Universidad de **Valladolid**

ESCUELA DE INGENIERÍA INFORMÁTICA
DE SEGOVIA

Grado en Ingeniería Informática
de Servicios y Aplicaciones

Reconstrucción de trayectorias de aeronaves usando Simulated Annealing para resolver una versión del problema del viajante (TSP)

Alumna: María Zorita Mínguez

Tutores: Miguel Ángel Martínez Prieto
Anibal Bregón Bregón

"Todos tus sueños pueden hacerse realidad
si tienes el coraje de perseguirlos"

Walt Disney

"La confianza en sí mismo es
el primer secreto del éxito"

Ralph Waldo Emerson

Agradecimientos

Quiero dar las gracias en primer lugar a mis tutores, Anibal Bregón Bregón y Miguel Ángel Martínez Prieto, por darme la posibilidad de participar en el proyecto "AIRPORTS" en el que ellos colaboran, así como por toda su dedicación y apoyo en el desarrollo del mismo. Además, dar las gracias a mi tutor del Trabajo de Fin de Grado de Matemáticas, Pedro César Álvarez Esteban, ya que junto con mis tutores me ha permitido formar parte de este proyecto y con ello poder realizar ambos trabajos de fin de carrera de manera coordinada y conjunta.

También dar las gracias a Boeing Research and Technology Europe (BR&T-E), ya que la realización del presente trabajo se enmarca en la colaboración que llevan a cabo diversos miembros de la Universidad de Valladolid con ellos. Por ello, quiero agradecerles toda la información y datos aportados sobre las trayectorias de los vuelos, dado que han sido de gran importancia para la realización del trabajo.

Dar las gracias también a mi compañero Juan Manuel Velasco Heras por su apoyo y amistad durante estos 5 años que hemos estado juntos, y en particular, por la convivencia durante los últimos años de la carrera.

Además, dar las gracias a toda mi familia y amigos, y en especial a mi madre, por su apoyo incondicional y por darme ánimos en los momentos que más lo necesitaba.

Resumen

Una de las líneas de investigación del proyecto AIRPORTS (CIEN, 2015), liderado por Boeing Research and Technology Europe (BR&T-E), se centra en el estudio de la eficiencia de los vuelos de diferentes aeronaves comerciales teniendo en cuenta las trayectorias que describen. Dichas trayectorias se construyen a partir de datos ADS-B que son obtenidos de diferentes proveedores y captados por entidades receptoras presentes a lo largo del planeta.

El problema surge cuando al realizar la fusión de todos los datos capturados para construir cada una de las trayectorias, se observa el fallo en el alineamiento temporal de las señales, debido principalmente, al retardo en el tiempo de recepción de los mensajes y a la falta de sincronización horaria presente en algunas entidades receptoras.

Por ello, el presente proyecto tiene como objetivos principales abordar el problema comentado con anterioridad, logrando reconstruir las trayectorias reales proporcionadas mediante la aplicación de ciertos algoritmos, así como reasignar los tiempos a los datos alterados en la reordenación. Para ello, se busca modelizar el problema planteado como una variante del problema conocido como "El problema del viajante" (Traveling Salesman Problem, TSP), donde los nodos inicial y final son diferentes, buscando el camino hamiltoniano de mínima longitud.

El algoritmo estudiado y analizado para resolver dicho problema será el *Simulated Annealing*, ya que se tiene constancia de que funciona bien en contextos y situaciones muy diversas. Se trata de un algoritmo estocástico de optimización, diseñado principalmente para resolver problemas generales en los que existen muchos óptimos locales, y que por sus características, se espera que funcione bien y se obtengan resultados satisfactorios en el caso del problema planteado. Esto se debe a que en dicho problema la solución óptima global dista poco o no demasiado de la solución de partida.

Una vez realizado el proyecto, se puede concluir que los objetivos fijados han sido cumplidos con éxito, ya que se ha podido dar una solución escalable al problema planteado mediante la implementación de la heurística de mejora *Simulated Annealing* usando el modelo de programación *MapReduce*. Dado que los resultados obtenidos al aplicar dicho algoritmo a las distintas trayectorias de vuelo mejoran la situación de partida, el estudio realizado sirve para mejorar y avanzar en la gestión del tráfico aéreo.

Palabras claves: Recocido simulado, El problema del viajante, ADS-B, Big Data.

Abstract

One of the research lines of the AIRPORTS project (CIEN, 2015), led by Boeing Research and Technology Europe (BR&T-E), focuses on the study of flight efficiency of different commercial aircrafts taking into account the flight paths. These flight paths are constructed from ADS-B data that are obtained from different providers and collected by receivers present throughout the planet.

The problem arises when merging all the captured data used for constructing each one of the flight paths, the error in the temporal alignment of the signals is observed, mainly due to the delay time of reception of the messages and lack of time synchronization between the receivers.

Therefore, this project aims at addressing the problem discussed previously, managing to reconstruct the real flight paths provided by applying certain algorithms as well as reassigning the times to the altered data in the reording. To do this, we seek to model the problem raised as a variant of the problem known as "The Traveling Salesman Problem (TSP)", where the initial and final vertex are different, looking for the Hamiltonian path of minimum length.

The algorithm studied and analyzed to solve this problem will be the *Simulated Annealing* since it is known to provide good performance in very diverse contexts and situations. It is a stochastic optimization algorithm, designed mainly to solve general problems in which there are many local optimum, and because of its characteristics, it's expected to work well and obtain satisfactory results in the case of the problem posed. This is because in case of the problem raised, the global optimal solution is not far from the starting point solution.

Once the project is finished, it can be concluded that the goals set have been met successfully met, as it has been possible to provide a scalable solution to the problem raised through an implementation of the *Simulated Annealing* improvement heuristics using the *MapReduce* programming model. Given that the results obtained by applying this algorithm to the different flight paths improve the starting situation, the study carried out serves to improve the Air Traffic Management operations.

Keywords: Simulated Annealing, Traveling Salesman Problem, ADS-B, Big Data.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Desde hace muchos años existen mecanismos e instituciones que se encargan de gestionar y coordinar el tráfico aéreo de todo el mundo. Considerando el espacio aéreo europeo, que es uno de los más transitados a nivel mundial, se tiene como gestores de tráfico aéreo más importantes los siguientes: ENAIRE en España, DSNA en Francia, DFS en Alemania, ENAV en Italia, NATS en Reino Unido, EUROCONTROL MUAC en el Noroeste de Alemania, Bélgica, Luxemburgo y los Países Bajos. Hoy en día, se espera que el tránsito aéreo europero aumente de manera considerable en los próximos años [12]. Dado que los sistemas que existen en la actualidad no son capaces de afrontar esta nueva situación, se requiere desarrollar nuevos proyectos e iniciativas que logren mejorar la gestión del tráfico aéreo existente en la actualidad.

Por su gran importancia, destaca como gestor de tráfio aéreo europeo Eurocontrol, pues es la Organización Europea para la Seguridad de la Navegación Aérea que fue creada en 1963 con el objetivo de apoyar la aviación europea. Además, junto con la Comunidad Europea (CE), fundaron la empresa SESAR Joint Undertaking (SESAR JU) que está llevando a cabo la iniciativa conocida como Single European Sky ATM Research (SESAR). Dicho proyecto surge como instrumento tecnológico de la iniciativa europea Single European Sky (SES), y se trata de uno de los más innovadores desarrollados por la Unión Europea. Con su desarrollo, se busca reformar la arquitectura actual de la gestión del tráfico aéreo europeo, con el objetivo de crear y poner en práctica una política común de transporte aéreo. Todo esto, es necesario y muy importante (principalmente por la fragmentación del espacio aéreo europeo), y con ello, se pretende lograr mejorar la eficiencia y la seguridad del tráfico aéreo, reducir gastos de funcionamiento, el impacto medioambiental, así como optimizar el uso del espacio aéreo.

Dado que la gestión del tráfico aéreo es un tema bastante complejo y delicado de realizar, ya que intervienen muchos elementos y factores, estos nuevos sistemas buscan introducir mejoras para modernizar los sistemas que existen en la actualidad. Para ello, se requiere hacer uso de varias técnicas de vigilancia, entre las que destacan las técnicas radar y el sistema Automatic Dependant Surveillance (ADS).

Dentro de este marco de investigación, surge el proyecto "AIRPORTS (CIEN, 2015)" liderado por Boeing Research and Technology Europe, el cual, entre otras muchas líneas de investigación, busca estudiar como mejorar la eficiencia de los vuelos de aeronaves comerciales en función de la trayectoria que estos describen al realizar dicho vuelo. Dichas aeronaves están equipadas con un sistema conocido como Automatic Dependant Surveillance Broadcast (ADS-B) que se encarga de enviar diferentes mensajes ADS-B describiendo el estado del vuelo en cada momento. Estos mensajes contienen información de importancia como: el identificador del mensaje, el tiempo de recepción (*timestamp*), la posición (en términos de longitud, latitud y altitud) de la aeronave, su velocidad (tanto horizontal como vertical), si está o no en suelo, la fecha de realización, etc. Además, se necesitan diferentes redes de sensores distribuidas por distintas partes del planeta, las cuales, permiten obtener diversas fuentes de datos ADS-B. Dichas redes de sensores pueden ser comerciales o no.

