritmo** | **IBE04HT** | | **IBE04NL** | | **IBE0519** | | **IBE05DK** | | **RYR9KY_4CA97C** | |
|---|---|---|---|---|---|---|---|---|---|---|
| **de resolución** | Long | Tmp | Long | Tmp | Long | Tmp | Long | Tmp | Long | Tmp |
| Sin aplicar algoritmo | 511,99 | - | 659,99 | - | 682,84 | - | 602,47 | - | 346.98 | - |
| Inserción más cercana | 453,22 | 5,46 | 588,89 | 7,24 | 639,72 | 5,60 | 573,73 | 5,59 | 300.85 | 0.78 |
| Inserción más lejana | 533,07 | 5,41 | 570,51 | 6,14 | 600,42 | 5,97 | 801,13 | 4,97 | 360.63 | 0.80 |
| Inserción más barata | 452,95 | 3,76 | 570,42 | 4,03 | 600,38 | 3,89 | **558,69** | 3,04 | **299.99** | 0.49 |
| Inserción aleatoria | 529,76 | **0,05** | 576,98 | **0,05** | 873,01 | **0,04** | 653,8 | 0,05 | 330.80 | 0.02 |
| Vecinos más próximos | 518,98 | **0,05** | 888,53 | **0,05** | 1.227,06 | 0,05 | 913,47 | **0,04** | 332.44 | **0.01** |
| Vecinos más próximos repetitivo | 453,49 | 66,22 | 570,76 | 67,54 | 602,57 | 55,41 | 558,77 | 51,36 | 310.66 | 6.75 |
| 2-Opt | 452,95 | 0,42 | 570,42 | 0,63 | 600,38 | 0,5 | **558,69** | 0,28 | **299.99** | 0.03 |
| Lin-Kernighan | 452,97 | 20,85 | 570,39 | 18,58 | 600,38 | 22,14 | **558,69** | 16,97 | **299.99** | 3.62 |
| Concorde | **452,62** | 17,37 | **569,77** | 20,06 | **600,26** | 14,11 | **558,69** | 8,40 | **299.99** | 2.49 |
| Simulated annealing, 40000 iteraciones | 505,91 | 5,04 | 657,75 | 5,08 | 677,75 | 5,22 | 601,86 | 5,59 | 340.84 | 3.65 |
| Simulated annealing, 100000 iteraciones | 503,99 | 12,20 | 656,83 | 14,06 | 664,79 | 12,56 | 590,95 | 11,98 | 327.98 | 9.17 |
| Simulated annealing, 1000000 iteraciones | 482,97 | 141,86 | 611,94 | 133,66 | 641,93 | 120,99 | 578,84 | 120,21 | 309.50 | 89.64 |

Tabla 6.1: Comparativa de los algoritmos en longitud de la ruta encontrada (Long) en kilómetros y en tiempo de ejecución (Tmp) en segundos

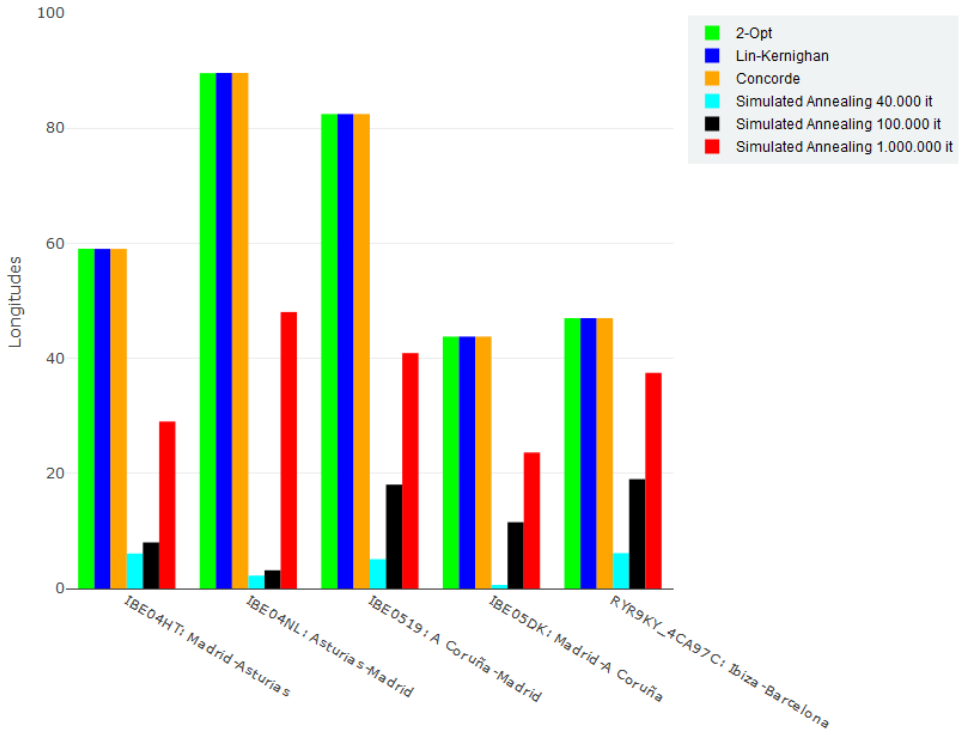



Figura 6.2: Diagrama de barras: Longitud mejorada (en km) registrada con respecto a la ruta de partida inicial obtenida por los métodos no constructivos

En la Tabla 6.1 se muestran los datos obtenidos fruto de esta primera parte del estudio contenido en este capítulo. Tal y como se puede apreciar, en los algoritmos más simples la calidad de las rutas encontradas difiere en función de la ruta a la que se aplican. Este comportamiento se observa en la mayoría de métodos constructivos. Exceptuamos de esta irregularidad al algoritmo de *inserción más barata*, que para cada uno de los cinco vuelos obtiene rutas muy próximas a la más corta encontrada por algoritmos de implementación más compleja como son el *Concorde* o el *Lin-Kernighan* (hasta un $0,11\,\%$ más larga que la encontrada por *Concorde* para el IBE04NL). Por su parte, el algoritmo *2-Opt* obtiene soluciones que en todos los casos son de longitud muy similar a la mejor encontrada (como el algoritmo de *inserción más barata*, hasta un $0,11\,\%$ más larga que la encontrada por *Concorde* para el IBE04NL), mientras que el tiempo de ejecución requerido para ello se reduce notablemente, obteniendo la solución más corta encontrada y en el menor tiempo registrado para el vuelo IBE3118. Este buen comportamiento del *2-Opt* se debe a que la ruta inicial de la que parte (orden de llegada de los mensajes a estaciones receptoras), es muy similar a la correcta ordenación de los mensajes. En estas condiciones, como se comenta en la Sección 4.5, el *2-Opt* es muy adecuado por el pequeño alcance de las modificaciones que realiza sobre la ruta (denominadas *2-changes*).

En cuanto a *Lin-Kernighan* y *Concorde* se obtienen rutas muy adecuadas en todos los vuelos dada la elevada complejidad de ambos algoritmos. Sin embargo, en tiempo de ejecución es el *Concorde* el que es preferible, requiriendo por ejemplo el $49,5\,\%$ del tiempo necesitado por *Lin-Kernighan* para obtener la misma solución. Estos datos registran diferencias respecto a los resultados del estudio contenido en la Sección **??**, donde el tiempo de ejecución del *Concorde* en ocasiones triplicaba el requerido por *Lin-Kernighan* para la misma instancia.

Por último, *Simulated Annealing* realiza leves mejoras sobre la ruta inicial dada, obteniendo resultados cada vez mejores a medida que se aumenta el número de iteraciones consideradas para el algoritmo. Así, para el vuelo IBE04HT, la ruta encontrada es un 1,2 % más corta que la original para 40,000 iteraciones del algoritmo, un 1,66 % para 100,000 iteraciones y un 5,67 % para 1,000,000 iteraciones. Sin embargo, cuantas más iteraciones, más tiempo de ejecución se requiere, el cual supera los 2 minutos de ejecución en este último caso para todos los vuelos salvo el IBE3118, que requiere de minuto y medio. Los resultados contrastan fuertemente con los del algoritmo *2-Opt*, que en ningún vuelo ha registrado tiempos de ejecución superiores al segundo.

Como conclusión de esta parte, podemos afirmar, por un lado, la eficacia del algoritmo *Concorde* a la cual ahora podemos sumar, a diferencia de lo registrado en la Sección **??**, los buenos resultados en términos del tiempo de ejecución. Sin embargo, no resulta adecuado para instancias grandes del problema debido principalmente a su complejidad computacional. En estos casos es preferible hacer uso de un algoritmo más simple como el *2-Opt*, cuyos resultados en términos de longitud casi alcanzan los del *Concorde* pero reduciendo en varios órdenes el tiempo de ejecución requerido. Este tiempo es casi inapreciable en la mayoría de los casos para el *2-Opt* frente a los más de 10 segundos que el

*Concorde* ha requerido para resolver algunas de las rutas propuestas. El comportamiento del resto de algoritmos, exceptuando al *simulated annealing*, resulta altamente irregular, obteniendo buenos resultados para ciertas instancias y descartables en otras. Por su parte, el *simulated annealing* garantiza mejorar, al menos levemente, la ruta de partida dada, pero requiriendo para ello un elevado tiempo de ejecución.

## 6.2. Análisis del comportamiento del algoritmo 2-Opt a la reordenación de mensajes ADS-B en base a métricas de eficiencia

En esta segunda parte del estudio se examina el comportamiento del algoritmo *2-Opt*, como el más adecuado de las *heurísticas de mejora local*, al aplicarse a la reordenación de un conjunto grande de trayectorias.

Se toman 96 vuelos europeos de diversa naturaleza, y sobre ellos se ejecuta varias veces el algoritmo *2-Opt* con diferentes modos de ejecución y se analizan los resultados en base a diferentes métricas.

Los modos de ejecución considerados son los siguientes:

**M1** Se ejecuta el algoritmo considerando al conjunto íntegro de todos los mensajes ADS-B del vuelo como un supuesto del problema del viajante donde cada mensaje ADS-B conforma una ciudad por la que ha de pasar el camino solución.

**M2** Se ejecuta el algoritmo haciendo uso de la *técnica de las ventanas de ejecución* (ver Subsección 5.4.1), con tamaño máximo de ventana fijado en 100 registros y el solapamiento en 20.

**M3** Se ejecuta el algoritmo haciendo uso de la *técnica de las ventanas de ejecución* con dos tamaños máximos de ventana prefijados en 100 registros para la parte central del vuelo y 20 para las secciones restantes, y con solapamiento 20 para la parte central y 5 para el resto. Para separar la parte central del vuelo del inicio y final del mismo se utilizan los datos de altura como discriminante, eligiendo como parte inicial de la ruta el conjunto de mensajes que van desde el más antiguo registrado hasta el primero que registra una altura mayor a 6.000 pies, y como parte final a la que abarca el conjunto de mensajes que van desde el más reciente hasta el primero en que el vuelo desciende de los 6.000 pies para no volver a situarse su altura por encima de este valor.

Por otro lado, para analizar la ingente cantidad de resultados que se obtiene, se determinan las siguientes métricas para evaluar el comportamiento de las ejecuciones:

- **Media**: consiste en la media aritmética de un tipo de datos para las ejecuciones para todos los vuelos con un mismo modo de ejecución. En `R` viene dada por la

función `mean()`.

- **MAD** o *median absolute deviation*: consiste en la mediana de las desviaciones absolutas con respecto a la media de todas las observaciones. Esto es: dadas las observaciones $\{X_1, X_2, ..., X_n\}$, la MAD es la mediana del conjunto $\{|X_1 - \overline{X}|, |X_2 - \overline{X}|, ..., |X_n - \overline{X}|\}$, siendo $\overline{X}$ la media muestral. En `R` viene dada por la función `mad()`.
- **Desviación típica muestral (DT)**: dadas las observaciones $\{X_1, X_2, ..., X_n\}$, la desviación típica muestral viene dada por

$$s = \sqrt{\frac{\sum_{i=1}^{n} |X_i - \overline{X}|}{n-1}}$$

En `R`, este estadístico viene dada por la función `sd()`.

A su vez, el sistema informático sobre el que se han ejecutado los algoritmos es un *HP Pavilion x360 Convertible* que incluye entre sus características más destacables un sistema operativo *Windows 10 Home*, procesador *Intel Core i5-7200U con velocidad de 2.5 GHz* y memoria RAM de 8GB.

La Tabla 6.2 contiene los resultados de esta segunda parte del estudio. En ella se observa que el *2-Opt* obtiene de forma general mejoras importantes en todos los modos de ejecución. Los mejores resultados en términos de longitud de la ruta encontrada, se obtienen con el modo M1, que, sin embargo, requiere de tiempos de espera mucho mayores, como indica la media muestral, con una alta variabilidad como indica la desviación típica y la MAD.

| | **Longitud (km)** | | | | **Porcentaje mejora** | | | **Tiempo ejecución (seg)** | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **Original** | **M1** | **M2** | **M3** | **M1** | **M2** | **M3** | **M1** | **M2** | **M3** |
| **Media** | 1868.65 | **1455.13** | 1487.38 | 1495.90 | **15.88** % | 14.82 % | 14.54 % | 31.67 | **0.03** | 0.25 |
| **MAD** | 948.83 | **743.11** | 786.70 | 776.69 | 7.83 % | 8.53 % | **7.70** % | 11.66 | **0.03** | **0.03** |
| **DT** | 1265.18 | **838.36** | 871.94 | 880.17 | **16.24** % | 15.14 % | 14.47 % | 49.29 | **0.03** | 1.11 |

Tabla 6.2: Resultados de la segunda parte del estudio preliminar

Combinando los resultados de tiempo de ejecución y longitud de la ruta encontrada es posible afirmar la mayor adecuación del modo M2 frente al resto, con el que se obtienen los mejores resultados de tiempo de ejecución (media más baja y menor variabilidad, dada por los datos de desviación típica y MAD), y el segundo mejor en longitud.

En este sentido, es posible afirmar que el uso de la técnica de descomposición del problema (aplicada en los modos M2 y M3), es adecuada al ser aplicada junto al algoritmo *2-Opt* frente a la resolución del problema completo. Su uso reduce la fiabilidad a cambio de una ganancia temporal importante, por lo que resulta recomendable cuando las prestaciones del sistema no son suficientes para resolver en un tiempo razonable una instancia del problema debido a su tamaño.

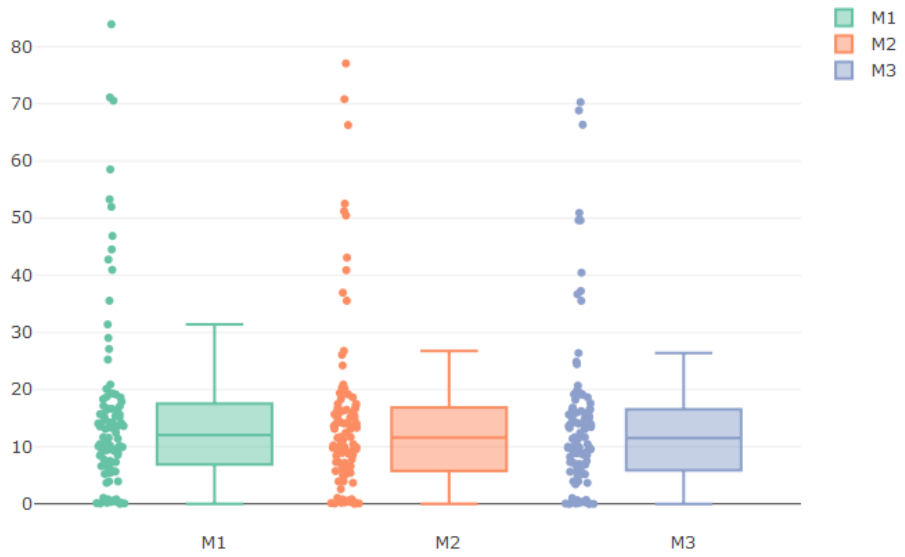



Figura 6.3: Diagrama de cajas: Efectividad (en %) de mejora en longitud registrada con respecto a la ruta de partida inicial comparando los resultados para cada modo de ejecución

# Capítulo 7

# Implementación escalable

El análisis preliminar del proyecto, expuesto en el Capítulo 6, arroja las primeras conclusiones prácticas acerca del comportamiento de los algoritmos aplicados a la reordenación de trayectorias reales.

En este capítulo se describen los siguientes pasos llevados a cabo, orientados al desarrollo de una implementación escalable de un algoritmo de reordenación que pueda ser posteriormente integrado a un sistema de gestión del tráfico aéreo o *Air Traffic Management* (ATM).