Una vez que se reciben los mensajes ADS-B que describen el estado del vuelo, el equipo del proyecto "AIRPORTS" los utiliza para reconstruir las distintas trayectorias de vuelo. Gracias a su reconstrucción es posible realizar su estudio, y para ello, se usa una plataforma Big Data, la cual, es un prototipo que permite calcular una gran variedad de métricas de eficiencia.

1.1. Motivación

La motivación de este proyecto surge cuando al estudiar dichas trayectorias nos damos cuenta de que hay problemas con respecto al alineamiento temporal de las señales obtenidas. Las causas principales de dicho problema son: el retardo en el tiempo de recepción de dichos mensajes y la falta de sincronización horaria existente entre algunas entidades receptoras.

Por lo tanto, nuestro propósito es dar una solución a dicho problema, reconstruyendo las trayectorias para obtener soluciones lo más próximas posibles a las trayectorias que realmente fueron voladas por las aeronaves. El objeto de estudio presenta cierta relación con problemas ya existentes y estudiados desde hace mucho tiempo como es la teoría de grafos, la optimización, etc. pudiendo además ser formulado como un problema matemático que es una variante del famoso problema conocido como "El problema del viajante o Traveling Salesman Problem (TSP)".

Dicho problema consiste en lo siguiente: dado un número n de ciudades, se debe buscar el camino de coste mínimo formado por $n+1$ ciudades que pase por cada una de ellas en una única ocasión, excepto para la primera, que tiene que ser visitada dos veces puesto que la ciudad de partida y de llegada deben coincidir, esto es, son la misma.

1.2. Objetivos

El presente trabajo tiene los siguientes objetivos:

1. Realizar un estudio y análisis del comportamiento de distintos algoritmos cuando son aplicados a diferentes vuelos. Los subobjetivos que se derivan son:

   a. Construir un dashboard usando el entorno *R-Studio*. Con ello se pretende facilitar la realización del estudio preliminar sobre el funcionamiento de cada uno de los algoritmos considerados sobre distintos ejemplos.

   b. Aplicar, usando dicho dashboard, los algoritmos a mensajes ADS-B que definen vuelos reales, para así poder hacer una comparativa de los mismos en cuanto a la distancia mejorada entre la trayectoria ordenada y la de partida, así como en función del tiempo de ejecución.

2. Una vez realizado dicho estudio, considerar aquellos algoritmos que mejor se adapten o comporten frente al problema del fallo en la asignación de tiempos en mensajes ADS-B. El subobjetivo que se deriva es el siguiente:

   a. Implementar de manera escalable usando el lenguaje Java y el modelo de programación *MapReduce* el algoritmo propuesto en el proyecto, esto es, el *Simulated Annealing*.

3. Una vez aplicados los algoritmos correspondientes para ordenar las trayectorias, reasignar los tiempos o *timestamps* a los puntos que han sufrido cambios al realizar dicha ordenación. El subobjetivo que se deriva es el siguiente:

   a. Estudiar y analizar diferentes modelos de optimización para realizar la interpolación de los tiempos asociados a los puntos que han sufrido cambios.

Con el logro de dichos objetivos se podrá: Obtener una aproximación más realista acerca de las trayectorias que realmente fueron voladas por las aeronaves, y como consecuencia de ello, utilizar los datos correspondientes para mejorar tanto la eficiencia como la seguridad del tráfico aéreo actual.

1.3. Estructura del documento

El presente trabajo se va a estructurar en una serie de capítulos que se detallan y describen a continuación:

Capítulo 1. Introducción: Durante este capítulo, que es el presente, se realiza una pequeña introducción para describir brevemente cuál es el motivo por el que se plantea la realización del trabajo, así como los objetivos que se persiguen conseguir.

Capítulo 2. Plan del proyecto: Durante este capítulo se describe la metodología o planificación seguida para el desarrollo del presente trabajo de investigación, esto es, cómo se han ido realizando las diferentes tareas desde el inicio hasta la finalización del mismo.

Capítulo 3. Air Traffic Management (ATM): Durante este capítulo se describe la situación de la gestión del tráfico aéreo existente en la actualidad, así como los nuevos sistemas de mejora que se están desarrollando. También, se da una explicación detallada sobre la importancia que tienen los mensajes ADS-B a la hora de determinar las trayectorias seguidas por las aeronaves, y los problemas que estos sistemas plantean. Finalmente, se describe el problema encontrado sobre el fallo en la sincronización de los tiempos de dichos mensajes, siendo el propósito del trabajo su mejora y resolución. Como consecuencia de dicha mejora, se podrán obtener trayectorias más próximas a las que realmente fueron voladas por las aeronaves y realizar la reasignación de tiempos correspondiente para aquellos puntos cuyo orden ha sido alterado.

Para la realización de la primera parte de dicho capítulo ha sido de gran utilidad la lectura de los artículos [10], [11] y [12] proporcionados por los tutores, ya que me han ayudado a conocer en mayor profundidad el entorno aeronáutico.

Capítulo 4. Traveling Salesman Problem (TSP): Durante este capítulo se describe el problema del viajante ya que es una variante del problema que debemos resolver. Para resolver este problema existen diversos métodos que encuentran una ruta factible a través de la búsqueda de ciclos hamiltonianos, pero en la realización del trabajo, nos centraremos en los algoritmos de mejora, y en particular, en el algoritmo *Simulated Annealing*. Este algoritmo se caracteriza por ser estocástico y por funcionar de manera adecuada en bastantes ocasiones, principalmente, en el caso en que la solución de partida sea próxima a la óptima, que es la situación que tendremos en la práctica. Aunque no se detallarán de manera formal los fundamentos matemáticos sobre los que se basa la convergencia asintótica del algoritmo, si se mencionarán los resultados fundamentales.

Capítulo 5. Estudio preliminar de los algoritmos en *R-Studio*: Durante este capítulo se describen los objetivos y la funcionalidad que se pretende conseguir con la realización del dashboard, la implementación realizada sobre el mismo usando el entorno *R-Studio*, así como las herramientas necesarias para ello.

Capítulo 6. Análisis de los resultados obtenidos: Durante este capítulo se realiza un análisis y comparativa de los resultados obtenidos al aplicar una serie de métodos (tanto en términos de distancia, de eficacia como tiempo de ejecución) para reordenar las trayectorias de un conjunto de vuelos dados. Además, se analiza el comportamiento del algoritmo *Simulated Annealing* cuando es aplicado a un conjunto grande de vuelos para distintas configuraciones de ventanas (sin usar ventanas de tiempo, usando ventanas del mismo tamaño en el aeropuerto y vuelo y cuando se usan ventanas de distinto tamaño en ambas zonas). Esto sirve para poder determinar en qué situaciones el algoritmo funciona mejor, esto es, arroja resultados más próximos a la solución óptima.

Capítulo 7. Implementación del *Simulated Annealing* usando *MapReduce*: Durante este capítulo se explican las herramientas y el entorno necesario para realizar una implementación escalable en *MapReduce* del algoritmo *Simulated Annealing*. Además, se detalla cómo se ha realizado la correspondiente reasignación de tiempos para cada uno de

los mensajes ADS-B una vez obtenida la trayectoria ordenada tras aplicar el algoritmo implementado.

Capítulo 8. Conclusiones y trabajo futuro: Finalmente, en este último capítulo se detallan las conclusiones obtenidas sobre la realización del trabajo, así como el aprendizaje obtenido con el mismo y el posible trabajo futuro a realizar.

Conviene decir que la realización del presente Trabajo de Fin de Grado se ha realizado de manera paralela con el Trabajo de Fin de Grado de Matemáticas. Por ello, la temática de ambos es la misma, pero mientras que el de Matemáticas se centra principalmente en el estudio y análisis teórico de la convergencia del algoritmo *Simulated Annealing*, el de Informática busca implementar una solución escalable que permita resolver el problema planteado en la práctica. Ha sido de gran utilidad poder realizar ambos trabajos de manera conjunta puesto que así se ha podido profundizar más en el estudio del algoritmo y con ello dar una buena solución al problema planteado.

También, conviene señalar que dicho trabajo se ha realizado a la par que el Trabajo de Fin de Grado de Informática llevado a cabo por mi compañero Juan Manuel Velasco Heras. Aunque el objetivo de ambos trabajos es el mismo, yo me he centrado en estudiar la heurística de mejora *Simulated Annealing* que se basa en la aleatoriedad o el azar, y él las heurísticas de mejora local, como es el caso del método *2-opt* o *Lin-Kernighan*, que se basan en la realización de intercambios. Todo esto ha enriquecido el trabajo realizado y ha permitido realizar una comparativa de los resultados obtenidos por dichos algoritmos para sacar conclusiones sobre su funcionamiento.

Capítulo 2

Plan del proyecto

Durante este capítulo se pretende describir el plan de proyecto seguido para realizar el presente trabajo. Al ser un proyecto de investigación, aunque se parte de unos objetivos claros, los cuales fueron comentados en el Capítulo 1, todas las etapas y pasos a realizar no están perfectamente definidos al comienzo.

Por ello, en este caso, las metodologías tradicionales no son apropiadas y, se puede decir, que el desarrollo del mismo se ajusta a una metodología de tipo ágil ya que el trabajo se ha ido realizando de forma iterativa e incremental. Durante cada sprint o iteración se realizan una serie de tareas o actividades que se contrastan con los clientes, en este caso los tutores, para dar paso a la realización de las siguientes tareas y así sucesivamente. Por lo tanto, se trata de un plan de proyecto donde, tras cada iteración, se consiguen una serie de funcionalidades establecidas, logrando cada vez estar más cerca de alcanzar los objetivos fijados al comienzo.

A continuación, se describe el plan de proyecto seguido, el cual, puede dividirse básicamente en cuatro iteraciones diferentes. En cada una de estas iteraciones se realizan una serie de actividades y para que sea más claro, se muestra para cada etapa una tabla con las actividades de las que consta, así como una estimación del tiempo y el estado de las mismas.

Cabe destacar que aunque la realización del trabajo comenzó en el mes de Septiembre, esto es, con el comienzo del curso, no fue hasta mediados-finales de Enero cuando se empezó a dedicar un mayor tiempo a su desarrollo. Esto fue debido a que en el primer cuatrimestre todavía había asignaturas y a que al principio me centré más en el Proyecto de Fin de Carrera de Matemáticas, dado que antes de poder programar los algoritmos usando distintos programas y lenguajes de programación, era necesario conocer el funcionamiento de los mismos en mayor profundidad.

Sprint 1 $\rightarrow$ Consta de todas aquellas tareas iniciales que tienen como objetivo conocer en mayor profundidad el entorno aeronáutico, así como el problema del viajante o TSP, junto con los distintos algoritmos que permiten su resolución.

ID	Descripción	Tiempo estimado	Estado
T-01	Investigar y profundizar en el estudio del entorno aeronáutico y la gestión del tráfico aéreo.	12 horas	Acabada
T-02	Entender y especificar con claridad el problema sobre el alineamiento de las trayectorias obtenidas por Boeing.	5 horas	Acabada
T-03	Lectura y consulta de documentos y libros relacionados con el problema del viajante (TSP) y los diferentes algoritmos que permiten su resolución.	45 horas	Acabada
T-04	Redacción de la introducción (Capítulo 1) junto con los Capítulos 3 y 4 del presente documento.	50 horas	Acabada
		112 horas	

Tabla 2.1: Descripción de las tareas de la primera etapa

Sprint 2 → Consta de todas aquellas tareas centradas en el estudio y diseño del dashboard usando el entorno de trabajo *R-Studio*, así como el análisis de los resultados obtenidos.

ID	Descripción	Tiempo estimado	Estado
T-05	Instalación de *R-Studio*.	0.25 horas	Acabada
T-06	Instalación de los distintos paquetes y librerías necesarios para la realización del dashboard.	0.35 horas	Acabada
T-07	Aprendizaje y documentación sobre la creación del dashboard usando *R-Studio*.	35 horas	Acabada
T-08	Creación de la versión inicial del dashboard (funcionalidad básica).	30 horas	Acabada
T-09	Creación de una segunda versión que incluye la posibilidad de elegir ventanas de tiempo tanto en las zonas del aeropuerto como en la zona central del vuelo.	45 horas	Acabada
T-10	Mejora del dashboard incluyendo una interfaz gráfica más visual para el usuario. Esto permite que sea más usable y que la información mostrada sea más representativa, logrando así mejorar su satisfacción.	25 horas	Acabada

Sigue en la siguiente página...

ID	Descripción	Tiempo estimado	Estado
T-11	Mejora del dashboard añadiendo la posibilidad de realizar baterías de pruebas seleccionando los vuelos, métodos y consideraciones deseadas, y tras ello, visualizar los resultados (en términos de distancia, eficacia y tiempo de ejecución) en una tabla. Además, se añade la posibilidad de descargar en formato *csv* la información de cada tabla.	28 horas	Acabada
T-12	Implementar la parte asociada a la asignación de los nuevos tiempos tras la reordenación, añadir los comentarios que faltaban al código y limpieza del mismo.	25 horas	Acabada
T-13	Análisis de los resultados obtenidos una vez aplicados los distintos métodos a diversos vuelos.	15 horas	Acabada
T-14	Aplicación de los distintos algoritmos a una serie de vuelos dados para las distintas posibilidades de configuraciones de ventanas, y en particular, para el *Simulated Annealing*.	20 horas	Acabada
T-15	Redacción de los Capítulos 5 y 6.	30 horas	Acabada
		253.6 h	

Tabla 2.2: Descripción de las tareas de la segunda etapa

Sprint 3 → Consta de todas aquellas tareas centradas en la implementación del algoritmo *Simulated Annealing* en Java así como su integración en *MapReduce*. El objetivo es dar soporte a la computación paralela de grandes conjuntos de datos provenientes de distintos vuelos. Con ello, se consigue una solución eficiente y escalable para el problema planteado.

ID	Descripción	Tiempo estimado	Estado
T-16	Consulta y búsqueda de ayuda sobre la implementación del algoritmo *Simulated Annealing* o la obtención del pseudocódigo.	5 horas	Acabada
T-17	Comienzo de la implementación del algoritmo en *NetBeans 8.2* (primeras clases y funcionamiento básico).	15 horas	Acabada
T-18	Continuación de la implementación incluyendo una interfaz gráfica que muestra tanto la distancia obtenida al aplicar el algoritmo como el tiempo de ejecución.	18 horas	Acabada
T-19	Ampliación de la implementación considerando las distintas configuraciones de ventanas de tiempo tanto en las zonas del aeropuerto como en la zona central del vuelo.	15 horas	Acabada

Sigue en la siguiente página...

ID	Descripción	Tiempo estimado	Estado
T-20	Análisis de los resultados obtenidos.	2 horas	Acabada
T-21	Búsqueda y aprendizaje sobre el uso de *MapReduce* (tutoriales, guías, etc.).	20 horas	Acabada
T-22	Instalación de *X2Go Cliente* para poder usar la máquina virtual *Cloudera* y realización del programa WordCount en Java para familiarizarme con el uso de *MapReduce*.	5.35 horas	Acabada
T-23	Adaptación de la implementación del algoritmo en *NetBeans* usando *MapReduce*.	8 horas	Acabada
T-24	Implementar la parte asociada a la asignación de tiempos tras la reordenación de las trayectorias, añadir ciertos comentarios que faltaban al código y limpieza del mismo.	25 horas	Acabada
T-25	Redacción del Capítulo 7.	15 horas	Acabada
		128.35 h	

Tabla 2.3: Descripción de las tareas de la tercera etapa

Sprint 4 → Consta de todas aquellas actividades relacionadas con las pruebas y validaciones de los resultados obtenidos, la corrección de errores, así como la formulación de las conclusiones sobre el trabajo.

ID	Descripción	Tiempo estimado	Estado
T-26	Realización de diversas pruebas sobre el correcto funcionamiento del dashboard y el proyecto creado usando el modelo de programación *MapReduce*.	25 horas	Acabada
T-27	Evaluación final de los resultados obtenidos y conclusiones del trabajo (redacción del Capítulo 8).	8 horas	Acabada
T-28	Redacción del esquema de trabajo realizado (Capítulo 2 del presente documento) y revisión final del documento.	15 horas	Acabada
		48 horas	

Tabla 2.4: Descripción de las tareas de la cuarta etapa

Sumando las horas totales de las cuatro tablas mostradas con anterioridad, se obtiene un total de 541.95 horas. Como se puede apreciar, el sprint o iteración de mayor duración es el segundo, esto es, aquel cuyo principal objetivo era la creación del dashboard usando *R-Studio*, junto con el análisis de los resultados obtenidos al ejecutar los diferentes algoritmos a los vuelos proporcionados, haciendo uso del mismo. Tras ello, el siguiente sprint de mayor duración es el tercero, dado que la implementación del algoritmo *Simulated Annealing* en *MapReduce* era algo novedoso ya que no conocía dicho modelo de programación. Por ello, ha sido necesario dedicar un tiempo para entenderlo y saber usarlo antes de poder realizar la implementación del algoritmo.

El primer sprint también ha requerido de bastantes horas, dado que antes de comenzar las distintas implementaciones era necesario conocer en mayor profundidad el entorno aeronáutico, el funcionamiento del problema del viajante o TSP, así como de los distintos algoritmos que permiten su resolución, centrándome en especial en el estudio del comportamiento del algoritmo *Simulated Annealing*.

Cabe destacar, como ya se comentó en la Introducción, que dicho Trabajo de Fin de Grado se realiza de manera coordinada con el Trabajo de Fin de Grado de Matemáticas. Aunque en ambos hay partes comunes (ese era el objetivo al realizarlo de esta manera), la línea de trabajo de cada uno de ellos es diferente. Mientras que en el de Matemáticas el pilar fundamental es el estudio teórico de la convergencia del algoritmo *Simulated Annealing*, en el de Informática, es la implementación de una solución escalable que permita resolver el problema que se nos plantea mediante el uso de dicho algoritmo. Por ello, aunque las horas dedicadas al estudio y análisis teórico sobre la convergencia de dicho algoritmo no se describen ni detallan en profundidad en el presente documento, cabe decir que antes de realizar las implementaciones prácticas se dedicaron varios meses a realizar dicho trabajo. Por todo esto, se puede decir que ha sido de gran utilidad realizar dichos trabajos de manera conjunta. Por un lado, en el Trabajo de Fin de Grado de Matemáticas he podido corroborar los resultados teóricos descritos sobre la convergencia del algoritmo mediante su visualización práctica. Por otro lado, para el de Informática, he podido entender en detalle y profundidad el funcionamiento de dicho algoritmo antes de realizar la implementación del mismo usando el modelo de programación *MapReduce*, lo cual, me ha ayudado a ir más rápido en su realización y entender cada uno de los pasos realizados.