La primera tarea a llevar a cabo en esta nueva fase del proyecto será la elección del algoritmo más adecuado para llevar a cabo la reordenación de trayectorias de aeronaves a partir de los datos de sus mensajes ADS-B asociados. El estudio preliminar anterior aporta un gran conjunto de información asociada a la ejecución de los algoritmos considerados sobre un gran conjunto de trayectorias reales, y como conclusión el buen y adecuado comportamiento del algoritmo *2-Opt* frente a otros algoritmos estudiados. Este hecho justifica la elección de la familia de los métodos *k-Opt* para ser implementada en esta etapa.

El desarrollo se realiza en *Java*, pues en *Java* será también la implementación escalable a realizar posteriormente en *MapReduce*. La elección de este lenguaje de programación viene sustentada a su vez por disponer de alguna experiencia previa personal trabajando con este lenguaje de programación, al requerirse su uso a lo largo del grado en varias asignaturas como *Programación Orientada a Objetos* [40812] o *Programación y Estructura de Datos* [40816].

La estructura del presente capítulo es la siguiente: primero, en la Sección 7.1 se describen los detalles de implementación de los métodos *k-Opt*. A continuación, en la Sección 7.2 se expone el desarrollo de la implementación en Java asociada a esta parte del proyecto. En la Sección 7.3 se analiza el comportamiento de esta nueva implementación, ejecutando reordenaciones sobre trayectorias reales y comparando los resultados con los obtenidos usando las implementaciones de las librerías de `R`. Para finalizar, se describe el

proceso de adaptación de la aplicación al modelo de programación en *MapReduce*, en la Sección 7.4.

## 7.1. Implementación del *k-Opt*

Para implementar el algoritmo se toma como referencia el Algoritmo 1, que se incluye en la Sección 4.5.

La implementación en Java se obtendría por tanto realizando iteraciones con un bucle `while` cuya condición de parada sea cuando no haya ningún *k-change* por realizar sobre la ruta. Sin embargo, para evitar que en rutas grandes y para un alto valor de $k$ el valor de *k-changes* a probar sea desmedido, se introduce un atributo `int` que fije el número total de *k-changes* que se pueden probar sobre una ruta sin obtener ninguna mejor. De esta forma la condición de parada del bucle se sustituye por deternerse si se sobrepasa el número de *k-changes* máximo fijado.

En cada iteración, el algoritmo prueba una modificación sobre la ruta actual obteniendo una nueva, calcula la longitud de esta nueva ruta y la compara con la actual. En caso de ser más corta sustituye la ruta actual por la nueva y reinicia las variables del bucle. El orden en que se realizan los *k-changes* es arbitrario. Se generana cada iteración un número aleatorio que indica cual de todos los posibles *k-changes* se realizará.

Para comenzar a iterar, lo primero que es necesario es determinar qué *k-changes* son posibles de realizar sobre la ruta, para lo cual es necesario imponer una serie de restricciones que permita dar con ellos, así como encontrar un tipo de datos que permita almacenar estos *k-changes* y eliminarlos una vez que han sido probados. Este tipo, con las características dinámicas requeridas, bien podría ser un `ArrayList<int[]>`, que permite el acceso aleatorio a cualquiera de sus posiciones, a diferencia de otros tipos estructurados, así como la eliminación de elementos a partir de su posición en la estructura. Por otro lado es necesario especificar las restricciones a tener en cuenta para poder determinar los *k-changes*. Un *k-change*

- debe eliminar $k$ aristas determinadas de la ruta y sustituirlas por otras $k$ determinadas por aquellas que fueron suprimidas y
- no puede eliminar dos aristas consecutivas de la ruta (que confluyan en un mismo vértice), pues el *k-change* daría como resultado un *r-change* donde $0 < r < k$ (suponiendo como *1-change* la modificación que transformaría la ruta en ella misma).

Así, por tanto, bastaría con que el `ArrayList` almacene $k - uplas$ de enteros no negativos donde cada uno de ellos indique una de las aristas a eliminar. De esta forma es necesario numerar todas las aristas de la ruta. Por otro lado se debe imponer la condición de que ninguna $k - upla$ contenga dos números consecutivos, o por la primera y última arista (recordemos que se añade una ciudad artificial que una el primero y el último

de los nodos, por lo que las aristas que unen la primera y última ciudad con la ciudad artificial también deben ser tenidas en cuenta como parte de la ruta). Asimismo, para evitar que cada *k-change* se pruebe más de una vez (ya que, por ejemplo, la $2-upla$ $(n, m)$ equivaldría a la $2-upla$ $(m, n)$ en cuanto a que supone eliminar las mismas aristas y por tanto reemplazarlas por las mismas dos aristas) se añade la restricción de que cada par sea de la forma $(a, b)$ donde $a < b$.

En cada iteración, se escoge aleatoriamente el *k-change* a realizar, generando un número aleatorio que indique la posición del `ArrayList` de *k-changes* a considerar. Tras comparar la nueva ruta obtenida con la actual se elimina el *k-change* probado del `ArrayList`, para que no vuelva a salir, y prosigue la ejecución. En caso de haber obtener una nueva ruta más corta que la actual se reinicia el `ArrayList` de *k-changes* para poder probar todos de nuevo. En el momento en que el `ArrayList` quede vacío no podrá mejorarse la ruta a través de *k-changes* lo que desencadena el final de la ejecución del algoritmo.

## 7.2. Desarrollo de una aplicación en Java que realice reordenaciones a través de la implementación de los métodos *k-Opt*

En el contexto de nuestro problema, una vez se ha llevado a cabo la nueva implementación de los algoritmos *k-Opt*, se requiere de una aplicación que la contextualice en el marco de la reordenación de trayectorias de aeronaves. El objetivo de este desarrollo es la obtención de una aplicación que, a partir de conjuntos de mensajes ADS-B asociados a un mismo vuelo, sea capaz de reordenarlos haciendo uso de la implementación del algoritmo *k-Opt* realizada, y genere como salida la nueva reordenación de los datos, así como información complementaria necesaria para conocer la eficacia y eficiencia de la ejecución del algoritmo.

Esta aplicación, se desarrolla a modo de prototipo de la implementación escalable que se pretende desarrollar, a fin de estudiar la nueva implementación de los algoritmos *k-Opt* realizada, y tiene las siguientes características:

- La aplicación es capaz de leer ficheros *csv* de una carpeta *data_vuelos*. Cada uno de estos ficheros contiene registros con los mensajes ADS-B asociados a un vuelo.
- La aplicación procesa los datos contenidos en cada registro, eliminando aquellos con valores no tratables o con formato incorrecto en aquellos campos necesarios a la hora de realizar la reordenación como son los datos de *longitud*, *latitud* o *timestamp* del mensaje.
- Una vez se han leído todos los registros de un fichero, y previamente a la reordenación de los mensajes, los registros de la tabla se ordenan ascendentemente por su valor de *timestamp*

- Se ejecuta el algoritmo *k-Opt*, con el valor de $k$ elegido, tras haber procesado la lectura de los datos de un vuelo considerando como ciudades de la instancia del TSP a resolver cada uno de los mensajes ADS-B. Para poder transformar el problema de reordenación de trayectorias en el TSP es además necesaria la introducción de una ciudad artificial que se una a todas las demás a través de aristas cuyo coste o longitud sea 0. La aplicación, antes de comenzar a ejecutar el algoritmo, creará esta nueva ciudad así como la matriz de distancias, calculando la distancia dos a dos entre los mensajes ADS-B a partir de sus coordenadas geográficas (se descarta la altura a modo de simplificación). Este valor se obtiene haciendo uso de la **fórmula de la distancia de Haversine** que propone obtener el valor de la distancia entre dos puntos sobre la Tierra a través de la expresión

$$2r \cdot arcsin\left(\sqrt{sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + cos(\phi_1)cos(\phi_2)sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \tag{7.1}$$

  donde

  $\phi_1$ es el valor de la latitud del punto 1.

  $\lambda_1$ es el valor de la longitud del punto 1.

  $\phi_2$ es el valor de la latitud del punto 2.

  $\lambda_2$ es el valor de la longitud del punto 2.

  $r$ es el valor del radio terrestre en la métrica en la que se quiera obtener la distancia (en nuestro caso kilómetros). Su valor es de 6371 kilómetros.

- La aplicación ejecuta la implementación del *k-Opt* realizada sobre la nueva instancia del TSP generada a partir de los datos del vuelo, obteniendo como resultado una nueva reordenación de las ciudades.

- Tras finalizar la ejecución del algoritmo sobre un vuelo, la aplicación elimina la ciudad artificial, obteniendo así la reordenación definitiva de los registros. En este proceso, se localiza la ciudad artificial en la ruta. Las ciudades anexas a esta ciudad artificial en la ruta se corresponderán con el comienzo y el final de la nueva ruta del problema de reordenación de trayectorias. Para determinar cual de las dos ciudades es el inicio o la llegada, se determina tomar como origen aquella que menor *timestamp* tenga asociado, correspondiéndose entonces la ciudad con mayor *timestamp* con el final de la trayectoria.

- La aplicación mide el tiempo que tarda en ejecutarse el algoritmo a modo de determinar la eficiencia de la ejecución, así como la distancia de cada ruta intermedia obtenida durante la ejecución del algoritmo. Esto se hace con el fin de medir la bondad de las mismas y compararlas a través de este valor durante la ejecución.

- Una vez obtenida la reordenación de los mensajes ADS-B, se genera un nuevo fichero *csv* que contiene las filas del fichero *csv* leído previamente a la ejecución, ordenadas

en función de la nueva ordenación generada por la ejecución del algoritmo *k-Opt*.

## 7.3. Estudio de rendimiento de la nueva implementación para el *2-Opt*

Una vez finalizada la implementación del programa en Java se requiere analizar el funcionamiento de la nueva implementación de los algoritmos *k-Opt*, centrándonos en el método *2-Opt* ($k = 2$), en busca de determinar si los resultados obtenidos de su ejecución (sin limitar el número de *2-change* realizables sin mejora) son comparables a los obtenidos con la implementación de la librería `TSP` de `R`. En caso de obtener resultados similares estaremos en condiciones de afirmar que la implementación es adecuada con el fin de adaptar esta aplicación a la computación paralela.

Se realizan tres baterías de pruebas en cada una de las aplicaciones. La primera de ella consiste ejecutar una serie de reordenaciones sobre vuelos de 1000 o más puntos. Frente a los buenos resultados obtenidos en `R`, en `Java`, el tiempo de espera supera los dos minutos para cada vuelo, lo que hace el problema intratable. Esto nos indica la imposibilidad de hacer uso de la nueva implementación del *2-Opt* sin el uso de la técnica de *ventanas de ejecución*, a menos que la implementación se hiciese sobre un sistema de altas prestaciones que pudiese hacer frente el alto coste computacional requerido.

En la segunda batería se eligen ventanas de tamaño 100 y solapamiento 20, y se miden los resultados obtenidos en longitud de la nueva ruta obtenida y tiempo de ejecución. Los resultados de esta ejecución se ilustran en la Tabla 7.1.

Nótese que, la medida de la distancia de la ruta original en las aplicaciones de `R` y *Java* es diferente, debido al distinto orden de precisión del redondeo en la aplicación de la *Fórmula de Haversine* (7.1). Esta diferencia es notable al aplicarse esta expresión tantas veces como aristas hay en la ruta (que coincide con el número de ciudades menos una), al ser necesario obtener la distancia entre los puntos dos a dos a partir de sus coordenadas geográficas.

Teniendo en cuenta la observación del párrafo anterior, parece clara la similitud en tanto a la eficacia de reordenación de la implementación en *Java*, cuyos resultados en términos de longitud son del orden de los obtenidos por la implementación de la librería `TSP` de `R`. Por otro lado, y como era de esperar, la implementación en *Java* se queda algo atrás en cuanto al tiempo de ejecución, pero las diferencias, al tratarse subinstancias pequeñas, son poco apreciables, y por tanto el coste computacional es razonable.

La última batería de pruebas utiliza dos tipos de ventanas: las primeras de tamaño 100 y solapamiento 20 en vuelo, y las segundas de tamaño 25 y solapamiento 5 en aeropuertos, distinguiendo las partes del vuelo por el valor 6.000 de altura. Los resultados de esta prueba se incluyen en la Tabla 7.2.

| | **IBE04HT** | **IBE04NL** | **IBE0519** | **IBE05DK** | **IBE3118** | **IBE32GP** | **IBE32HV** |
|---|---|---|---|---|---|---|---|
| **Longitud Original R (km)** | 511,99 | 659,99 | 682,84 | 602,47 | 595,88 | 2622,32 | 2169,55 |
| **Longitud Original Java (km)** | 511,41 | 659,25 | 682,08 | 601,79 | 595,2 | 2619,38 | 2167,12 |
| **Nueva longitud R (km)** | 452,95 | 570,42 | 600,38 | 558,69 | 595,82 | 1924,6 | 1894,93 |
| **Nueva longitud Java (km)** | 452,55 | 570,25 | 599,9 | 558,15 | 595,15 | 1922,96 | 1892,06 |
| **Mejora longitud R ( %)** | **11,53 %** | **13,57 %** | **12,07 %** | **7,26 %** | **0,01 %** | **26,60 %** | 12,65 % |
| **Mejora longitud Java ( %)** | 11,50 % | 13,50 % | 12,04 % | 7,25 % | 0,01 % | 26,58 % | **12,69 %** |
| **Tiempo de ejecución R (s)** | **0,05** | **0,04** | **0,01** | **0,02** | **0,00** | **0,02** | **0,06** |
| **Tiempo de ejecución Java (s)** | 0,09 | **0,04** | 0,04 | 0,04 | **0,00** | 0,20 | 0,10 |

Tabla 7.1: Comparativa, resultados de la reordenación usando *2-Opt* con ventanas únicas de tamaño 100 y solapamiento 20 en las aplicaciones de `R` y *Java*

| | **IBE04HT** | **IBE04NL** | **IBE0519** | **IBE05DK** | **IBE3118** | **IBE32GP** | **IBE32HV** |
|---|---|---|---|---|---|---|---|
| **Longitud Original R (km)** | 511,99 | 659,99 | 682,84 | 602,47 | 595,88 | 2622,32 | 2169,55 |
| **Longitud Original Java (km)** | 511.41 | 659.25 | 682.08 | 601.79 | 595.2 | 2619.38 | 2167.12 |
| **Nueva longitud R (km)** | 452.96 | 570.85 | 600.38 | 558.69 | 595.88 | 1924.6 | 1894.93 |
| **Nueva longitud Java (km)** | 452.55 | 570.25 | 599.9 | 558.15 | 595.2 | 1922.96 | 1892.08 |
| **Mejora longitud R ( %)** | **11,53 %** | **13,50 %** | **12,07 %** | **7,26 %** | **0,00 %** | **26,60 %** | 12,65 % |
| **Mejora longitud Java ( %)** | 11,50 % | 13,50 % | 12,04 % | 7,25 % | **0,00 %** | 26,58 % | **12,69 %** |
| **Tiempo de ejecución R (s)** | **0.02** | **0,03** | 0,05 | **0,00** | **0,00** | **0.04** | **0.08** |
| **Tiempo de ejecución Java (s)** | 0,07 | 0,06 | **0,04** | 0,04 | **0,00** | 0,20 | 0,16 |

Tabla 7.2: Comparativa, resultados de la reordenación usando *2-Opt* con ventanas de tamaño 100 y solapamiento 20 en vuelo y de tamaño 25 y solapamiento 5 en aeropuertos, distinguiendo por altura 6000, en las aplicaciones de `R` y *Java*

Las diferencias entre los resultados de la tercera batería en relación con los de la

segunda son mínimos. En el caso de `R`, el tiempo de ejecución se reduce levemente mientras que la eficacia de la reordenación disminuye también, aunque de forma inapreciable. Por el contrario, la implementación de *Java* se mantiene aun más invariante, notando también la adecuación de la implementación a este tipo de ejecución.

En conclusión, es posible afirmar la adecuación de la nueva implementación en la reordenación de instancias pequeñas, así como su no recomendable utilización para la reordenación de trayectorias con un gran número de mensajes, a menos que se disponga de un sistema de altas prestaciones. Sin embargo, el obstáculo de disponer de instancias grandes es salvable mediante el uso de la técnica de las *ventanas de ejecución* que exponíamos en la Subsección 5.4.1, ya que el algoritmo pasa a ejecutarse sobre subinstancias más pequeñas, y por tanto tratables para la nueva implementación del algoritmo. A su vez, como se ha visto, el uso de esta técnica no reduce la eficacia de la reordenación con respecto a la que se obtendría aplicando el algoritmo a la trayectoria completa, haciendo a su vez esta técnica aplicable en la práctica, para la reordenación de trayectorias reales de aeronaves de forma exitosa.