Aunque el cuarto sprint es el de menor duración, también tiene mucha importancia, dado que es necesario realizar una batería de pruebas y comprobaciones para verificar el correcto funcionamiento del dashboard creado y la implementación del algoritmo en *MapReduce*, junto con una revisión del presente documento para evitar posibles errores.

Capítulo 3

Air Traffic Management (ATM)

Aunque la gestión del tráfico aéreo se viene realizando desde hace años, cada vez es más necesaria y compleja, dado que, con el paso del tiempo se prevé que el tráfico aéreo vaya aumentando de manera importante y considerable.

Existen unos sistemas, conocidos como "Sistemas de Navegación Aérea", que sirven para proporcionar diversos servicios a las aeronaves durante todo el trayecto, esto es, desde el despegue hasta el aterrizaje, para que así, estos puedan llevar a cabo el plan de vuelo previsto. Además, permiten gestionar el espacio aéreo logrando satisfacer la demanda de forma óptima, teniendo siempre presente la seguridad de las aeronaves.

En la Figura 3.1 se muestran las distintas etapas del vuelo junto con los sistemas de control y gestión usados en cada una de ellas.

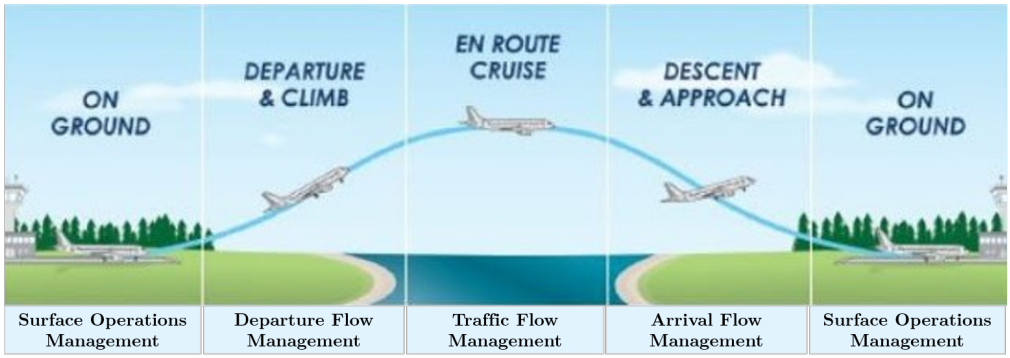


Figura 3.1: Gestión del tráfico aéreo [13]

La gestión del tráfico aéreo requiere la instalación de nuevos sistemas y tecnologías de comunicación entre el aeropuerto, la aeronave y el centro de control, para así, aumentar la capacidad que tienen los Servicios de Navegación Aérea. Esto es necesario debido a la saturación de datos que se prevé como consecuencia del aumento del tráfico aéreo.

ATM es un término de aviación que engloba todos los sistemas encargados de ayudar a las aeronaves en el despegue, en el tránsito en vuelo y en el aterrizaje, incluyendo los

Servicios de tránsito aéreo (Air Traffic Services, ATS), Gestión del espacio aéreo (Airspace Management, ASM), Flujo de tráfico aéreo y Gestión de capacidad (Air Traffic Flow and Capacity Management , ATFCM). Por todo ello, ATM es esencial en el trasporte aéreo y la aviación, ya que se ocupa de realizar diversas funciones importantes como son:

- Garantizar la seguridad de las aeronaves (dos aeronaves no pueden encontrarse en la misma posición en el mismo momento) y lograr que se cumpla el plan de vuelo fijado.
- Conectar miles de ciudades entre sí de todo el mundo.
- Tener en cuenta el cambio climático permitiendo disponer de un mayor número de zonas verdes y espacios libres en las ciudades.
- Permitir aumentar la infraestructura que existe en la actualidad y proporcionar servicios de información avanzados.
- Actuar como catalizador de la competitividad y la capacidad de innovación de Europa.

Sin embargo, esta gestión no es nada sencilla, ya que el entorno considerado presenta muchas variables que hay que tener en cuenta a la hora de llevarla a cabo. Alguna de ellas son la gran cantidad de aeropuertos que existen, así como de aerolíneas y de aeronaves por cada uno de ellos.

Se sabe que el espacio aéreo europeo es uno de los más transitados en todo el mundo, y además, se prevé que esta situación vaya en aumento [12]. Dado que los sistemas ATM no son lo suficientemente eficaces para hacer frente a esta situación, debido a que se basan en tecnologías y métodos anticuados, es necesario desarrollar una modernización de los mismos. Este hecho se trata de un problema frecuente en la actualidad que no sólo afecta al tráfico aéreo europeo sino a nivel mundial.

En el caso europeo la iniciativa desarrollada para modernizar la gestión del tráfico aéreo se conoce como Single European Sky ATM Research (SESAR). Este proyecto es uno de los más conocidos junto con NextGen, que es desarrollado por Estados Unidos.

3.1. Single European Sky ATM Research (SESAR)

SESAR es un proyecto llevado a cabo por SESAR Joint Undertaking (SESAR JU) que fue fundada por la Comunidad Europea (CE) junto con Eurocontrol. Eurocontrol es la Organización Europea para la Seguridad de la Navegación Aérea que fue creada en 1963 y tiene por objeto apoyar la aviación europea.

SESAR surge como instrumento tecnológico de la iniciativa europea Single European Sky (SES), la cual, busca reformar la arquitectura actual de la ATM en Europa con el objetivo de desarrollar y poner en práctica una política común de transporte aéreo. Se trata de uno de los proyectos de infraestructura más innovadores llevados a cabo por la

Unión Europea con el objetivo de definir, desarrollar y desplegar todo aquello que sea necesario para aumentar el rendimiento de la gestión del tráfico aéreo, esto es, modernizarlo.

Su principal objetivo es implantar en el año 2020 una red ATM a nivel europeo que posea altas prestaciones y un rendimiento más eficiente que el presente en la actualidad. Para ello, es necesario su integración dentro del Sistema Global de transito Aéreo, así como su interoperabilidad con otros sistemas del mundo, como es el caso del estadounidense llevado a cabo por el proyecto NextGen.

Para ello, algunos de los propósitos que pretende conseguir SES con este proyecto son: poder gestionar el triple del volumen de tráfico aéreo actual, reducir el impacto sobre el medioambiente en aproximadamente un 10 % para cada vuelo, aumentar por un factor de 10 la seguridad, facilitar la libre circulación de personas, mercancías, etc., así como reducir costes, tiempos de viaje, etc.

En el caso de querer obtener más información sobre SESAR, esta puede encontrarse en la referencia bibliográfica [9].

Para conseguir estos logros, se requiere el uso de técnicas de vigilancia aérea, entre las que destacan las técnicas radar y el sistema Automatic Dependent Surveillance (ADS).

- **Técnica radar**: Es una técnica antigua que comenzó a usarse en los años de la Segunda Guerra Mundial (siglo XIX). Estos sistemas permiten conocer la distancia en la que está la aeronave, y si la aeronave tiene instalado un transpondedor, es posible enviar información desde la misma sobre su altura e identificación. Con esa nueva información se puede conocer la posición de la misma en términos de latitud, longitud y altura.

  El proceso se basa en lo siguiente: en primer lugar una señal en forma de onda electromagnética es emitida por el radar, y tras ser reflejada en la aeronave, vuelve a la estación radar. Así, se puede determinar el tiempo que transcurre desde que se envía la señal hasta que se vuelve a recibir en la estación, pudiendo determinar la distancia a la que se encuentra teniendo en cuenta una serie de consideraciones.

  Ahora bien, se trata de una técnica que tiene bastantes limitaciones tanto geográficas (zonas de sombras como es el caso de los océanos) como operacionales. Por ello, surgen otros sistemas más modernos para solventar estos problemas, como es el caso de la técnica ADS.

- **Automatic Dependant Surveillance (ADS)**: Debido a que es el tipo de datos que se usarán en el proyecto, se explica de manera detallada en la siguiente sección apoyándose en la información obtenida de los artículos anteriormente indicados en la Introducción, así como de otras referencias bibliográficas [8].

3.2. ADS

Automatic Dependent Surveillance es un sistema de control de tráfico aéreo que permite que las aeronaves proporcionen de manera automática, a través de un enlace de datos, todos los datos necesarios sobre la navegación, posición, identificación, etc. al servicio de control de tráfico aéreo (ATS), para así, lograr mejorar la seguridad del tráfico aéreo.

Estos datos pueden ser transmitidos al servicio de control de tráfico aéreo bien de manera periódica o bien por petición de dicho servicio. En función de ello, se consideran dos tipos de informes diferentes que son: el "Informe Periódico" que se envía de forma periódica y contiene datos de carácter obligatorio (identificación de la aeronave, posición en términos de latitud, longitud y altitud, tiempo, etc.) así como otros de carácter opcional y el "Informe sobre otros aspectos adicionales" que se genera a petición del servicio.

ADS se caracteriza por ser un sistema:

1) Automático, esto es, no requiere la intervención de ningún operador (como puede ser el piloto o una entidad externa) para realizar el envío de los datos.

2) Dependiente ya que la generación de los datos depende de los datos del sistema de navegación de la aeronave a bordo.

3) Proporciona datos similares a los datos de radar tanto cuando se realiza la transmisión de los mismos a los controladores de tierra como a otros aviones. Permite llegar a zonas que no están cubiertas por los radares además de mejorar la vigilancia en las zonas que sí lo están.

Actualmente, se usan dos versiones diferentes de ADS, las cuales son: ADS-C y ADS-B.

Automatic Dependent Surveillance-Contract (ADS-C): Este sistema sólo transmite información sobre la aeronave a bordo a ciertos receptores, las Unidades de Servicios de Tránsito Aéreo (ATSU) terrestres. Además, en este tipo de sistemas, el tránsito de información se basa en un contrato previamente establecido entre la ATSU y la aeronave correspondiente que puede ser: por demanda, periódico, de evento y/o de emergencia.

Automatic Dependent Surveillance-Broadcast (ADS-B): Este sistema transmite de manera periódica y automática datos sobre la aeronave a bordo a todos los receptores que se encuentren dentro de un determinado rango cercano. Estos receptores pueden ser bien una estación de control situada en la tierra, así como otras aeronaves dentro de una zona determinada. Se trata de la versión más reciente y usada de sistemas ADS, y por lo tanto, la que se considerará en el trabajo.

Centrándonos en el sistema ADS-B, los mensajes transmitidos por dicho sistema se conocen como mensajes ADS-B. En ellos, se proporciona información sobre la posición de la aeronave (latitud, longitud y altitud) que se obtiene por GPS, el identificador del mensaje, el identificador de la aeronave, la velocidad, así como otra información adicional. Como en este caso la transmisión de los datos no tiene una dirección fija, pueden ser transmitidos a

cualquier aeronave de su entorno siempre que cuente con el equipamiento necesario para ello.

Existen dos tipos de componentes para la tecnología ADS-B que son:

· **ADS-B Out**: Es de uso obligatorio para todas las aeronaves.

· **ADS-B In**: Hasta el momento su uso es opcional.

Por un lado, ADS-B Out permite a la aeronave transmitir información de forma continua sobre posición, identificación, altitud y velocidad a estaciones terrestres. Por otro lado, ADS-B In es de uso opcional y permite a las aeronaves con el equipamiento adecuado recibir mensajes ADS-B de otras aeronaves dentro de su zona o de un rango establecido.

En la Figura 3.2 se muestra el proceso de transmisión de mensajes ADS-B tanto entre varias aeronaves como a estaciones receptoras en tierra.

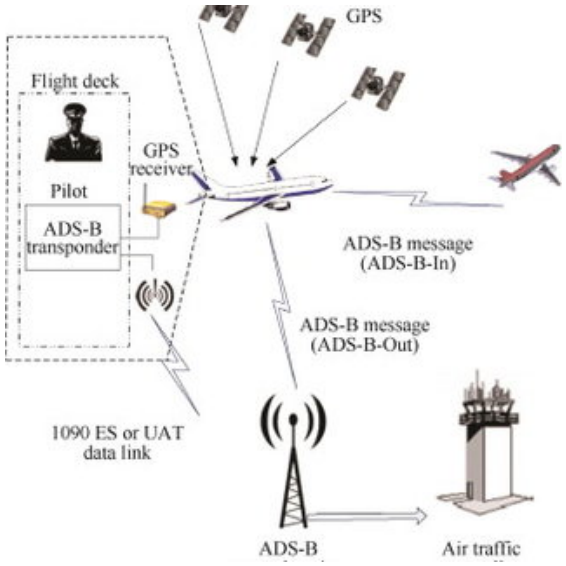


Figura 3.2: Proceso de transmisión de mensajes ADS-B

Por lo tanto, la tecnología ADS-B tiene varios objetivos, siendo los más importantes los siguientes:

· Transmitir los datos entre aeronaves de una misma zona aérea con el propósito de aumentar la seguridad del tráfico aéreo. De esta manera, las aeronaves conflictivas se marcan en color rojo, las potencialmente conflictivas en color amarillo y las no conflictivas en color verde permitiendo que el piloto pueda tomar las medidas oportunas para garantizar la mayor seguridad posible en el tránsito aéreo.

· Transmitir los datos a determinadas unidades de control de tránsito aéreo terrestres para que puedan regular dicho tráfico aéreo de manera eficiente.

· Transmitir los datos desde la aeronave durante la maniobra en tierra a las diferentes unidades de control de tránsito aéreo terrestre.

Este tipo de sistemas presentan grandes ventajas como son:

- Proporcionan información de posición en tiempo real que obtienen de un sistema de navegación que, por lo general, es más preciso que un sistema basado en radar. Además, abarcan un área de cobertura mucho más extensa que la de radar. Por lo tanto, al tener mayor precisión se garantiza una mayor seguridad y una mayor capacidad para controlar el espacio aéreo.
- Los mensajes ADS-B permiten reconstruir la trayectoria seguida por el vuelo.
- Permite a las unidades ATS mayor facilidad a la hora de identificar y monitorizar la aeronave a través de los datos recibidos de la misma. Por lo tanto, tienen un conocimiento más preciso del tráfico aéreo existente.
- Alerta de la situación en la que se encuentra cada una de las aeronaves.
- Permite realizar cambios de manera rápida y sencilla mediante la comunicación de voz en caso de la existencia de algún peligro.
- Permite que se reduzca la carga de trabajo que tienen los controladores aéreos.
- Permite reducir también el tiempo de uso del canal a través del cual se transmiten los mensajes.
- Se reducen los retrasos de las aeronaves tanto en el despegue, el rodaje y el aterrizaje en pista.

Aunque los sistemas ADS-B presentan una gran variedad de ventajas es cierto que tienen algunos inconvenientes, siendo, el principal de ellos, los problemas de escalabilidad que presentan cuando tienen que tratar grandes volúmenes de datos, algo muy frecuente dado que los mensajes ADS-B se emiten dos veces por segundo por cada una de las aeronaves. Otro de los inconvenientes es que, en la actualidad, no existe una adecuada cobertura sobre la infraestructura a nivel mundial y, además, muchas aeronaves aún no disponen del equipamiento adecuado para el tratamiento de mensajes ADS-B.

3.3. AIRPORTS

"AIRPORTS" es un proyecto en el que participan varias instituciones españolas (como es el caso de la Universidad de Valladolid) y que está liderado por Boeing Research and Technology Europe (BR&T-E).

En dicho proyecto se comunican y coordinan entre sí distintas líneas de investigación que buscan desarrollar soluciones tecnológicas que contribuyan a modernizar y mejorar el transporte aéreo futuro, esto es, optimizar el uso del espacio aéreo como consecuencia del gran aumento en el número de vuelos al que nos enfrentamos en la actualidad. Una de las contribuciones más importantes de dicho proyecto en la optimización del tráfico aéreo, se basa en la explotación de las nuevas tecnologías y métodos de trabajo que surgen, como

consecuencia de los estudios realizados en el campo de Big Data.

El término Big Data se refiere a conjuntos de datos de gran tamaño, complejidad y velocidad de crecimiento lo que hace difícil su captura, gestión, tratamiento, manejo, etc. mediante el uso de las tecnologías y herramientas convencionales. Por ello, es necesario usar nuevos mecanismos y herramientas que permitan trabajar con esa gran cantidad de datos obtenidos (de esto se ocupa el campo de la ciencia de datos o data science).

Como ya se comentó, las aeronaves emiten y reciben un gran número de mensajes de tipo ADS-B. Estos mensajes provienen de diferentes proveedores abarcando cada uno de ellos una zona determinada del espacio aéreo. Dentro del conjunto de proveedores existentes se destacan OpenSky y Frambuesa, los cuales, obtienen sus datos usando la técnica ADS-B.

A continuación se detallan los aspectos más destacados de cada uno de ellos haciendo uso de las páginas web indicadas para cada uno de los proveedores considerados.

OpenSky: Es una asociación que tiene su sede en Burgdorf (Suiza). OpenSky Network comenzó en el 2012 como un proyecto de investigación entre Suiza, Alemania y Reino Unido, pero hasta el 2015 no se fundó la asociación.

Su objetivo es mejorar la seguridad, confiabilidad y eficiencia del espacio aéreo, y para ello, se encarga de adquirir, recopilar, procesar y registrar datos sobre el control del tráfico aéreo. No tiene propósitos comerciales, por lo que el acceso a dichas fuentes de datos es gratuito para investigaciones realizadas en instituciones académicas y gubernamentales. Para lograr su objetivo cuenta con una red de sensores (aproximadamente 1000), donde la mayoría se encuentran distribuidos entre Europa y EE.UU. Dicho proveedor destaca por la calidad de los datos que obtiene. La información está disponible en: https://opensky-network.org/.