## 7.4. Adaptación a MapReduce

Las pruebas destinadas a verificar el buen funcionamiento de la implementación realizada de los algoritmos *k-Opt* (concretamente el *2-Opt*) en *Java* concluyen de forma exitosa, al comprobar que a partir de los mismos vuelos, los resultados obtenidos en términos de longitud y de tiempos de ejecución son análogos a los de `R`.

Con todo esto, parece razonable concluir la utilidad de la implementación realizada para su aplicación al problema de reordenación de trayectorias. Sin embargo, no podemos olvidar el entorno en el que nos encontramos. Tal y como se mencionó en el Capítulo 1, la gestión del tráfico aéreo europeo requiere del uso de tecnologías *bigdata*, capaces de gestionar ingentes cantidades de datos en tiempo real. Esto se debe, por un lado, a la reciente utilización de las tecnologías de seguimiento ADS-B que aumentan la frecuencia de datos emitidos por cada aeronave, y que deben ser procesados por los sistemas ATM. Por otro lado, a la notable subida que prevé Eurocontrol en el número de aeronaves circulando por el espacio aéreo europeo [Eurocontrol, 2017].

Esta situación exige que la implementación anterior expuesta en la Sección 7.2 sea escalable, de forma que pueda ser aplicable a múltiples trayectorias de forma simultanea, esto es, que sea paralelizable sobre un cluster que dote a la implementación de la infrastructura necesaria para poder llevar a cabo este procesamiento de forma escalable, y por tanto, sin perder rendimiento al aumentar de forma ingente el número de datos a procesar. De esta forma se opta por una adaptación de la implementación a *MapReduce* que, como veremos en la siguiente subsección, permite el procesamiento paralelo de datos abstrayendo al programador de toda la complejidad interna que implica este tipo de computación.

### 7.4.1. Introducción a MapReduce

*MapReduce* [Dean y Ghemawat, 2008] se define como *un modelo de programación e implementación asociada para procesar y generar grandes volúmenes de datos, a través de la especificación de un mapper, o función que procesa pares de la forma clave-valor con el fin de generar un conjunto de pares intermedios clave-valor; y un reducer, o función que combina los valores intermedios asociados a una misma clave.*

Nace a través de *Google*, en busca de una implementación que pueda ser ejecutada sobre clusteres de grandes dimensiones así como el desarrollo de aplicaciones altamente escalables y fáciles de desarrollar, abstrayendo a los programadores de la complejidad de la computación paralela y distribuida, y permitiendo así a aquellos sin experiencia en computación distribuida el desarrollo de este tipo de programas.

En 2010, nace Hadoop como una implementación *OpenSource* de este paradigma, y con el que su uso se expande a toda la comunidad informática. Su desarrollo fue llevado a cabo en sus inicios por *Yahoo* y más tarde pasa a ser realizado por el *proyecto Apache*. Su inminente expansión se debió a las múltiples aplicaciones reales expresables a través de este paradigma.

El modelo de programación que propone *MapReduce* consiste en considerar como entrada un conjunto de pares *clave-valor*. Una misma clave usualmente será compartida por varios pares y actúa a modo de clasificador (los pares quedan agrupados por el valor de ésta). Los pares que comparten una misma clave serán procesados conjuntamente.

Existen dos funciones principales:

- *Map*: toma cada par de datos de entrada *clave-valor* y produce con ello un conjunto de pares *clave-valor* intermedios. Estos últimos son agrupados en base a su *clave* de manera que cada grupo de ellos es la entrada de una ejecución de la función *Reduce*.
- *Reduce*: toma como entrada uno de los subgrupos de pares *clave-valor* intermedios, resultado de la función *Map*. En su ejecución, el *Reducer* tiene como finalidad el procesado de todos los pares recibidos, dando como resultado una información con valor para el usuario, fruto del procesamiento. La salida más típica de esta función suele ser 0 o 1 en función del resultado del procesamiento.

El ejemplo más sencillo de aplicación de este modelo de programación, propuesto en Dean y Ghemawat [2008] es el famoso *WordCount*, que consiste en la lectura de varios textos procedentes de un conjunto de documentos con el fin de obtener el número total de apariciones de cada palabra en los textos.

Se propone así una función *map* que considere como entrada cada uno de los textos. En caso de que estos se encuentren en distintos documentos del sistema, se considera la URL del fichero como *clave* y el texto contenido como *valor*.

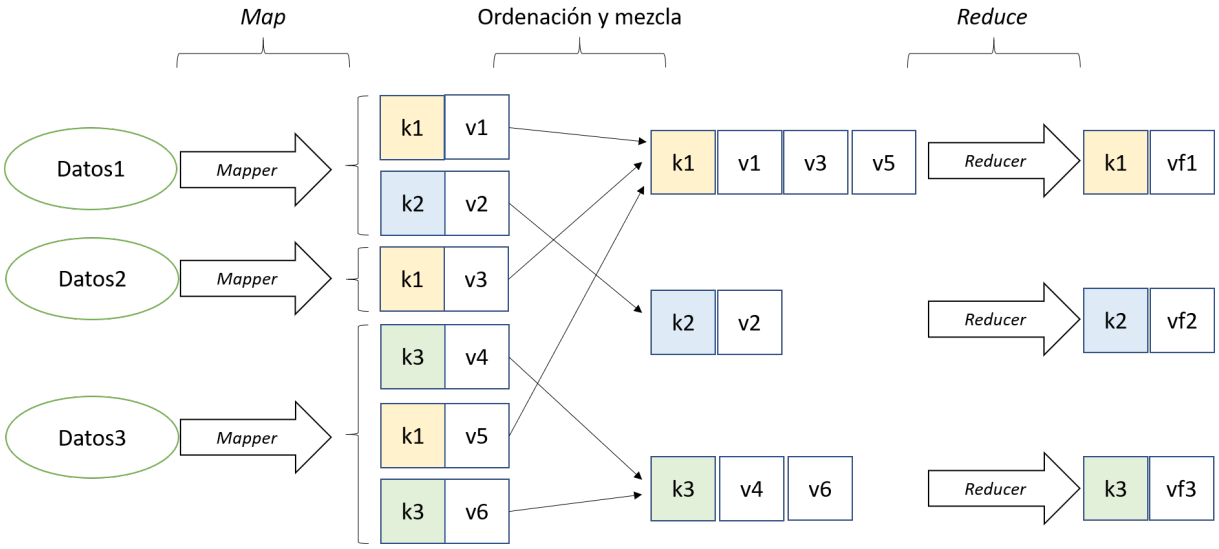



Figura 7.1: Esquema de funcionamiento de MapReduce

```
map(String key, String value):
   // key: document name
   // value: document contents
   for each word w in value:
      EmitIntermediate(w, "1");
```

La función *Map* se ejecuta tantas veces como documentos haya, y en cada una de ellas se recorre el texto, generando para cada palabra del mismo, un par intermedio que toma como clave la propia palabra, y como *valor* el literal "1".

La librería *MapReduce*, de forma transparente al programador, agruparía el conjunto total de claves intermedias generadas por las ejecuciones de la función *map*, generando un tipo de dato que engloba todos los valores asociados a una misma clave, en este caso, todas las apariciones del literal "1".

La función *Reduce* de este ejemplo tomaría como entrada cada uno de esos pares intermedios, ejecutándose así una vez por cada clave intermedia distinta generada, esto es, una vez por cada palabra distinta en el texto. Para cada palabra se asocia la lista que incluye todos los valores asociados a esa clave.

```
reduce(String key, Iterator values):
   // key: a word
   // values: a list of counts
   int result = 0;
   for each v in values:
      result += ParseInt(v);
```

```
        Emit(AsString(result));
```

Para cada ejecución de la función se produce el procesamiento, en este caso tan simple como contar el número de valores que contiene el *iterable*, de manera que se obtiene el número de veces que aparece la palabra en los textos. Este valor, así como la palabra asociada, es lo que devuelve la función *reduce*.

Obsérvese que al mismo tiempo, varias ejecuciones de la función *Reduce* están teniendo lugar, por lo que la aplicabilidad de las implementaciones que hacen uso de esta librería está claramente orientada a su ejecución sobre clusteres donde se procesen simultáneamente varios subconjuntos de datos asociados a diferentes claves.

Parece entonces clara la utilidad de este modelo de programación en el contexto del problema de reordenación de los mensajes ADS-B de una trayectoria. Se requiere una implementación capaz de llevar a cabo el procesamiento de un gran número de mensajes ADS-B por unidad de tiempo procedentes de diferentes vuelos que continuamente, y de forma simultanea, emiten datos de su posición que necesariamente han de ser procesados por el sistema en un breve periodo de tiempo.

Se requiere a su vez que el sistema sea capaz de reordenar paralelamente las señales ADS-B asociadas a distintos vuelos, de tal manera que se obtenga, a modo de salida, las rutas realmente recorridas por las aeronaves, lo que implica ejecuciones simultaneas del algoritmo *k-Opt* elegido, una por vuelo a procesar.

En la siguiente subsección se analizan los componentes de la nueva implementación así como las peculiaridades asociadas al proceso de adaptación de la anterior implementación en *Java*.

### 7.4.2. Implementación escalable en MapReduce

A continuación se exponen los detalles de la implementación escalable objeto de exposición en este capítulo, que parte de la implementación previa realizada en *Java* a modo de prototipo y que alberga la implementación realizada de los métodos *k-Opt*.

Antes de empezar a desarrollar la adaptación es necesario estudiar su viabilidad, pues para su elaboración se requiere disponer de herramientas adecuadas que permitan el desarrollo de implementaciones paralelizables. Se identifican así los siguientes elementos necesarios para esta implementación:

- Un sistema operativo que soporte Apache Hadoop sobre el que se puedan desarrollar implementaciones paralelizables simulando ejecuciones sobre un cluster. La UVa pone a disposición del proyecto una máquina virtual con el sistema operativo *Cloudera* donde poder realizar la implementación.

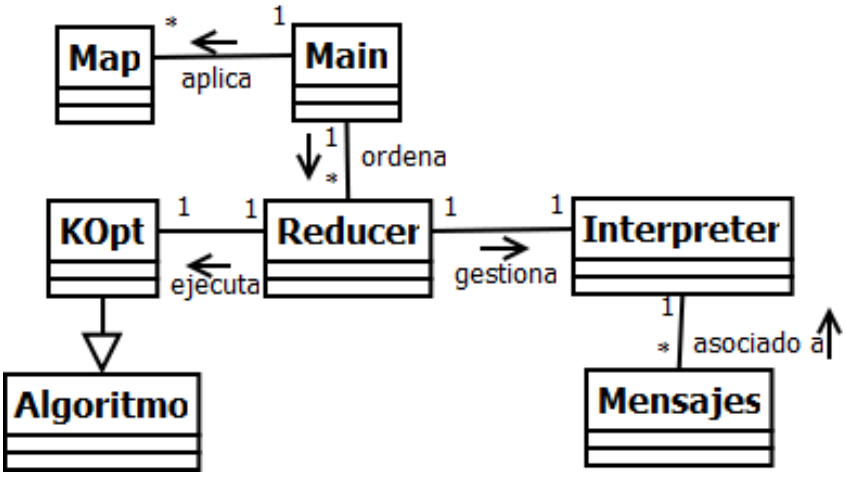



Figura 7.2: Análisis de la implementación en *MapReduce*: diagrama de clases

- Disponer de accesibilidad a material y contenidos de iniciación al uso del modelo de computación *MapReduce* al no tener experiencia previa trabajando con él.

Con estas necesidades cubiertas se comienza a adaptar la aplicación a *MapReduce*. Esta aplicación, engloba los objetivos de la anterior aplicación y añade a mayores: ejecución paralelizable de múltiples reordenaciones con los parámetros de ejecución deseados (modo de ejecución, tamaños de ventana, solapamiento, etc).

La nueva aplicación, cuyo uso final no prevé la interacción con los usuarios, únicamente incluye la funcionalidad de ejecutar reordenaciones sobre los datos que toma como entrada, posteriormente corregir sus marcas temporales y finalmente generar como salida la nueva ordenación de los mensajes ADS-B de las rutas elegidas.

Las clases en que se divide esta aplicación son las siguientes (ver Figura 7.2):

- `Algoritmo`: clase abstracta con aquellas propiedades (atributos) y operaciones (métodos) comunes a todos los algoritmos que resuelven instancias del problema del viajante. Diseñada por tanto para que cualquier implementación de un algoritmo de resolución del TSP extienda esta clase.

- `Interpreter`: clase que implementa el acceso a datos. Lee los ficheros *csv* en la carpeta *data_vuelos* por separado. Para cada registro de los ficheros crea una instancia de la clase `Mensaje`, donde almacena los valores leídos. A continuación, los mensajes de cada fichero son ordenados ascendentemente por *timestamp*.

- `KOpt`: clase que contiene la implementación de los algoritmos *k-Opt*. El constructor de esta clase admite un parámetro que permite seleccionar el valor de $k$, esto es, que algoritmo se ejecutará. Extienden la clase algoritmo al que añade la funcionalidad asociada a la inserción de una ciudad artificial, necesaria para la ejecución de este

algoritmo (al contrario que otros algoritmos de optimización más generales como el *Simulated Annealing*).

- `Main`: clase principal de la aplicación. Incluye el método `main()` que desencadena las distintas funcionalidades de la aplicación, y que instancia los objetos de las clases principales. Incluye toda la funcionalidad de *MapReduce* distribuida en varias subclases:

  - `Map`: implementa el *mapper* de la aplicación. Lee el contenido de un fichero que se introduce como parámetro del método `main()`. Se lee cada fila del fichero, que si es correcto, llevará 12 campos por fila situados en el orden

    $$id, timestamp, latitude, longitude, altitude, speed, vspeed, squawk, track, ground,$$
    $$leg, date$$

    donde las columnas que la lógica interna de la aplicación considera en su operativa son:

    *timestamp*: marca temporal del mensajes ADS-B

    *latitude*: valor de latitud en grados de la geolocalización de la aeronave al momento de emitir el mensaje.

    *longitude*: valor de longitud en grados de la geolocalización de la aeronave al momento de emitir el mensaje.

    *altitude*: altitud de la aeronave.

    *speed*: velocidad de la aeronave.

    Esta disposición de los campos se supone fija, pues es la que tendrán los *csv* sobre los que se espera que trabaje la aplicación.

    De cada fila se toma el valor del campo *leg* que será la clave del registro. De esta forma todos los mensajes ADS-B leidos se agrupan por el valor de este campo antes de aplicar reordenaciones.

    El *mapper* envía para cada registro, como clave el valor de *leg* y como valor la fila completa.

  - `Reduce`: implementa el *reducer* de la aplicación. Toma como entrada el conjunto de todas las filas (valores) del fichero de entrada con el mismo valor del campo *leg* (clave). A continuación toma los parámetros de ejecución que se pasan como parámetro al método `main()` para a continuación iniciar una ejecución del método *k-Opt*, donde el valor de $k$ viene dado entre estos parámetros. Tras ejecutarse la reordenación de los registros se aplica el método de corrección de los timestamps `corregirTimestamps()`, que a partir del resultado del método `runAlgorithm()` y del array de distancia recorrida, obtenido tras finalizar la

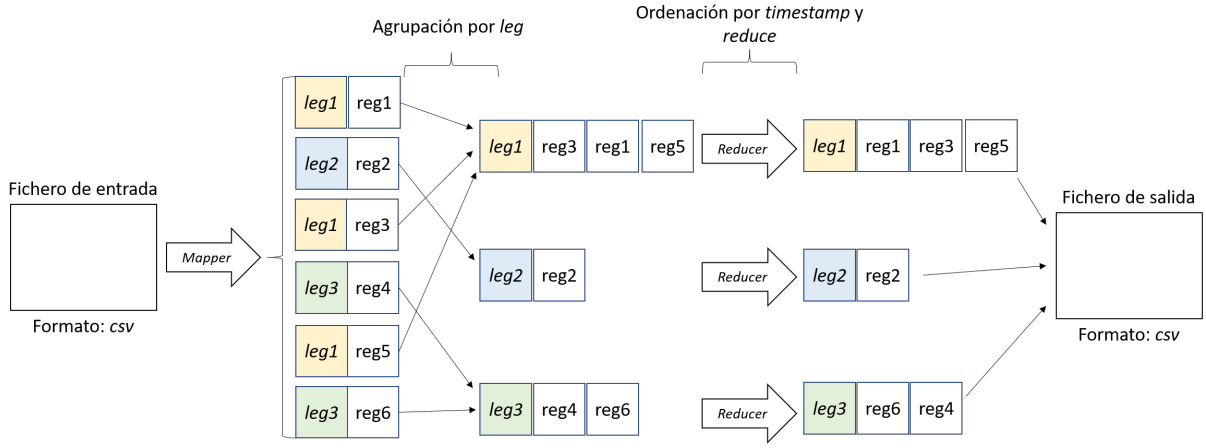



Figura 7.3: Procedimiento de procesado de los mensajes ADS-B por la implementación sobre *MapReduce*

ejecución del algoritmo. Tras esto el *reducer* genera los registros de salida, que incluyen la relación de campos

$$id, time, latitude, longitude, altitude, speed, vspeed, squawk, track, ground,$$

$$leg, date, distancia, time\_nuevo, tag\_orig, tag\_ord$$

donde las nuevas columnas son las siguientes:

*distancia*: es la distancia que ha recorrido la aeronave desde que ha iniciado la trayectoria.