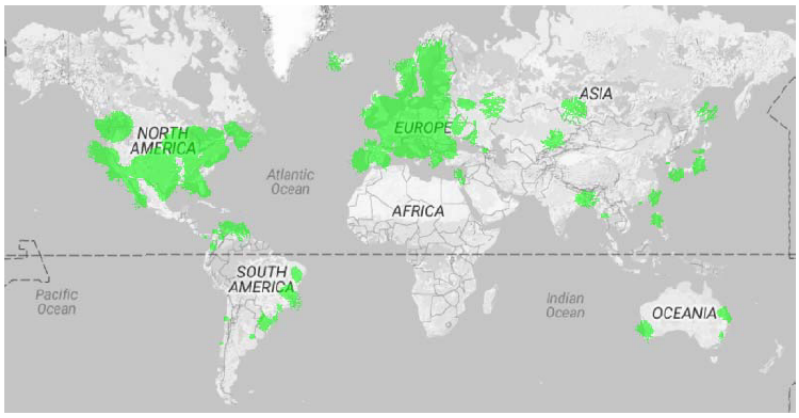


Figura 3.3: Cobertura del proveedor OpenSky

Frambuesa: Se trata de un sensor con propiedad de BR&T-E que trabaja en el aeropuerto de Madrid-Barajas.

Frambuesa ofrece mensajes ADS-B en bruto. Dichos mensajes enviados permiten obtener descripciones de vectores de posición de las aeronaves, siendo en este caso, mucho más reducidos que para el caso del resto de proveedores. Además, dicho servicio abarca una área mucho más reducida que el resto de proveedores, pero tiene la ventaja de ofrecer una mayor densidad de mensajes dentro del área cercana al aeropuerto de Madrid-Barajas. También, presenta una mayor rapidez entregando mensajes que OpenSky pues manda 1 mensaje por cada segundo, mientras que OpenSky manda 1 por cada 5 segundos.

Figura 3.4: Cobertura del proveedor Frambuesa

En muchas ocasiones suele ocurrir que una zona del espacio aéreo esté cubierta por diferentes proveedores por lo que se reciben múltiples mensajes ADS-B, y en consecuencia, surge la necesidad de fusionarlos y coordinarlos. Gracias al desarrollo en el campo de Big Data y a su contribución, es posible tratar todos esos datos obtenidos, logrando mejorar la calidad y el valor de los mismos. Así, se consigue una mayor fiabilidad de los datos resultantes.

3.4. Problema a considerar

Como ya se ha indicado, el presente trabajo busca resolver el problema sobre el alineamiento temporal de las trayectorias de las aeronaves, que son construidas a partir de las señales que estas transmiten. Dicho problema puede formularse como sigue:

Problema: *La construcción realizada sobre las trayectorias voladas por las aeronaves, teniendo en cuenta los datos recibidos sobre la posición (altitud, longitud y latitud), velocidad, etc. así como el tiempo asignado en los mensajes ADS-B, no resulta ser del todo correcta.*

Motivos: El principal motivo se debe a que el tiempo horario que es asignado a cada uno de los mensajes ADS-B recibidos sobre la posición, (en términos de latitud, longitud y altitud), identificación, etc. de cada una de las aeronaves que se conoce como timestamp, no es el correcto. Dicho inconveniente surge por ser dicho timestamp asignado por la entidad receptora y no por la aeronave que envía el mensaje.

Así, lo que ocurre, es que aunque los mensajes ADS-B son enviados por la aeronave en el momento adecuado y siguiendo el orden correcto, estos no llegan a la entidad receptora en ese mismo orden. Esto es debido principalmente al retardo en el tiempo de recepción de dichos mensajes y a la falta de sincronización horaria que existe entre algunas entidades receptoras.

Consecuencias: Por lo tanto, como es la entidad receptora la encargada de asociar el timestamp a cada uno de dichos mensajes ADS-B que recibe, al llegar en orden incorrecto, el tiempo que les asigna es erróneo provocando que los mensajes se registren en un orden inadecuado. Así, las trayectorias construidas teniendo en cuenta los datos presentes en dichos mensajes, resultan ser erróneas.

Esto se puede observar analizando algunas de las trayectorias construidas a partir de las fuentes de datos ADS-B del proyecto "AIRPORTS". En la figura 3.5 se muestra el problema comentado con anterioridad.

Figura 3.5: Problema planteado sobre el alineamiento temporal

También es posible que en las trayectorias se encuentre otro tipo de problemas, los cuales, son debidos a la presencia de *outlayers*. Sobre este tipo de problemas ya se está trabajando para su resolución y su aparición en las distintas trayectorias se ha reducido de manera considerable. Aunque este no será un problema que tengamos que tratar conviene mencionarlo, ya que a simple vista, podría ser confundido con el problema asociado a la asignación de tiempos que motiva la realización de este proyecto.

Por lo tanto, uno de los principales objetivos del desarrollo de este trabajo, consiste en analizar las trayectorias construidas a partir de los datos proporcionados por diversas

fuentes de datos ADS-B, y una vez determinadas aquellas que presentan alguna anomalía por asignación de tiempos, corregirlas mediante la aplicación de una solución escalable basada en la ejecución de ciertos algoritmos.

Como ya se comentó, dicho problema tiene una relación directa con el problema del viajante o TSP, por lo que en la siguiente sección se realizará una descripción y explicación del mismo, para así poder formular de manera teórica el problema que se pretende resolver sobre la reconstrucción de las trayectorias. Con ello se tendrá el conocimiento necesario para abordarlo y buscar una solución práctica para resolverlo.

Capítulo 4

Traveling Salesman Problem (TSP)

4.1. Definición

El problema del viajante también conocido como "Traveling Salesman Problem (TSP)", consiste en determinar la ruta más corta posible que recorre un conjunto de ciudades (de manera general nodos), de manera que el nodo final coincida con el nodo de partida y que todas las ciudades sean visitadas una única vez.

Dicho problema ha sido estudiado durante muchos años y aún es objeto de estudio dentro de la optimización combinatoria pues, aunque aparentemente parezca un problema fácil de resolver debido a que el número de posibles caminos que existe entre un conjunto de nodos sea finito, en realidad no lo es. Se trata de un problema complejo de resolver, de hecho, es un problema de tipo NP-Duro. Por ello, incluso se considera la posibilidad de no llegar nunca a encontrar un algoritmo que, en todas las situaciones posibles, encuentre la solución óptima.

Para desarrollar el contenido de los apartados de esta sección se ha usado el libro de William J.Cook [1] y el de E.L. Lawler y otros [2].

4.2. Historia

El estudio del problema del viajante surgió hace muchos años y es debido principalmente a la gran utilidad que presenta en las situaciones de la vida real estando muy relacionado con la logística, la distribución de productos, el transporte, etc.

En el año 1832 se dio a conocer en Alemania el primer libro sobre este tema denominado *"El viajante de comercio: cómo debe ser y qué debe hacer para conseguir comisiones y triunfar en su negocio. Por un viajante de comercio veterano*".

Durante los siguientes años, muchos matemáticos se dedicaron a investigar sobre dicho problema, sin embargo, no fue hasta el año 1930 cuando fue definido formalmente en términos matemáticos. Dicha formulación fue realizada por el matemático y economista

Karl Menger quien consideró en un primer momento como método de resolución la fuerza bruta, aunque pronto observó que no era un método eficiente y menos aún óptimo.

Poco después Hassler Whitney dio a conocer dicho problema con el término anglosajón "Traveling Salesman Problem" y, poco a poco, en las décadas de los 50 y 60 dicho problema ganó mucha popularidad como consecuencia principalmente de la publicidad llevada a cabo por Procter and Gamble en 1962. En ella, se propuso un concurso donde se podía obtener un premio de 10.000 dólares si se resolvía el problema del viajante para un conjunto de 33 ciudades de EE.UU.

Resultó ser un problema complejo de resolver y, aunque nadie logró llevarse el premio, se pudo demostrar que ya en el año 1954, esto es, 8 años antes, tres matemáticos de Rand Corp. (George Dantzing, Ray Fulkerson y Selmer Johnson) habían encontrado la solución óptima para un conjunto de exactamente 49 ciudades. Para ello desarrollaron el método de los Planos de Corte y lo aplicaron a su estudio obteniendo resultados muy positivos.

Este resultado fue un gran avance y, de hecho, supuso un reto a superar que no se logró hasta 1971, esto es, 17 años después, cuando los investigadores de IBM Michael Held y Richard Karp resolvieron el problema para el caso de 64 ciudades distribuidas al azar en una región cuadrada, donde los costos eran considerados como la distancia en línea recta entre cada par de ellas.

Cuatro años más tarde, concretamente en 1975, Panagiotis Miliotis logró la solución óptima para el caso de 80 puntos distribuidos de manera aleatoria.

Ya en 1977 Grötschel publicó su Tesis Doctoral en la que determinaba la solución óptima para el caso de 120 ciudades. Fue entonces cuando se asociaron Padberg y el investigador de IBM Harlan Crowder obteniendo la solución óptima para el problema de 318 ciudades distribuidas en un tablero de circuitos. La ocurrencia de todos estos sucesos fueron muy relevantes en la historia del TSP y dieron lugar a un gran avance en el desarrollo de dicho problema, ya que, Grötschel y Padberg de manera independiente lograron hallar la solución óptima para el caso de 532 ciudades en Estados Unidos, 666 localizaciones en el mundo, 1.002 ciudades con problemas de perforación, y posteriormente, de 2.392 ciudades.

Más tarde en 1988, como consecuencia de los éxitos ocurridos, Vasek Chvátal y William J.Cook se unieron en el estudio del problema logrando en el año 1992 resolverlo para el caso de 3.038 ciudades, para lo que hicieron uso de una amplia red de computadoras que trabajaban en paralelo. Siguiendo con ello, lograron en 1998 encontrar la ruta óptima de 13.509 ciudades en Estados Unidos, otra de 24.978 en Suecia en el año 2004, y finalmente, otra en 2006 de 85.900 ciudades.

Todos estos avances en el estudio y desarrollo del problema del viajante fueron posibles gracias al uso de una herramienta informática denominada *Concorde* que se comenzó a usar ya en el año 1992. Se trata de un programa en *C* muy usado actualmente (intentando su mejora y avance) para este tipo de problemas de optimización de redes y, que en años

anteriores, supuso una gran revolución en el avance del TSP.

En la Figura 4.1 se puede ver el avance producido gracias al uso de la herramienta *Concorde* puesto que se pasó muy rápidamente de encontrar la ruta óptima para el caso de 33 ciudades (se corresponde con la ruta negra) y de 120 ciudades (se corresponde con la azul) a lograr la solución óptima para el caso de 15112 ciudades (se corresponde con la ruta roja).

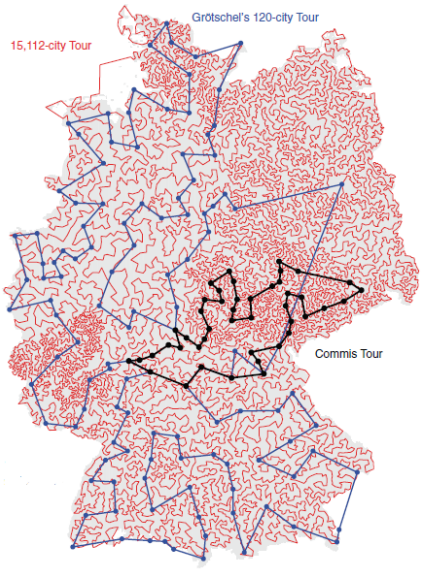


Figura 4.1: Tres recorridos distintos en Alemania ([1, pág. 14])

En la Figura 4.2 se muestra de manera gráfica la evolución en el avance de resultados obtenidos a medida que aumenta el número de ciudades consideradas. Se aprecia como a partir del 2006 dicho aumento es exponencial.

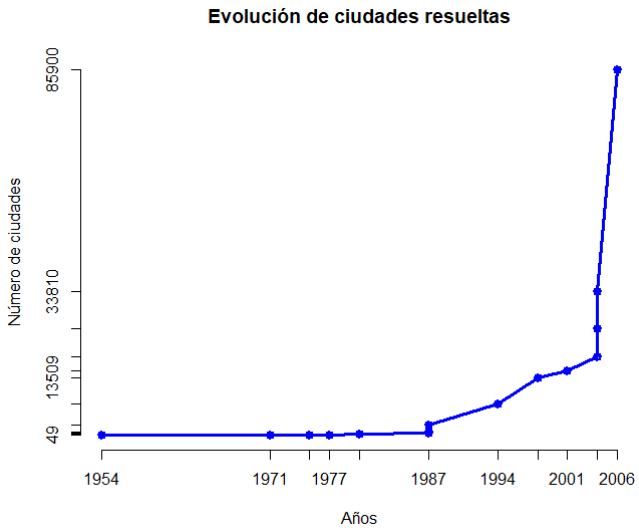


Figura 4.2: Evolución de ciudades resueltas a lo largo de los años

4.3. Aplicaciones

El problema del viajante o TSP ha sido y sigue siendo uno de los principales problemas de estudio en el desarrollo del diseño de algoritmos y de la teoría de complejidad computacional, debido principalmente a su contribución y aplicación en diferentes áreas para mejorar y resolver diversas situaciones y problemas de la vida diaria. Las principales áreas de aplicación son la logística y distribución, así como la programación de curvas de producción.

Al inicio, las mejoras del TSP tenían como objetivo conseguir aplicarlo de manera directa, como por ejemplo en rutas de autobuses escolares o de una empresa de lavandería. Poco a poco el ámbito de aplicación fue creciendo y hoy en día incluso es útil para problemas en el ámbito genético (creación de clusters de genes) y biológico (creación de árboles filogenéticos).

Se detallan a continuación las aplicaciones más importantes dentro del área de logística así como en la industria:

Logística: El TSP tiene aplicaciones muy abundantes en logística como son las siguientes.

- **Vendedores, Turistas**: Suelen usar algún tipo de sistema para planificar las rutas turísticas de manera que están sean óptimas tanto en tiempo como en coste volviendo al punto de partida. Dichos planificadores suelen basarse en algoritmos de resolución de tipo TSP.
- **Rutas escolares y laborales**: Al igual que en el caso anterior se determinan rutas mediante algoritmos de tipo TSP para ahorrar costes y tiempo.
- **Transporte de paquetes y mercancías**: Este tipo de problemas se suele adaptar mejor a problemas de arcos en lugar de problemas de nodos como es el caso del TSP pero, sin embargo resulta útil cuando las distancias entre lugares a repartir son lejanas o sólo se desea visitar un lugar concreto.

Sector Industrial: Aunque las aplicaciones del TSP en la industria son menos abundantes también son importantes y se podrían considerar las siguientes.

- **Secuenciación de las tareas**: Consiste en realizar diversas tareas de la manera más rápida posible minimizando el costo de su producción, y para ello, el orden de su realización debe ser independiente entre las mismas. En este caso cada una de las tareas se asemeja al papel de una ciudad y el tiempo que se tarda en realizar una tarea i habiendo hecho antes la tarea j es lo que equivale a la distancia entre las ciudades.
- **Producción de placas de circuitos electrónicos**: Consiste en realizar agujeros en una placa mediante la perforación automática de la misma y, para ello se usa la técnica del TSP donde se consideran cada uno de los puntos a perforar como una ciudad diferente de manera que el tiempo en crear dichas placas se reduce al mínimo posible.

Existen más aplicaciones del TSP, aunque algunas de ellas tienen una relación menos intuitiva con dicho problema, ya que no requieren de movimientos físicos. Algunas de dichas aplicaciones es el caso de la búsqueda de planetas o la organización de datos en diferentes grupos, lo cual, es usado tanto en la minería de datos como para la extracción de patrones en los datos.

4.4. Complejidad

La clasificación de los problemas a resolver puede realizarse según su complejidad y, según dicho criterio se distingue entre problemas NP, problemas P y problemas NP-Completos de forma que, hasta el momento, los problemas de tipo P y NP-Completos tienen intersección vacía siendo ambos problemas de tipo NP.

Un problema de tipo P es aquel que se resuelve en tiempo polinomial por un algoritmo determinista (ej: una máquina de Turing determinista), mientras que un problema NP es aquel que se resuelve en tiempo polinomial por un algoritmo no determinista (ej: una máquina de Turing no determinista).

Los problemas de tipo NP-Completo son problemas de tipo NP-Duro que están contenidos en la clase de problemas NP, siendo x un problema NP-Duro si cualquier problema que pertenezca a la clase de problemas NP puede reducirse en tiempo polinomial a x.

En la Figura 4.3 se muestra un gráfico de la relación existente entre este tipo de problemas para que sea más clara y visual. Para ello se ha considerado que P $\neq$ NP ya que el resultado contrario hasta el momento no ha sido probado.

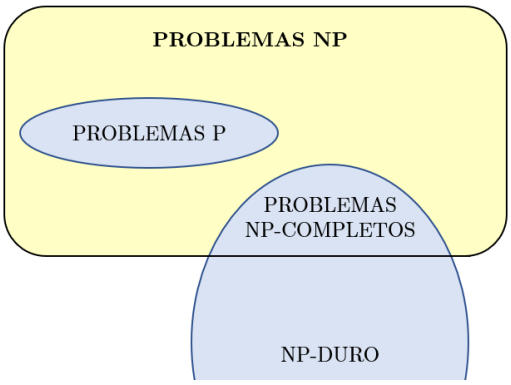


Figura 4.3: Diagrama de Venn siendo P $\neq$ NP

Como ya se comentó, el problema del viajante o TSP es considerado a día de hoy como un problema NP-Duro, luego, todo problema que pertenezca a la clase de problemas de tipo NP puede transformarse en tiempo polinomial en él.

Frente a estos conceptos surge la cuestión de encontrar algún problema NP-Completo que sea de tipo P ya que, en ese caso, se obtendrá que P = NP. Dicho resultado se dio a

conocer por Stephen Cook en 1971 [1, pág. 9] y todavía no ha sido posible su demostración.

Con respecto al problema del viajante, al ser un problema de tipo NP-Completo, desde hace años se está estudiando si se puede resolver en tiempo polinomial para poder con ello demostrar que P = NP. Esto, que aún no se ha podido determinar, tiene una gran importancia, ya que en ese caso tendríamos que todos los problemas NP-Completos y por lo tanto todos los problemas NP podrían ser resueltos en tiempo polinomial, permitiendo resolver muchos problemas que hoy en día son intratables.