*time_nuevo*: es la nueva marca temporal que se asocia al mensaje.

*tag_orig*: es la posición que ocupaba el mensaje en el orden previo a la reordenación.

*tag_ord*: es la posición que el mensaje ocupa tras haberse ejecutado la reordenación.

Los resultados de las ejecuciones del *reducer* se vierten sobre un fichero de resultados que engloba todos los vuelos reordenados se parados por un salto de línea, y con los registros de cada trayectoria ordenados según el nuevo orden obtenido tras ejecutar la reordenación.

- `Mensaje`: clase que abstrae los mensajes ADS-B. Almacena tanto los campos asociados a éste como los valores de estos campos.

En la Figura 7.3 se presenta el mecanismo de reordenación efectuado por la implementación escalable que trata este epígrafe. En primer lugar, el *mapper* lee todos los registros de un fichero de entrada *csv* que contendrá todos los registros asociados a varios vuelos (no necesariamente agrupados ni ordenados). A través del valor del campo *leg* de

```
fr24-34250C-1517591521,1517591521,40.49247,-3.57694,0,2,0,0000,90,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.1,1517591521,74,74
fr24-34250C-1517591536,1517591536,40.49247,-3.57670,0,11,0,0000,90,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.12,1517591536,75,75
fr24-34250C-1517591547,1517591547,40.49250,-3.57639,0,16,0,0000,84,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.15,1517591547,76,76
fr24-34250C-1517591552,1517591552,40.49250,-3.57635,0,16,0,0000,81,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.15,1517591552,77,77
fr24-34250C-1517591562,1517591562,40.49251,-3.57636,0,16,0,0000,81,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.16,1517591562,78,78
fr24-34250C-1517591566,1517591566,40.49278,-3.57552,0,21,0,0000,61,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.23,1517591566,79,79
fr24-34250C-1517591582,1517591582,40.49310,-3.57473,0,20,0,0000,59,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.31,1517591582,80,80
fr24-34250C-1517591587,1517591587,40.49313,-3.57468,0,16,0,0000,53,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.31,1517591587,81,81
fr24-34250C-1517591602,1517591602,40.49344,-3.57463,0,0,0,0000,2,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.35,1517591602,82,82
fr24-34250C-1517591607,1517591607,40.49344,-3.57463,0,0,0,0000,2,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.35,1517591607,83,83
fr24-34250C-1517591612,1517591612,40.49345,-3.57463,0,0,0,0000,2,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.35,1517591612,84,84
fr24-34250C-1517591627,1517591627,40.49345,-3.57462,0,0,0,0000,2,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.35,1517591623,86,85
fr24-34250C-1517591618,1517591618,40.49344,-3.57463,0,0,0,0000,2,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.35,1517591628,85,86
fr24-34250C-1517591633,1517591633,40.49344,-3.57463,0,2,0,0000,2,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.35,1517591633,87,87
fr24-34250C-1517591637,1517591637,40.49344,-3.57463,0,2,0,0000,2,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.35,1517591637,88,88
fr24-34250C-1517591648,1517591648,40.49344,-3.57463,0,2,0,0000,2,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.35,1517591648,89,89
fr24-34250C-1517591653,1517591653,40.49344,-3.57463,0,2,0,0000,2,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.35,1517591653,90,90
fr24-34250C-1517591673,1517591673,40.49360,-3.57463,0,25,0,0000,2,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.37,1517591673,91,91
fr24-34250C-1517591688,1517591688,40.49668,-3.57466,0,70,0,0000,359,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,2.71,1517591688,92,92
fr24-34250C-1517591698,1517591698,40.50038,-3.57467,0,100,0,0000,359,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,3.12,1517591698,93,93
fr24-34250C-1517591709,1517591709,40.50671,-3.57472,0,135,0,3411,359,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,3.83,1517591709,94,94
fr24-34250C-1517591709,1517591709,40.50671,-3.57472,0,135,0,0000,359,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,3.83,1517591709,95,95
fr24-34250C-1517591729,1517591729,40.52084,-3.57458,2525,146,2112,3411,359,False,IBE04HT_34250C_1517590697_1517594767,2018-02-02,5.4,1517591729,96,96
fr24-34250C-1517591734,1517591734,40.52345,-3.57467,2675,147,2112,3411,358,False,IBE04HT_34250C_1517590697_1517594767,2018-02-02,5.69,1517591734,97,97
fr24-34250C-1517591740,1517591740,40.52769,-3.57488,2900,148,2176,3411,357,False,IBE04HT_34250C_1517590697_1517594767,2018-02-02,6.16,1517591740,98,98
osky-34250c-1517591740,1517591740,40.52840,-3.57490,2925,148,21,null,357,False,IBE04HT_34250C_1517590697_1517594767,2018-02-02,6.24,1517591740,99,99
fr24-34250C-1517591745,1517591745,40.53034,-3.57505,3025,148,2176,3411,357,False,IBE04HT_34250C_1517590697_1517594767,2018-02-02,6.46,1517591745,100,100
osky-34250c-1517591745,1517591745,40.53100,-3.57510,3075,149,21,null,358,False,IBE04HT_34250C_1517590697_1517594767,2018-02-02,6.53,1517591745,101,101
fr24-34250C-1517591750,1517591750,40.53430,-3.57513,3250,150,2176,3411,359,False,IBE04HT_34250C_1517590697_1517594767,2018-02-02,6.9,1517591750,102,102
osky-34250c-1517591750,1517591750,40.53500,-3.57510,3300,150,21,null,0,False,IBE04HT_34250C_1517590697_1517594767,2018-02-02,6.97,1517591750,103,103
osky-34250c-1517591754,1517591754,40.53690,-3.57510,3375,150,21,null,1,False,IBE04HT_34250C_1517590697_1517594767,2018-02-02,7.19,1517591754,104,104
osky-34250c-1517591755,1517591755,40.53890,-3.57500,3500,150,21,null,1,False,IBE04HT_34250C_1517590697_1517594767,2018-02-02,7.41,1517591755,105,105
```

Figura 7.4: Salida de la aplicación en *MapReduce*

cada mensaje, los registros quedan agrupados para posteriormente ejecutar el *reducer*. El método *reduce* recibe todos los registros asociados a un mismo valor de *leg* y los ordena ascendentemente por *timestamp* a través del método *reordenarMensajes()* de la clase *Interpreter*. A continuación, se efectúa el procesado de los registros, que consiste en su reordenación mediante el algoritmo *k-Opt* y la corrección de los *timestamps*. Se ejecutan tantas veces el *reducer* como distintos valores de *leg* haya en el fichero de entrada.

Posteriormente a la reordenación, se efectúa la corrección de las marcas temporales de los mensajes ADS-B. Esta funcionalidad se basa en el modelo matemático que se presenta en la Sección 5.5, que toma en consideración la posición antes y después de la reordenación de cada registro en la ruta, su valor de velocidad, la distancia recorrida desde el primer punto de la trayectoria (después de la reordenación), y la marca temporal.

La lógica de este modelo de corrección de *timestamps* se plasma en la función `corregirTimestamps()` de la clase `Interpreter`, que toma como parámetros

- el array de enteros resultado de la ejecución del método `runAlgorithm()` de la clase `KOpt`, con la que se lleva a cabo la reordenación y
- un array de `double` que asocia a la posición i-ésima la distancia recorrida por la aeronave al enviarse el i-ésimo mensaje ADS-B de la trayectoria (según la nueva ordenación).

La implementación replica la que se realizó para la aplicación que albergó el estudio comparativo, y que se expone en la Sección 5.5.

La Figura 7.4 muestra el contenido del fichero de salida del procedimiento. Obsérvese como el nuevo timestamp nuevo es distinto respecto del antiguo cuando el contenido de las columnas *tag_orig* (posición antes de la reordenación) y *tag_ord* (posición tras la

reordenación) difiere, y permanece igual en caso contrario. Esto se debe a que, en línea con lo expuesto en la Sección 5.5, los puntos cuya posición no cambia tras la reordenación se consideran correctos, por lo que su *timestamp* no se corrige, mientras que en los que no lo hacen se consideran erróneos o que necesitan corregir su marca temporal.

# Capítulo 8

# Conclusiones

## 8.1. Consideraciones finales

Este trabajo se ha enfocado a la resolución del *problema de la reordenación de los datos de seguimiento de una trayectoria*, que surge de la problemática asociada a la aplicación de las nuevas tecnologías ADS-B a los sistemas de *gestión del tráfico aéreo* o *Air Traffic Management* (ATM). La solución que ha propuesto este trabajo consiste en modelar el problema como una variante del *problema del viajante* (TSP) en la que las ciudades de origen y de destino son distintas. A su vez, esta consideración obliga a introducir ciertas suposiciones, como la de tomar como reordenación correcta de los mensajes ADS-B, aquella que haga mínima la longitud del camino que recorre cada uno de los puntos en orden.

Una vez alcanzado el capítulo final de este texto conviene echar la vista atrás hacia los objetivos iniciales del proyecto, que se presentaron al comienzo de esta memoria (ver Sección 1.2). El objetivo principal del proyecto [OB-1] impuso la realización de un estudio teórico del estado del arte (ver Capítulo 4), que obligatoriamente focalizaría sobre el problema del viajante, punto de partida de la propuesta que ambiciona este trabajo. El modelo teórico presentado que sustenta la reordenación de los mensajes ADS-B de una trayectoria, en línea con el subobjetivo [OB-1.1], se expone en la Subsección 5.4.1, donde se describe la técnica de las *ventanas de ejecución* que permite salvar el obstáculo que supone la alta complejidad computacional del problema del viajante. Pero para concluir la creación de este modelo es necesaria la selección de un método o algoritmo de resolución del TSP que se ejecute, y que cuyos resultados en términos de longitud de la ruta obtenida y de tiempo de ejecución sean lo más bajos posibles. Así, se motiva el desarrollo de una aplicación en `R` que implemente un estudio comparativo de un buen número de técnicas con el fin de seleccionar la más adecuada de ellas. Los fundamentos de esta aplicación se exponen en el Capítulo 5 y el análisis de estas técnicas en el Capítulo 6, obteniendo como conclusión principal la primacía del algoritmo *2-Opt*.

El paso siguiente que da este trabajo, en línea con el [OB-1] consiste en el desarrollo de una implementación escalable en *MapReduce* que implemente el modelo teórico anterior, de manera que sea paralelizable, condición necesaria para poder ser posteriormente integrada a la lógica interna de un sistema gestor del tráfico aéreo. El esfuerzo llevado a cabo en esta tarea se expone en el Capítulo 7.

Como segundo de los objetivos que plantea este Trabajo Fin de Grado [OB-2], se aborda la implementación de un segundo modelo que lleve a cabo el refinamiento o corrección de las marcas temporales de los mensajes ADS-B de una trayectoria recientemente reordenada. La Sección 5.5 presenta la propuesta que hace este trabajo, así como su puesta en práctica en la aplicación en `R` que expone el Capítulo 5.

Por último, este trabajo aborda la tarea de elaborar un compendio biográfico sobre el problema del viajante, figura central sobre la que se asienta la propuesta de solución del problema de la reordenación de los mensajes ADS-B de un vuelo que trata este trabajo. El Capítulo 4 lo presenta, incluyendo los fundamentos teóricos, así como algunos de los algoritmos más destacados. En línea con el subobjetivo [OB-3.2] se analiza la mecánica de las heurísticas de mejora local más destacadas.

## 8.2. Aprendizaje personal

En este epígrafe recojo unas breves notas sobre la experiencia personal que ha supuesto para mi la realización de este Trabajo Fin de Grado:

- Llevar a cabo este proyecto ha supuesto para mi una oportunidad para poner en práctica todos los conocimientos que he adquirido a lo largo de la carrera. Especialmente el obtenido en las asignaturas que tratan la *Ingeniería del Software* así como la programación en el marco de un proyecto.

- Por otra parte, ha resultado en una oportunidad personal para revisar ciertos lenguajes de programación que he tenido la oportunidad de aprender a lo largo de mi paso por la titulación. Algunos de ellos los he utilizado en repetidas ocasiones como *Java*, mientras que otros los he requerido menos a lo largo de la carrera, como `R`.

- He aprendido un nuevo modelo de programación, gracias al trabajo con *MapReduce* desde el punto de vista teórico, como práctico a través de la implementación llevada a cabo sobre este esquema.

- Por último, la redacción de esta memoria me ha ayudado a mejorar mis habilidades con el lenguaje para procesamiento de textos científicos LaTeX.

## 8.3. Líneas de trabajo futuro

Para poner punto y final a esta memoria, quedan señaladas a continuación una serie de aspectos que han quedado fuera del alcance de este Trabajo Fin de Grado y que definen las futuras actividades a llevar a cabo sobre el proyecto.

- Integración completa de la implementación escalable sobre la plataforma *bigdata* de gestión del tráfico aéreo objetivo del proyecto AIRPORTS, a fin de que pase a formar parte de su lógica interna. En este sentido, la aplicación haría su función durante el procesado de los datos de una trayectoria, donde en primer lugar se ejecutaría una reordenación mediante el algoritmo *2-Opt* para posteriormente corregir las marcas temporales de los mensajes.

- Desarrollo completo del componente de acceso a datos de la implementación escalable, así como de la aplicación que alberga el estudio comparativo de los métodos de resolución (ver Capítulo 5). En la actualidad figura un acceso a datos rudimentario que consiste en la lectura de uno o varios ficheros *csv* de una carpeta prefijada. Los pasos siguientes a llevar a cabo en esta materia consisten en la implementación de un acceso a las bases de datos de las distintas fuentes de datos *en crudo*.

- Sobre el punto anterior, la aplicación del Capitulo 5 podría albergar el seguimiento de los vuelos en tiempo real, una vez tenga implementado el acceso a las bases de datos de distintas fuentes como *OpenSky* o *Frambuesa*. De esta manera, sería posible la monitorización por parte del usuario a través del *dashboard* de las distintas trayectorias en directo.

- Por su parte, el acceso a datos sobre la implementación escalable permitiría el desarrollo de una nueva pieza de la funcionalidad que jugaría un papel fundamental: la corrección en caliente de las desórdenes que se produzcan en los mensajes ADS-B nada más son recibidos por una de las fuentes de datos. Para esto, el sistema podría ejecutar reordenaciones periódicas sobre los últimos mensajes (utilizando la técnica de ventanas para hacer más eficientes las ejecuciones) de la trayectoria en curso, corrigiendo posteriormente las marcas temporales en caso de producirse un error.

- En cuanto al modelo teórico de que se plantea para la reordenación de los mensajes ADS-B de una trayectoria, el siguiente paso a realizar podría consistir en aumentar su complejidad añadiendo el valor de altitud de la aeronave a la hora de calcular la distancia entre puntos. Esto aumentaría la eficacia del método así como un menor riesgo de error en los casos en que se requieren complejas maniobras de aterrizaje del piloto, y que a veces incluyen rodear el aeropuerto repetidas veces, para los que el modelo actual no se adapta correctamente.

- Por otro lado, el modelo de corrección de *timestamps* puede ser mejorado añadiendo nuevos parámetros al modelo como la marca temporal de más puntos o un planteamiento más complejo del modelo que plantee la corrección de los *timestamps* como

un problema de optimización. En este sentido, podrían llevarse a cabo técnicas de resolución comunes para este tipo de problemas como la programación lineal.

# Bibliografía


J. Ahumada. *Versión en español de R for beginners*. Emmanuel Paradis, 2002.

A. Alonso Isla, P.C. Alvarez-Esteban, A. Bregón, F. Díaz, I. García Miranda, P. Gordaliza, y M. Martínez-Prieto. Airports: Análisis de eficiencia operacional basado en trayectorias de vuelo. *JISBD*, 2018.

P.C. Álvarez Esteban, A. Bregón, F. Díaz, I. García Miranda, y M. Martínez Prieto. Towards a scalable architecture for flight data management. *DATA 2017 - 6th International Conference on Data Science, Technology and Applications*, pages 263–268, 2017.

G. Babin, S. Deneault, y G. Laporte. Improvements to the or-opt heuristic for the symmetric traveling salesman problem. *The Journal of Operational Research Society. Vol. 58, No. 3*, pages 402–407, 2007.

B. Chandra, H. Karloff, y C. Tovey. New results on the old k-opt algorithm for the traveling salesman problem. *SIAM Journal on Computing*, 1999.

W.J. Cook. *In Pursuit of the Traveling Salesman*. Princeton University Press, 2012.

Jeffrey Dean y Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

M. Englert, H. Röglin, y B. Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the tsp. *SODA '07 Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1295–1304, 2007.

Eurocontrol. Flight movements and service units 2017-2023. Technical report, Eurocontrol, Febrero 2017. Disponible en :https://www.eurocontrol.int/sites/default/files/content/documents/official-documents/forecasts/seven-year-flights-service-units-forecast-2017-2023-Feb2017.pdf.

A. González y S. González. *Versión en español de An introduction to R*. R Development Core Team, 2000.

Y. Grushka-Cockayne y B. De Reyck. Towards a single european sky. *Interfaces*, 39: 400–414, 2009.

G. Gutin y A. P. Punnen. *The Traveling Salesman Problem and its variations*. Springer, 2007.

B. Juricic, T. Bucak, y I. Francetic. Automatic dependent surveillance (ads). *Promet - Traffic - Traffico*, 14:111–115, 2002.

B. Korte y J. Vygen. *Combinatorial Optimization, Theory and Algorithms*. Springer, 2006.

G. Laporte. The traveling salesman problem: An overview of exact and approximated algorithms. *European Journal of Operational Research*, 59:231–247, 1992.

E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, y D.B. Shmoys. *The Traveling Salesman Problem*. Wiley, 1992.

M.A. Martínez Prieto, Bregon A., I. García Miranda, Álvarez Esteban, P.C. Díaz F., y D. Scarlatti. Integrating flight-related information into a (big) data lake. *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*, 2017.

# Apéndice A

# Contenido del CD y manuales de las aplicaciones

## A.1. Contenido del CD

El disco en formato CD-R anexo a la presente memoria, adjunta las dos aplicaciones objeto de desarrollo del proyecto.

- Por un lado, se incluye la aplicación expuesta en el Capítulo 5, que alberga el estudio comparativo de las técnicas de resolución del problema del viajante aplicadas a la reordenación de los datos de seguimiento de trayectorias de aeronaves. El contenido de esta aplicación se encuentra en el archivo comprimido ProyectoR.zip.
- Por otro lado, se encuentran los ficheros asociados a la implementación escalable en *MapReduce* del modelo de reordenación y posterior corrección del *timestamp* de los mensajes planteado en esta memoria. Todos los archivos de esta parte se encuentran en MapReduceTrayectoriasKOpt.zip

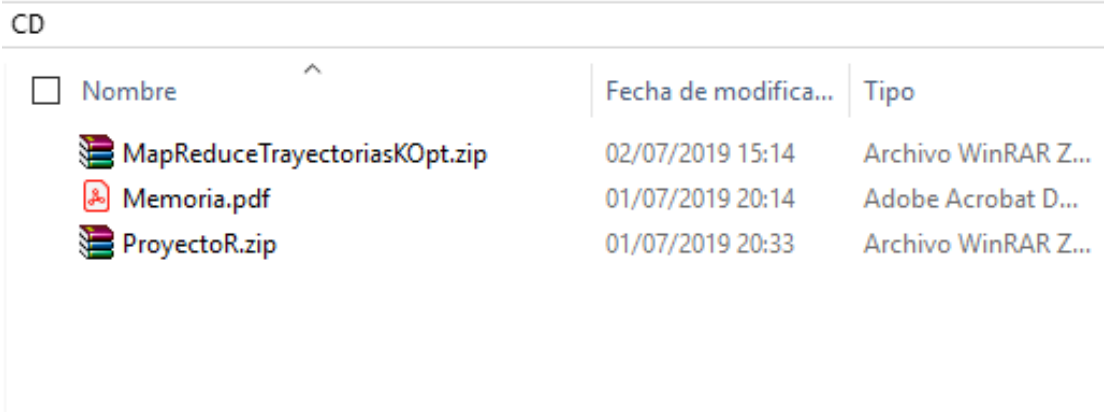



Figura A.1: Contenido del CD

El contenido del CD se ilustra en la Figura A.1, donde junto al contenido de los dos programas anteriores se incluye la presente memoria en formato pdf.

Por un lado, en el fichero ProyectoR.zip se encuentran los ficheros de la aplicación sobre R, descrita en el Capítulo 5 de la presente memoria.

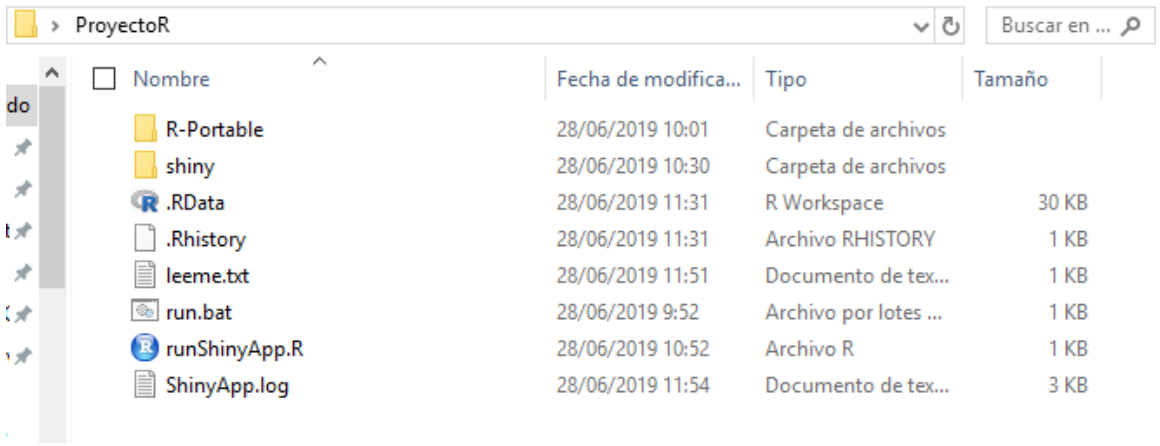



Figura A.2: Contenido de la carpeta ProyectoR

La Figura A.2 muestra el contenido de la carpeta del proyecto, la cual contiene otras dos carpetas. Por un lado, la carpeta R-Portable, que incluye los ficheros asociados al lenguaje de programación R, y que permiten la portabilidad de la aplicación sin necesidad de ejecutarse sobre un sistema con R instalado. Por otro lado, en shiny se incluyen los ficheros que componen la aplicación (ver Figura A.3).

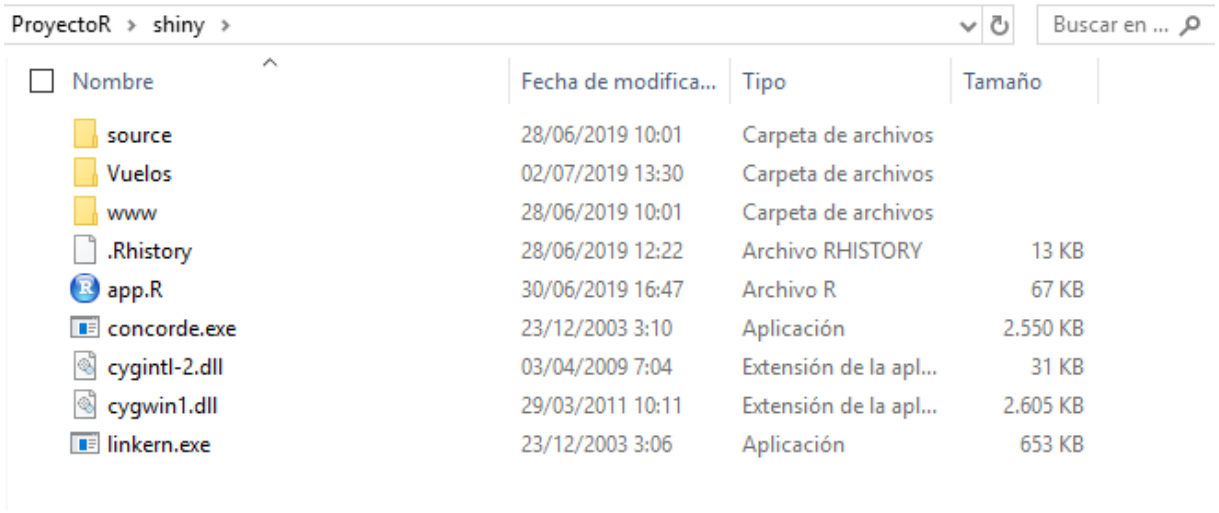



Figura A.3: Contenido de la carpeta shiny

La carpeta Vuelos contiene los ficheros csv con los vuelos a los que puede acceder la

aplicación. Por su parte, las carpetas www y source contienen ficheros secundarios de la aplicación. El código de la aplicación se encuentra en el fichero app.R de shiny.

Por su parte, en el fichero MapReduceTrayectoriasKOpt.zip se encuentra la estructura de directorios de la implementación escalable expuesta en el Capítulo 5. La Figura A.4 muestra su contenido.

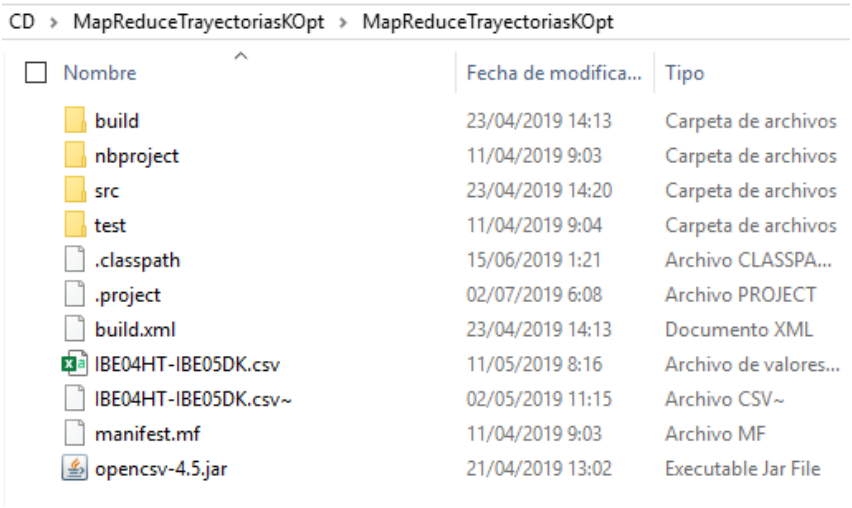



Figura A.4: Estructura de directorios de la implementación escalable

Las clases de la aplicación se encuentran en la carpeta src, donde está contenido todo el código de la aplicación. El resto de carpetas y ficheros son secundarios para el funcionamiento de la aplicación. A su vez, el archivo IBE04HT-IBE05DK.csv es un ejemplo de fichero que puede tomarse como entrada para la aplicación.

A lo largo de la siguiente sección se exponen los manuales de ambas aplicaciones, incluyendo los manuales de instalación, con las instrucciones básicas de puesta a punto de cada una de ellas.

## A.2. Manuales

En esta sección se incluyen los manuales de usuario y de instalación de las dos aplicaciones presentadas a lo largo de esta memoria. En primer lugar se presentan los manuales de instalación de cada uno de los programas. Posteriormente se presentan los manuales de uso, para los usuarios finales.

### A.2.1. Manuales de instalación

A continuación se incluyen las instrucciones básicas para efectuar la instalación de los programas incluidos en el CD. Primero se expone el manual relativo a la aplicación que trata el Capítulo 5, y finalmente la implementación escalable del Capítulo 7.

## Aplicación que alberga el estudio comparativo

La aplicación, desarrollada para estudiar la adecuación de algunos de los más importantes algoritmos de resolución del TSP aplicados a la reordenación de los datos de seguimiento de una trayectoria, no está pensada para ser de uso público, sino restringido al personal del presente proyecto, por lo que la aplicación es de escritorio para ser instalada sobre los ordenadores personales de cada miembro del proyecto.

Los requisitos previos para poder ejecutarse la aplicación son:

- Sistema operativo: Windows Vista,7, 8 o 10.
- Memoria RAM: un mínimo de 1 GB

Para ejecutar el programa, el usuario debe entrar en ProyectoR y hacer doble click sobre el fichero *run.bat*, que no es más que el ejecutable de la aplicación. Tras esto, se ejecutará un terminal de Windows durante unos segundos (ver Figura A.5)

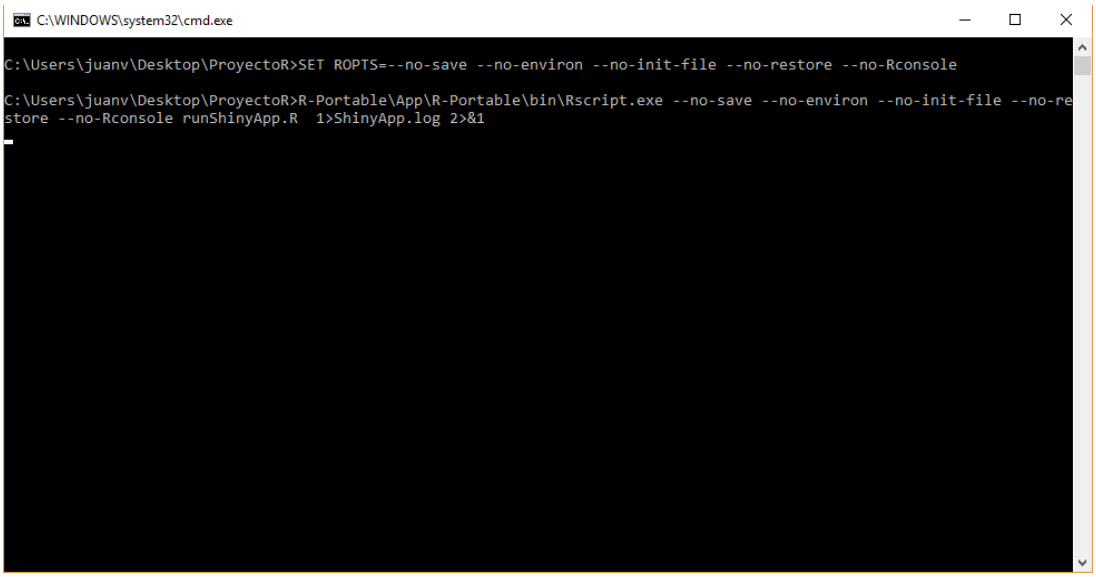



Figura A.5: Terminal desplegado tras ejecutar el fichero *run.bat*

Tras una breve espera se abre el navegador fijado en el sistema como predeterminado con la pantalla inicial de la aplicación, tal y como se muestra en la Figura A.6.

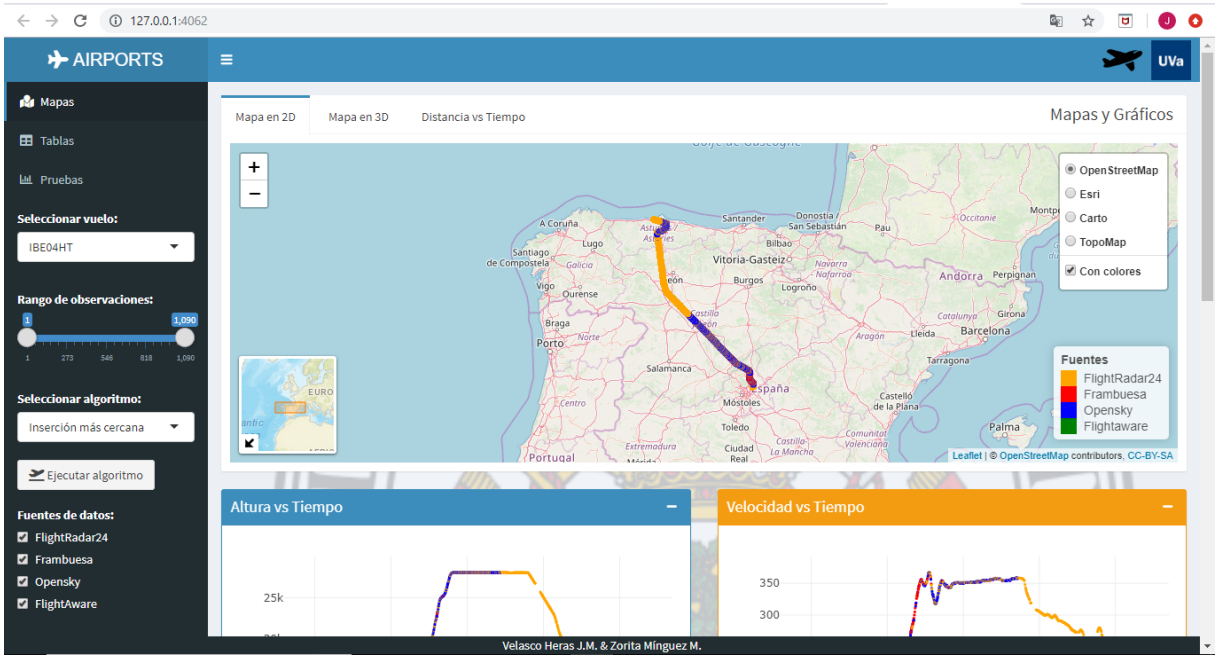



Figura A.6: Pantalla de inicio de la aplicación

Los datos de vuelos con los que trabaja la aplicación proceden de los ficheros csv contenidos en la carpeta ProyectoR/shiny/Vuelos del CD. Al comenzar, la aplicación lee el primero de los ficheros, y va accediendo al resto en función de las necesidades del usuario.

Para terminar la ejecución de la aplicación basta con cerrar la pestaña del navegador en que se desplegó la aplicación.

## Implementación escalable

Para poder ejecutar la implementación escalable se han identificado los siguientes requisitos:

- Disponer de Cloudera, que integra las librerías de Apache Hadoop
- Java JDK 1.7 o superior
- IDE Eclipse Luna

El sistema operativo Cloudera se puede adquirir desde su sitio web oficial `https://es.cloudera.com/` (ver Figura A.7).

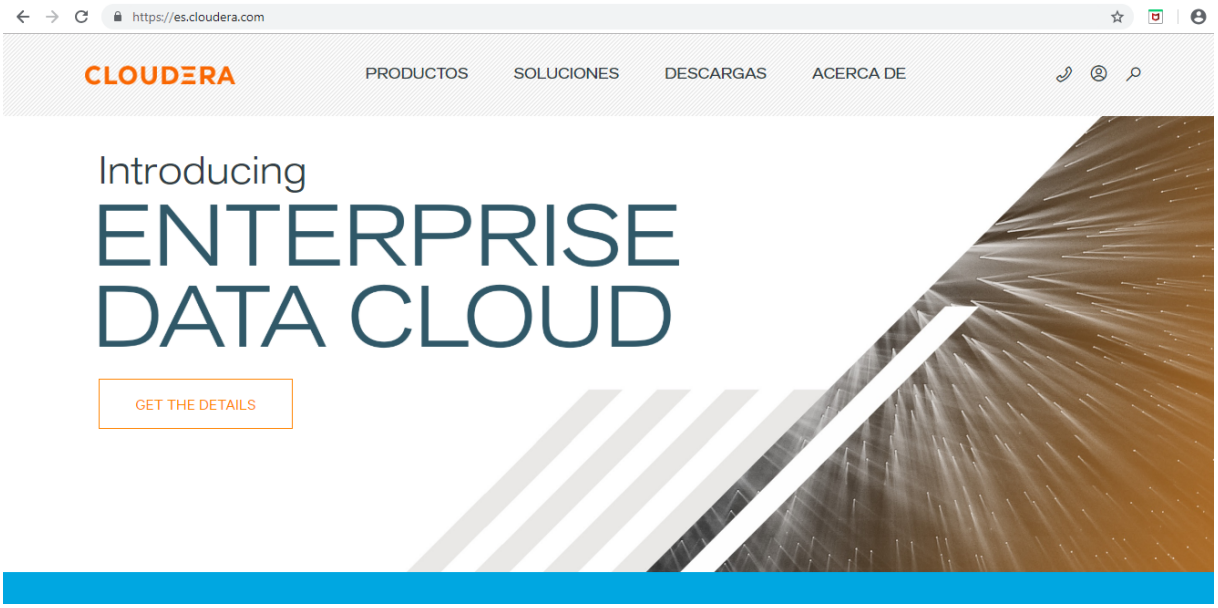



Figura A.7: Página principal del sitio web de Cloudera: `https://es.cloudera.com/`

Los ficheros de la implementación escalable, expuesta en el Capítulo 7 de la presente memoria, se incluyen en el archivo MapReduceTrayectoriasKOpt.zip. Este comprimido puede descomprimirse sobre la carpeta del sistema deseada. Para trabajar con la aplicación se requiere importar el proyecto sobre el entorno Eclipse y fijar las dependencias con las librerías de Hadoop necesarias.

### A.2.2. Manuales de usuario

A continuación se incluyen los manuales de uso de las aplicaciones que abarca este Trabajo Fin de Grado.

#### Aplicación que alberga el estudio comparativo

En esta sección se presenta un breve manual que describe los procedimientos para llevar a cabo las funcionalidades principales de la aplicación. El objetivo fundamental de la misma es estudiar el comportamiento de las implementaciones de los algoritmos de resolución asociados al problema del viajante (TSP) presentes en las librerías `stats` (*Simulated Annealing*) y `TSP` (*2-Opt*, *vecinos más próximos*, *vecinos más próximos repetitivo*, *inserción más cercana*, *inserción más barata*, *inserción más lejana*, *inserción arbitraria*, *Lin-Kernighan* y *Concorde*), en términos de longitud de la ruta encontrada y tiempo de ejecución, al ser aplicados al problema de reordenación de trayectorias a partir de datos reales de vuelos.

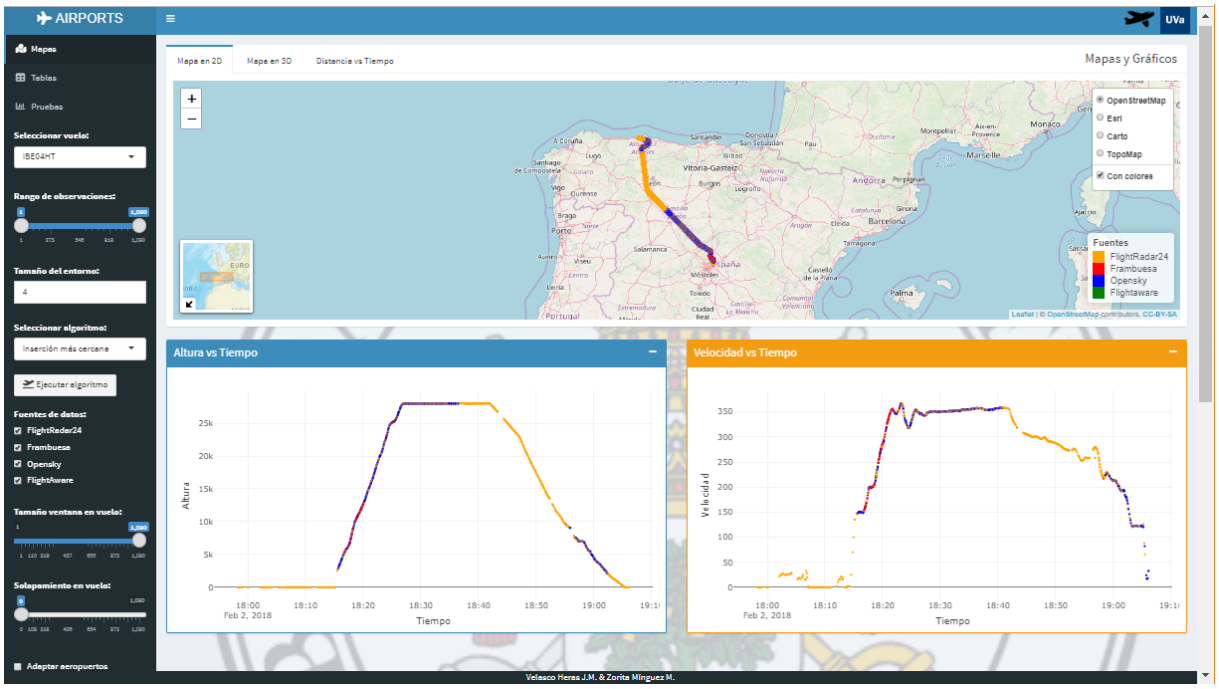



Figura A.8: Página de inicio

La Figura A.8 ilustra la página inicial de la aplicación de R. La estructura de la página se mantiene en todos sus escenarios de navegación: una cabecera, un menú lateral, una parte central donde se presentan los diferentes componentes del *dashboard* y un *pie de página*.

### Controles sobre la interfaz

Se listan a continuación los controles de la interfaz de usuario. Todos ellos se encuentran bien en el menú lateral, bien en la parte central de la pantalla.

La Figura A.9 ilustra la disposición de los controles sobre el menú lateral. Estos controles son los siguientes:

- *Widgets*: cambian la vista central de la interfaz. Hay tres vistas diferentes:
  - *Mapas*: presenta los diferentes gráficos del *dashboard* asociados al vuelo elegido
  - *Tabla*: presenta la tabla con los datos de los mensajes ADS-B que componen el vuelo. Los datos de un vuelo se leen de un fichero *csv* que contiene cada uno datos asociados a cada mensaje ADS-B y la primera fila contiene el nombre de los campos. El programa lee todas las columnas del fichero y las representa en la interfaz como columnas de la tabla que ocupa la parte central.
  - *Pruebas*: presenta un formulario que permite la introducción de los paráme-

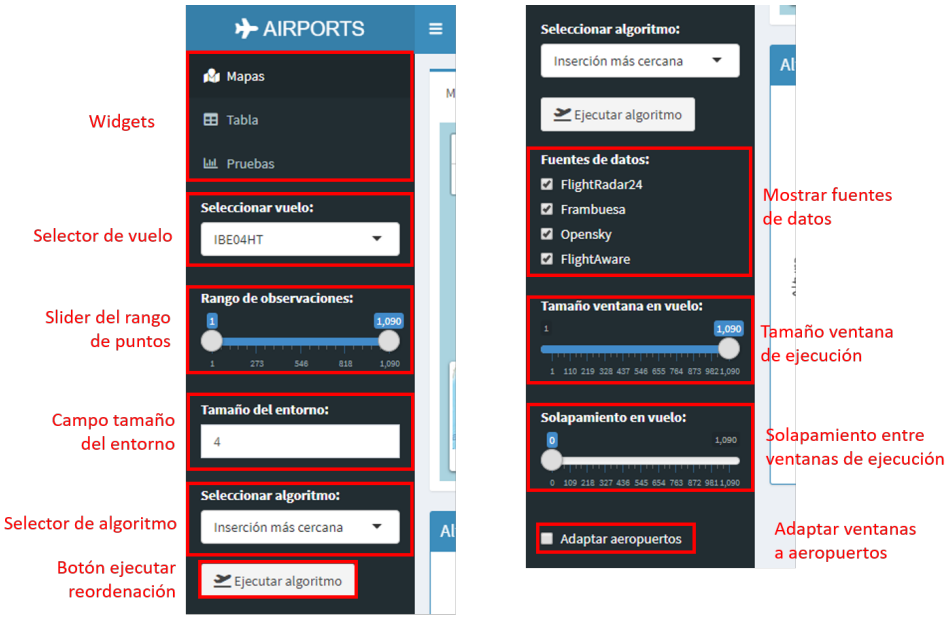



Figura A.9: Controles del menú lateral

tros asociados a la realización de baterías de pruebas consistentes en probar múltiples reordenaciones sobre varios vuelos sobre los que se aplican diferentes algoritmos. Los resultados de estas pruebas se presentan en tablas [CAR-2].

- *Selector de vuelo*: permite al usuario elegir el vuelo sobre el que trabajar [CAR]. Al cambiar de vuelo cambian todos los componentes que forman el *dashboard*, que pasan a ilustrar información asociada al nuevo vuelo elegido. Los vuelos presentes en el selector son los títulos de todos los ficheros *csv* presentes en la carpeta de la aplicación denominada *Vuelos* [RF-05].
- *Slider del rango de puntos*: permite al usuario acotar el rango de puntos del vuelo sobre los que trabajar. Los puntos descartados no se muestran en el *dashboard* ni se consideran al ejecutar una reordenación de la trayectoria [RF-06].
- *Campo tamaño del entorno*: los efectos del valor del parámetro introducido en este campo de texto se visibilizan en la pestaña *Distancia vs Tiempo* y tras ejecutar una reordenación. Este campo permite determinar el número de puntos anteriores y posteriores a cada punto desordenado (aquel cuyo orden en la trayectoria cambia tras ejecutar la reordenación) que se tomarán en consideración para determinar el nuevo *timestamp* asociado a estos puntos [RF-15].
- *Selector de algoritmo*: permite al usuario seleccionar el algoritmo a aplicar en la reordenación de la trayectoria elegida en el *selector de vuelo*, tras hacer *click* en el *botón ejecutar reordenación* [RF-01].

  Al seleccionar el algoritmo *Simulated Annealing* se despliegan controles adicionales sobre el menú lateral que permiten al usuario configurar los parámetros de ejecución

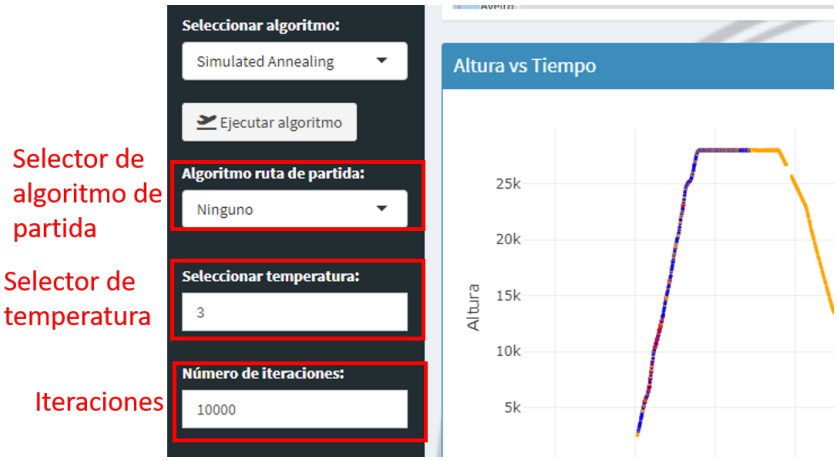



Figura A.10: Controles adicionales al seleccionar el *Simulated Annealing* como algoritmo en el *Selector de algoritmo*

especiales de este algoritmo. Estos controles se ilustran en la Figura A.10 y son los siguientes:

- *Selector de algoritmo de partida*: por su bajo rendimiento, se da la opción al usuario de apoyar el rendimiento del *Simulated Annealing* con una reordenación previa que obtenga una ruta que el *Simulated Annealing* tome como ruta de partida para su ejecución.
- *Campo de temperatura*: Permite fijar el valor inicial de temperatura para la ejecución del algoritmo. Como ya se ha comentado, el *Simulated Annealing* va probando modificaciones sobre la ruta actual para obtener nuevas rutas más cortas. El alcance de estas modificaciones será mayor cuanto más alto sea el valor de temperatura asociado. A lo largo de la ejecución del algoritmo este valor de temperatura va decrementando, lo que indica que las modificaciones que prueba el algoritmo van siendo cada vez más pequeñas.
- *Campo de iteraciones*: en el caso del *Simulated Annealing* cuya implementación es tomada de la librería `stats`, se permite indicar el número de iteraciones que se pretende que realice el algoritmo. Es claro que a mayor valor de este campo mayor se espera que sea la mejora sobre la ruta de partida al haber evaluado un mayor número de rutas.

- *Botón ejecutar reordenación*: permite al usuario iniciar la ejecución de una reordenación sobre el vuelo elegido en *selector de vuelo* aplicando el algoritmo seleccionado en *selector de algoritmo*. Al finalizar la ejecución, los componentes del *dashboard* se recargan para mostrar los resultados de la reordenación [RF-01].
- *Mostrar fuentes de datos*: con este grupo de *checkboxes* el usuario puede filtrar los datos del vuelo elegido en *selector de vuelo* por las distintas fuentes de captura de los mensajes ADS-B [RF-07].

- *Tamaño ventana de ejecución*: permite al usuario elegir el tamaño máximo de las ventanas de ejecución al ejecutar una reordenación a través del *botón ejecutar reordenación*, haciendo uso del procedimiento de reordenación descrito en la Sección 5.4.1 [RF-01].
- *Solapamiento entre ventanas de ejecución*: permite seleccionar el número de puntos que compartirán dos ventanas de ejecución consecutivas al ejecutar una reordenación a través del *botón ejecutar reordenación* [RF-01].
- *Adaptar ventanas a aeropuertos*: *checkbox* que permite configurar dos tamaños diferentes de ventanas al ejecutar una reordenación a través del *botón ejecutar reordenación*, de manera que el tamaño de la ventana sea diferente en las partes iniciales y finales del vuelo (cerca de un aeródromo) con respecto a la parte central (aeronave en vuelo) [RF-01].

  El *checkbox* despliega nuevos componentes sobre la interfaz que se muestran en la Figura A.11. Estos componentes son los siguientes:

  - *Campo de altura*: permite al usuario introducir un valor de altura como parámetro discriminador entre los puntos sobre los que se aplica un tipo u otro de ventana. El nuevo tipo de ventana se aplicará desde el primer punto hasta el primer punto que sobrepase el valor de altura introducido, y desde el último punto hasta el siguiente punto hacia atrás que sobrepase el mismo valor. En el resto de puntos, el tipo de ventana considerada será la dada por el tamaño introducido en el slider *tamaño ventana de ejecución* y con solapamiento dado en el slider *solapamiento entre ventanas de ejecución*.
  - *Slider de separación de ventanas*: permite fijar manualmente los puntos sobre los que se aplicará un tipo de ventana u otro. A los comprendidos por el slider se le aplicará el tamaño de ventana introducido en el slider *tamaño ventana de ejecución* con el solapamiento introducido en *solapamiento entre ventanas de ejecución*. Al resto de puntos se les aplicarán ventanas de tamaño el valor introducido en *tamaño de ventana en aeropuertos*, con el valor de solapamiento introducido en *solapamiento entre ventanas en aeropuertos*.
  - *Tamaño de ventana en aeropuertos*: permite al usuario configurar el tamaño máximo de las ventanas de ejecución sobre aeropuertos.
  - *Solapamiento entre ventanas en aeropuertos*: permite al usuario fijar el número de puntos compartidos por dos ventanas de ejecución consecutivas sobre aeropuertos.

A continuación se desglosan los controles presentes en cada uno de los tres *widgets* que permiten al usuario el control de la parte central de la pantalla.

- En *Mapas*: los controles situados en la parte central al seleccionar este *widget* tienen la finalidad de permitir al usuario interaccionar con cada uno de los componentes del


Figura A.11: Controles adicionales al hacer click sobre el *checkbox Adaptar ventanas a aeropuertos*

*dashboard* con el fin de obtener información específica vinculada a la representación.

Los controles asociados a esta pantalla son los siguientes:

- *Pestañas*: permiten al usuario cambiar el contenido mostrado en el componente más importante de la parte central, entre tres gráficos disponibles con información del vuelo seleccionado en *selector de vuelo*. Estas tres gráficas son: un mapa que representa las coordenadas geográficas asociadas a cada uno de los mensajes ADS-B del vuelo, una gráfica tridimensional que enfrenta las coordenadas (longitud y latitud) frente a la altura, y una última gráfica que enfrenta la distancia recorrida por la aeronave frente al *timestamp* de cada mensaje, de utilidad para observar los resultados del proceso de corrección del *timestamp* de los mensajes desordenados una vez ha sido ejecutada una reordenación. La Figura A.13 ilustra los controles asociados a las distintas pestañas de la componente.
    - En el mapa (opción "Mapa en 2D"), los controles se orientan a la navegabilidad a través del mapa, permitiendo al usuario controlar el enfoque o la posición del mapa a visualizar. Los controles *cambiar fondo del mapa* y *visualizar fuentes* permiten cambiar el contenido del mapa. El primero cambia el fondo del mapa, pudiendo optar el usuario por cuatro modos distintos en función de sus preferencias. Por su parte, el control *visualizar fuentes* permite dar color o quitárselo a los puntos representando los distintos mensajes ADS-B [RF-07]. Al accionar el control (por defecto accionado) cada mensaje se muestra de un color que indica la fuente de datos de la que procede. En caso de no accionarse, los mensajes se muestran de color gris si no se ha ejecutado ninguna reordenación, o gris y verde en

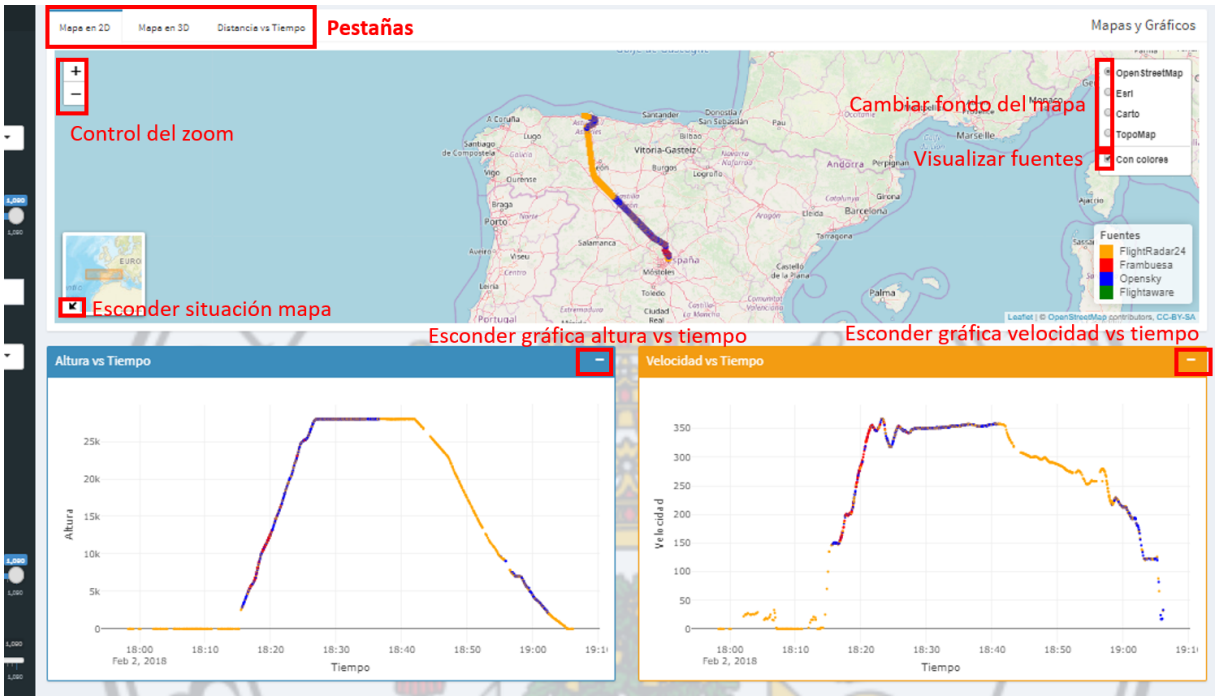



Figura A.12: Controles en la parte central con *widget Mapas* seleccionado

caso de haberse realizado esta operación.

- En la gráfica tridimensional (opción "Mapa en 3D") se permite al usuario mostrar la ordenación de los mensajes ADS-B (representados como puntos), a través de una línea que los une en función de su ordenación temporal. En caso de haber ejecutado una reordenación, el usuario podrá seleccionar la opción de visualizar tanto la reordenación como la ordenación original de los mensajes.
- En la gráfica de distancia recorrida contra el *timestamp* (opción "Distancia vs Tiempo") el usuario puede cambiar la coloración de los puntos. Esta gráfica solo aparece tras haber ejecutado una reordenación, pues el *timestamp* de los puntos cuyo orden temporal ha cambiado se ha corregido para adaptarlo al nuevo orden. Por defecto se muestran en verde aquellos puntos cuyo *timestamp* ha sido corregido, y en gris aquellos en que este dato ha permanecido estable, debido a que su orden no ha cambiado. Al accionar el control *cambiar coloración* los puntos se colorean en función de su altura asociada con un gradiente de colores.

- Controles para esconder las gráficas de altura vs. tiempo y velocidad vs. tiempo.

- En *Tabla*, los controles están destinados a cambiar las filas de la tabla que se muestran en pantalla [RF-09, RF-10]. Los controles de esta pantalla se ilustran en la Figura A.14, y son los siguientes:

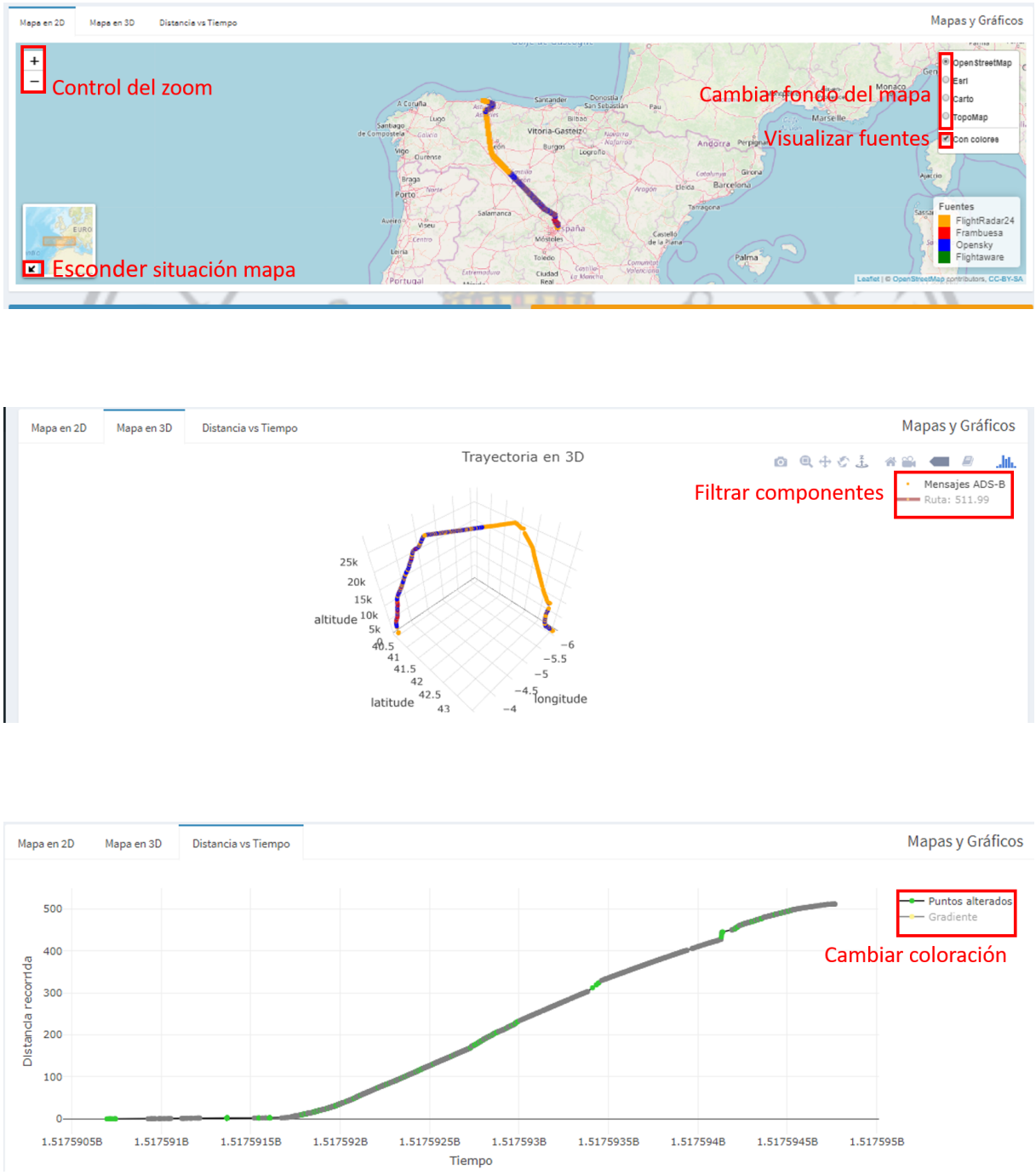



Figura A.13: Gráficas mostradas al seleccionar cada una de las pestañas del *dashboard*

- *Selector número de filas*: permite aumentar o disminuir el número de filas de la tabla que se muestran por pantalla.
- *Campo de filtrado*: permite realizar búsquedas sobre la tabla. El usuario introduce un valor y el sistema elimina aquellas filas en que ninguno de los campos


Figura A.14: Controles en la parte central con *widget Tabla* seleccionado

contengan o se identifiquen con ese valor, de modo que el usuario puede filtrar aquellas filas que no sean de su interés.

- *Control reordenación por campo*: el usuario puede hacer click sobre el nombre de cualquiera de las columnas de la tabla para reordenar las filas en base al valor de ese campo. Para invertir la ordenación (pasar de ser ascendente a descendente o viceversa), basta hacer de nuevo click sobre el nombre de la columna.

- *Navegación entre registros*: cuando no se muestran todas las filas de la tabla, y ésta queda dividida en "páginas", el usuario puede cambiar de una a otra a través de este control. Estas páginas están indexadas de modo que el usuario podrá seleccionar el número de página que desee para que se muestre por pantalla.

- En *Pruebas* se muestra un formulario que permite al usuario seleccionar los parámetros asociados a la prueba a ejecutar [CAR-2]. Los controles de esta parte se muestran en la Figura A.15 y son los siguientes:

  - *Selector múltiple de vuelos*: permite al usuario seleccionar los vuelos a considerar para la prueba. Mínimo uno [RF-13].

  - *Selector múltiple de algoritmos*: permite al usuario elegir los algoritmos que se aplicarán para reordenar los vuelos seleccionados en el *selector múltiple de vuelos*. Cada algoritmo se aplicará una vez a cada vuelo. Al menos un algoritmo

debe ser seleccionado para poder ejecutar la prueba [RF-14].

- *Opciones de ejecución*: parámetro principal de la prueba. Permite al usuario elegir el modo de ejecución a aplicar [RF-12]. El usuario podrá elegir entre una ordenación utilizando la técnica *divide y vencerás* de reordenación presentada en la Subsección 5.4.1), de tamaño fijo o de dos tamaños diferentes para adaptarse a las zonas aeroportuarias del trayecto, o por el contrario, ejecutar la reordenación sin aplicar esta técnica. Si esta última opción no es la elegida, se muestran controles adicionales para configurar los distintos parámetros asociados a las ventanas. Estos controles se muestran en la Figura A.16. Son los siguientes:
  - *Campo tamaño ventana en prueba*: análogo al control *tamaño ventana ejecución* del menú lateral.
  - *Slider solapamiento en prueba*: análogo al control *solapamiento entre ventanas de ejecución* del menú lateral.
  - *Campo altura en prueba*: análogo al control *campo de altura* del menú lateral.
  - *Campo tamaño ventana en vuelo*: análogo al control *tamaño ventana de ejecución* del menú lateral al accionar el *checkbox adaptar ventanas a aeropuertos*.
  - *Slider solapamiento en vuelo*:análogo al control *solapamiento entre ventanas de ejecución* del menú lateral al accionar el *checkbox adaptar ventanas a aeropuertos*.
  - *Campo tamaño ventana en aeropuerto*: análogo al control *tamaño de ventana en aeropuertos* del menú lateral.
  - *Slider solapamiento en aeropuerto*: análogo al control *solapamiento entre ventanas en aeropuertos* del menú lateral.
- *Botón ejecución de la prueba*: permite al usuario iniciar la batería de pruebas con los parámetros que haya elegido [RF-11]. Al menos debe haberse indicado un vuelo en *selector múltiple de vuelos* y un algoritmo en *selector múltiple de algoritmos* para poder ejecutarse la prueba.
- *Pestaña de resultados*: permite al usuario navegar por las diferentes tablas de resultados mostradas tras ejecutar una batería de pruebas.
- *Botón descargar resultados*: permite al usuario descargar el contenido de la tabla seleccionada en el control *Pestaña de resultados* en formato *csv* [RF-16].


Figura A.15: Controles en la parte central con *widget Pruebas* seleccionado


Figura A.16: Controles adicionales al seleccionar la opción “Utilizar ventanas de un mismo tamaño”, en la parte izquierda, o “Utilizar ventanas de distinto tamaño para aeropuertos”, en la parte derecha, en el control *opciones de ejecución*

### Principales funciones realizables

En esta sección se recogen las funciones más importantes de la aplicación describiéndose el conjunto de acciones realizadas por el usuario para poder llevarse a cabo. La aplicación se construye para estudiar el comportamiento de los distintos algoritmos de resolución del TSP aplicados a la reordenación de trayectorias. Es por tanto la funcionalidad principal de la aplicación la de ejecutar reordenaciones sobre vuelos que seleccione el usuario y utilizando los algoritmos a su elección. Además de la posibilidad de ejecutar una única reordenación sobre una ruta concreta y mostrar sus resultados a través de las diferentes gráficas y mapas que componen el *dashboard*, el usuario puede ejecutar una batería de pruebas que incluye la posibilidad de ejecutar varios algoritmos sobre uno o varios vuelos determinados, obteniendo resultados que permiten comparar el rendimiento de todas las reordenaciones. Así, se describen a continuación estas dos funcionalidades principales de la aplicación.

#### Ejecutar una reordenación [CAR-1.1]

A través de la aplicación, el usuario puede ejecutar una reordenación sobre un vuelo elegido a través de los controles presentes en el menú lateral del *widget Mapas*. La lista de pasos generales a seguir para llevar a cabo esta función es la siguiente:

1. Accionar la opción *Mapas* en el control *Widgets* del menú lateral.
2. Seleccionar el vuelo sobre el que ejecutar la reordenación de entre la lista de vuelos disponibles, en el control *Selector de vuelo*.
3. Seleccionar el algoritmo que se aplicará al ejecutar la reordenación de la lista de algoritmos disponibles, a través del control *Selector de algoritmo*.
4. Seleccionar los parámetros asociados a la ejecución de la reordenación mediante los controles *Tamaño ventana de ejecución*, *Solapamiento entre ventanas de ejecución*, *Adaptar ventanas a aeropuertos* y los controles adicionales del menú lateral si proceden.
5. Hacer click sobre el *Botón ejecutar reordenación* para realizar la reordenación.

Como resultado, se redibujan los gráficos del *dashboard* para mostrar los datos asociados a la nueva ordenación de los mensajes ADS-B de la trayectoria.

*Ejemplo: realizar una reordenación del vuelo IBE04NL usando el Simulated Annealing apoyado por el algoritmo de inserción más barata para obtener una ruta inicial, aplicando la técnica de uso de ventanas de un único tamaño máximo 130 puntos y solapamiento entre ventanas de 25 puntos. A continuación se lista la secuencia de pasos a seguir, ilustrada en la Figura A.17.*

1. Accionar la opción *Mapas* en el control *Widgets* (por defecto seleccionada).
2. Seleccionar el vuelo *IBE04NL* en el control *Selector de vuelo*.

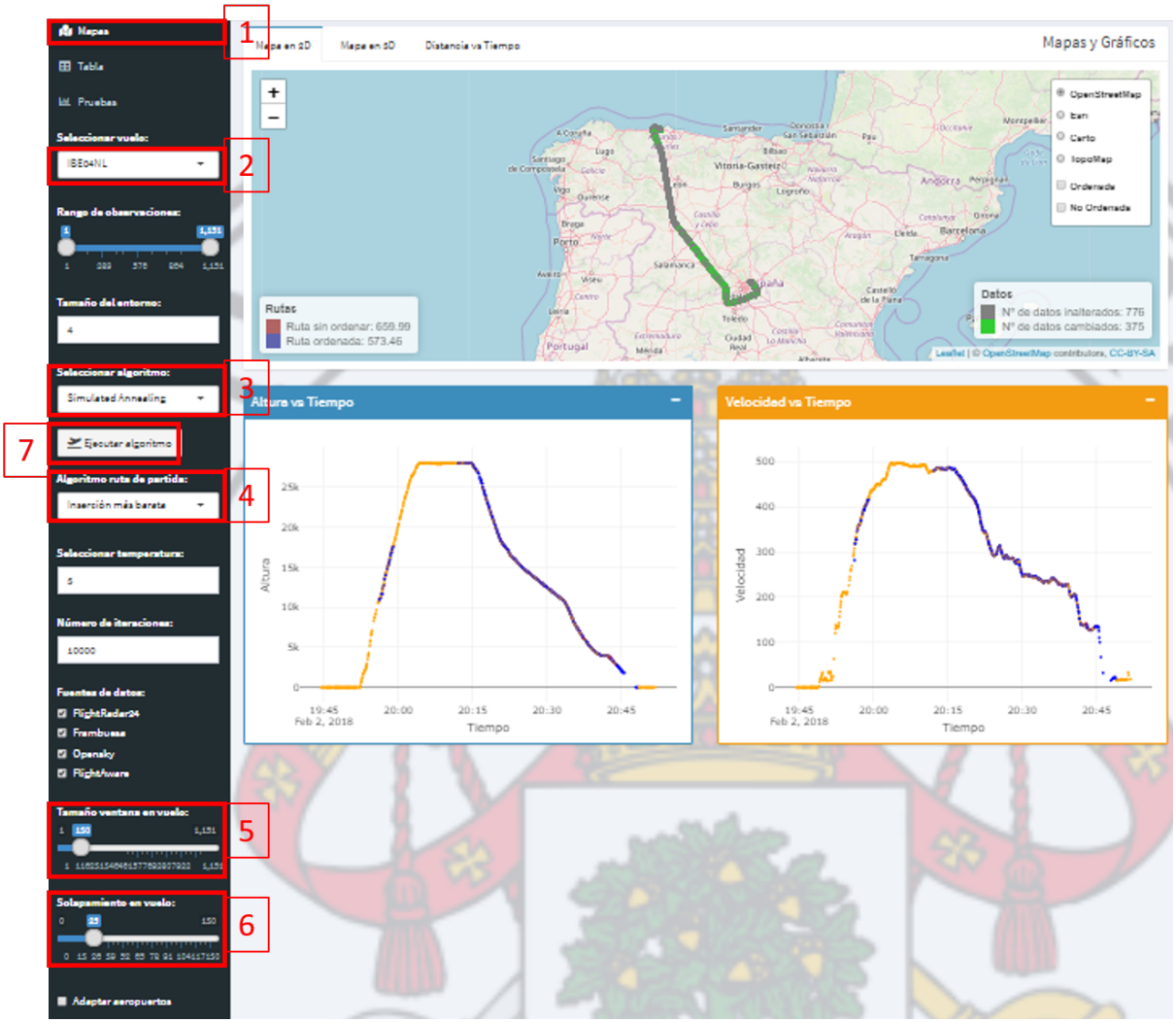



Figura A.17: Secuencia de pasos a realizar para llevar a cabo la función *Ejecutar una reordenación*

3. Seleccionar el algoritmo de reordenación *Simulated Annealing* a través del control *Selector de algoritmo*. Una vez hecho esto se mostrarán los controles adicionales asociados a este algoritmo (ver Figura A.10).
4. Seleccionar *Inserción más barata* en el control *Selector de algoritmo de partida*.
5. Fijar el valor 130 en el control *Tamaño de ventana de ejecución*.
6. Fijar el valor 25 en el control *Solapamiento entre ventanas de ejecución*.
7. Hacer click sobre el *Botón ejecutar reordenación*.

**Ejecutar una batería de pruebas [CAR-2.1]**

Además de la ejecución de una única reordenación, el usuario puede ordenar la ejecución de una batería de reordenaciones aplicadas a varios vuelos que el usuario elija y

haciendo uso a la carta de los algoritmos de reordenación disponibles en la aplicación. La secuencia de pasos a llevar a cabo para realizar esta función es la siguiente:

1. Accionar la opción *Pruebas* en el control *Widgets* del menú lateral.
2. Seleccionar los vuelos sobre los que ejecutar la reordenaciones de entre la lista de vuelos disponibles, en el control *Selector múltiple de vuelos*.
3. Seleccionar los algoritmos que se desean aplicar a las trayectorias elegidas en el paso anterior, de la lista de algoritmos disponibles, a través del control *Selector múltiple de algoritmos*.
4. Seleccionar la técnica de reordenación a llevar a cabo en el control *Opciones de ejecución*.
5. Si procede, seleccionar los parámetros de ejecución en el resto de campos del formulario (ver Figura A.16).
6. Hacer click sobre el *Botón ejecución de la prueba* para comenzar la batería de reordenaciones.

Como resultado, se muestra un nuevo componente en la zona inferior de la parte central de la pantalla que muestra las métricas asociadas a la prueba ejecutada, así como nuevos controles que permiten al usuario navegar entre los distintos datos que se muestran en pantalla.

*Ejemplo: realizar una batería de pruebas sobre los vuelos IBE04HT e IBE0519 usando los algoritmos 2-Opt e Inserción arbitraria, aplicando la técnica de uso de ventanas de dos tamaños con un máximo de 100 puntos en vuelo y solapamiento entre ventanas de 25 puntos, y un tamaño máximo de 20 puntos en vuelo y 5 puntos de solapamiento. Considerar una altura de 6000 para separar entre un tipo y otro de ventana. A continuación se lista la secuencia de pasos a seguir, ilustrada en la Figura A.18.*

1. Accionar la opción *Pruebas* en el control *Widgets* (por defecto seleccionada).
2. Seleccionar los vuelos *IBE04HT* y *IBE0519* en el control *Selector múltiple de vuelos*.
3. Seleccionar los algoritmos *2-Opt* e *Inserción arbitraria* en el control *Selector múltiple de algoritmos*.
4. Seleccionar en el control *Opciones de ejecución* la casilla "Utilizar ventanas de distinto tamaño para aeropuertos". A continuación aparecerán los controles adicionales específicos de este tipo de ejecución (ver parte derecha de la Figura A.16).
5. Fijar el valor 6000 en el control *Campo altura en prueba*.
6. Fijar el valor 100 en el control *Campo tamaño ventana en vuelo*.
7. Fijar el valor 20 en el control *Slider solapamiento en vuelo*.
8. Fijar el valor 20 en el control *Campo tamaño ventana en aeropuerto*.

9. Fijar el valor 5 en el control *Slider solapamiento en aeropuerto*.

10. Hacer click sobre el *Botón ejecución de la prueba*.

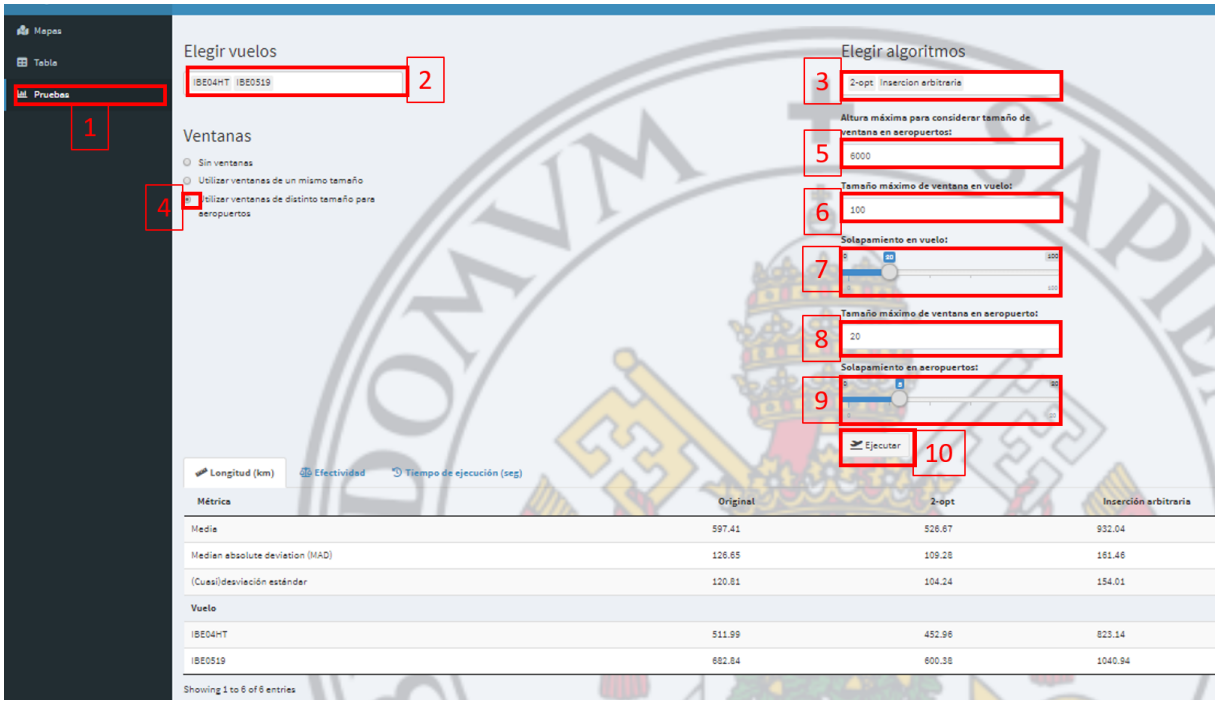



Figura A.18: Secuencia de pasos a realizar para llevar a cabo la función *Ejecutar una reordenación*

## Implementación escalable

En el caso de la implementación escalable, se adjunta un breve manual de usuario con sus posibilidades de uso.

La aplicación toma como entrada los parámetros indicados al `main()` antes de ejecutar. Éstos, se introducen separados por un espacio en blanco y en el orden siguiente:

- **Fichero de entrada** (`String`): ruta del fichero csv de entrada de datos.
- **Directorio de salida** (`String`): ruta donde se colocará el directorio con los resultados de la ejecución.
- **Modo de ejecución** (`int`): determina el uso o no de la técnica de las *ventanas de ejecución* (ver Subsección 5.4.1) para llevar a cabo la reordenación. Admite tres valores:
  - 0: no se utiliza la técnica de las *ventanas de ejecución*. Se ejecuta el algoritmo *k-Opt* una vez por trayectoria.

  - 1: se utiliza la técnica de las *ventanas de ejecución* con un único tamaño máximo y solapamiento.
  - 2: se utiliza la técnica de las *ventanas de ejecución* con dos tamaños máximo y solapamiento distinguiendo entre aeropuertos y zona central del vuelo.

- **Valor de k** (`int`): algoritmo *k-Opt* a ejecutar, con $k \geq 2$. Ejemplo: con $k = 2$ se ejecuta el *2-Opt*, con $k = 3$ se ejecuta el *3-Opt*, etc.
- **Tolerancia** (`int`): fija el número máximo de *k-changes* a probar sobre una ruta sin producirse ninguna mejora. Es un entero no negativo, y al introducirse con valor 0 se entiende que no hay límite de *k-changes* a probar, y por tanto, se prueban todos los posibles.
- **Tamaño en vuelo** (`int`): tamaño máximo de las ventanas que se usarán en la técnica de las *ventanas de ejecución* si el parámetro *modo de ejecución* es 1, o en la parte central del vuelo si el parámetro *modo de ejecución* es 2. Es un entero mayor que 2.
- **Solapamiento en vuelo** (`int`): solapamiento que se aplicará en la técnica de las *ventanas de ejecución* si el parámetro *modo de ejecución* es 1, o solo en la parte central del vuelo si el parámetro *modo de ejecución* es 2. Es un entero no negativo menor que el valor introducido en el parámetro *tamaño en vuelo*.
- **Tamaño en aeropuertos** (`int`): tamaño máximo de las ventanas que se usarán en la técnica de las *ventanas de ejecución* en la primera y última parte del vuelo si el parámetro *modo de ejecución* es 2. Es un entero mayor que 2.
- **Solapamiento en aeropuertos** (`int`): tamaño máximo de las ventanas que se usarán en la técnica de las *ventanas de ejecución* en la primera y en la última parte del vuelo si el parámetro *modo de ejecución* es 2. Es un entero no negativo menor que el valor introducido en el parámetro *tamaño en aeropuertos*.
- **Altura** (`double`): valor de altura que permite dividir la trayectoria en tres partes. El primer punto y el último que superen dicho valor serán los límites de la parte central del vuelo.

Al ejecutar la aplicación se lleva a cabo la separación de los registros por su valor de *leg*, la reordenación de los grupos de mensajes ADS-B resultantes y la posterior corrección de los *timestamps*. Como salida se obtiene un directorio de resultados en la ruta indicada por el usuario en el parámetro *directorio de salida*.