Por lo tanto, aunque a simple vista el TSP parezca un problema sencillo, hoy en día aún no existe un algoritmo eficiente para resolverlo, pero como se verá, sí existen buenos algoritmos que obtienen soluciones muy próximas a la óptima en un tiempo razonable.

4.5. Teoría de Grafos

Es una de las ramas matemáticas y de las ciencias de computación que se centra en estudiar las propiedades de los grafos.

Para poder entender algunos de los algoritmos propuestos para resolver el TSP es necesario conocer antes los conceptos fundamentales de *Teoría de Grafos* y, en este apartado, se hace una pequeña introducción a alguno de ellos. Para ello, se ha usado principalmente el libro de A. Caicedo, G. Wagner y R.M Méndez [6] junto con la referencia [5].

Definición 4.1. *Un grafo denotado por $G = (V, A)$ es una pareja ordenada en la que V es un conjunto de vértices no vacío $\{v_0, ..., v_n\}$ y A es el conjunto de aristas o arcos entre dichos vértices, esto es, A está formado por pares no ordenados de vértices de la forma $a_{ij} = (v_i, v_j) \in A$ siendo $i \neq j$.*

Las aristas son las líneas que unen un par de vértices y pueden ser de varios tipos:

- *Adyacentes*: Aquellas que convergen en el mismo vértice.
- *Paralelas*: Aquellas aristas conjuntas para las que el vértice inicial y final es el mismo.
- *Cíclicas*: Aquellas que parten de un vértice y entran en sí mismo.
- *Cruzadas*: Aquellas que se cruzan en el mismo punto.

Además, también existen diferentes tipos de grafos, siendo algunos los siguientes:

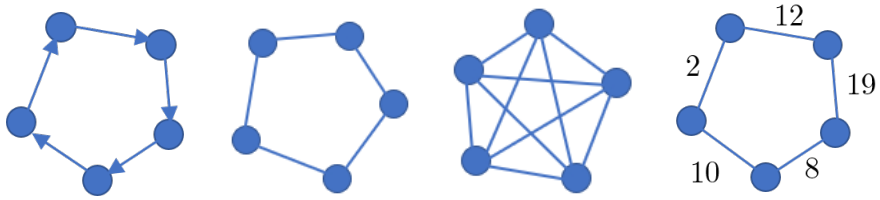
Definición 4.1.1 (Grafo dirigido u orientado). *Se suele denominar dígrafo y es aquel en el que las aristas son pares ordenados, esto es, $a_{ij} \neq a_{ji}$, $\forall\ i \neq j$. Dichas aristas se suelen representar con una flecha que va del vértice de partida al vértice final.*

Definición 4.1.2 (Grafo no dirigido o no orientado). *Es aquel en el que los pares a_{ij} y a_{ji} se corresponden con la misma arista, esto es, las aristas no tienen dirección. Será el tipo de grafos que se considerarán en el trabajo.*

Definición 4.1.3 (Grafo completo). *Es aquel en el que aparecen trazadas todas las posibles aristas.*

Definición 4.1.4 (Grafo etiquetado). *Es aquel en el que las aristas tienen un determinado peso asociado. Por ejemplo en el caso de las ciudades sería la distancia que hay entre cada par de ellas.*

En la Figura 4.4 se muestra un ejemplo de cada uno de ellos.



(a) Dirigido (b) No dirigido (c) Completo (d) Etiquetado

Figura 4.4: Tipos de grafos

A continuación se detallan unos conceptos que serán utilizados para describir el problema del viajante o TSP en términos matemáticos.

Definición 4.2. *Un camino es un conjunto de vértices conectados a través de aristas que van de un vértice inicial a un vértice final. Se trata de sucesiones en las que aparecen elementos de V y A de forma alternativa, esto es, $v_0, a_{01}, v_1, ..., v_{n-1}, a_{n-1n}, v_n$ donde $v_i \in V$, $\forall\ i = 0, ..., n$ y $a_{ij} \in A$, $\forall\ i = 0, ..., n-1$ y $j = i+1$. En el caso en el que el nodo o vértice inicial coincida con el vértice final, esto es, $v_0 = v_n$ se denomina circuito, ciclo o camino cerrado.*

Existen diferentes tipos de caminos y circuitos entre los que consideramos los siguientes:

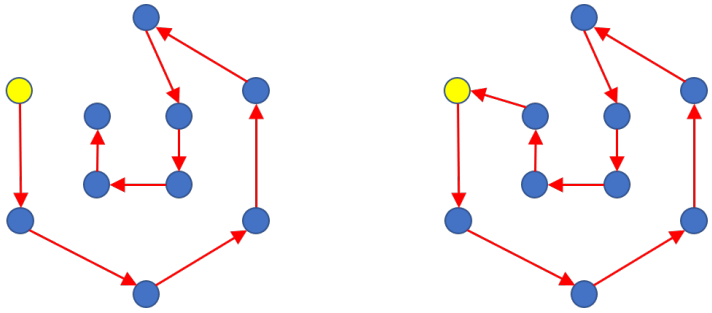
Definición 4.2.1 (Camino hamiltoniano). *Es aquel camino que pasa por todos los nodos del grafo en una única ocasión. Más formalmente es una secuencia de la forma $v_0 a_{01} ... a_{n-1n} v_n$ donde $v_i \neq v_j$, $\forall\ i, j = 0, ..., n$ con $i \neq j$, $v_i \in V$ y $a_{ij} \in A$, $\forall\ i = 0, ..., n-1$ y $j = i+1$.*

Definición 4.2.2 (Camino euleriano). *Es aquel camino que recorre todas las aristas del grafo en una única ocasión. Más formalmente es una secuencia de la forma $v_0 a_{01} ... a_{n-1n} v_n$ donde $a_{ij} \neq a_{ji}$, $\forall\ i = 0, ..., n-1$ y $j = i+1$, $v_i \in V$ y $a_{ij} \in A$, $\forall\ i = 0, ..., n-1$ y $j = i+1$.*

Definición 4.2.3 (Circuito hamiltoniano). *Es aquel circuito que pasa por todos los nodos del grafo en una única ocasión. Más formalmente es una secuencia de la forma $v_0 a_{01} ... a_{n-1n} v_n$ donde $v_0 = v_n$, $v_i \neq v_j$, $\forall\ i, j = 1, ..., n-1$ con $i \neq j$, $v_i \in V$ y $a_{ij} \in A$, $\forall\ i = 0, ..., n-1$ y $j = i+1$.*

Definición 4.2.4 (Circuito euleriano). *Es aquel circuito que recorre todas las aristas del grafo en una única ocasión. Más formalmente es una secuencia de la forma $v_0 a_{01} ... a_{n-1n} v_n$ donde $v_0 = v_n$, $a_{ij} \neq a_{ji}$, $\forall\ i = 0, ..., n-1$ y $j = i+1$, $v_i \in V$ y $a_{ij} \in A$, $\forall\ i = 0, ..., n-1$ y $j = i+1$.*

En la Figura 4.5 se muestra un ejemplo de camino y circuito hamiltoniano, ya que como será considerado en todo el documento, merece la pena que quede claro. El nodo de color amarillo es el de comienzo.



(a) Camino (b) Ciclo o circuito

Figura 4.5: Camino y circuito hamiltoniano

A partir de lo anteriormente descrito, se puede observar la relación existente entre la *Teoría de Grafos* y el TSP, de manera que el TSP puede describirse como un problema de *Teoría de Grafos* donde el objetivo es encontrar un circuito hamiltoniano de coste mínimo.

Para ello, se debe considerar un grafo completo $G = (V, A)$ donde V denota el conjunto de ciudades a visitar y A son las aristas entre ellas. Como ya se comentó, dicho grafo debe estar etiquetado, esto es, a cada arista de unión entre dos ciudades se la asocia un peso que, en este caso, es una distancia de la forma $d_{v_i v_j}$ (distancia de la ciudad en el vértice v_i a la ciudad en el vértice v_j). Por lo tanto, lo que se busca es encontrar en dicho grafo un circuito que sea hamiltoniano y de coste el menor posible, pues así, todas las ciudades se visitan exactamente una sola vez y con la menor distancia posible.

Todo ello nos lleva a considerar dos tipos de problemas TSP:

Definición 4.3 (TSP simétrico). *Es el que se considera en el trabajo y se basa en que G es un grafo no dirigido, por lo que $d_{v_i v_j} = d_{v_j v_i}$ para todo $(v_i, v_j) \in A$ con $i \neq j$, esto es, la distancia entre dos ciudades es la misma independientemente de la ciudad de partida y de destino.*

En términos matemáticos tendríamos que minimizar la función objetivo siguiente:

$$\sum_i \sum_j d_{v_i v_j} x_{ij}$$

donde x_{ij} toma el valor 0 si la arista (v_i, v_j) no forma parte de la solución y 1 en caso contrario. Además se tienen una serie de restricciones como son:

$$\sum_i x_{ij} = 1 \quad \forall\, j, \quad \sum_j x_{ij} = 1 \quad \forall\, i$$

De esta manera, aseguramos que sólo una arista puede entrar en un vértice del grafo (desde cada ciudad v_j sólo se puede llegar a una ciudad v_i) y, que sólo una arista puede

salir de cada vértice (desde cada ciudad v_i sólo se puede llegar a una ciudad v_j). Así, se logra que cada ciudad sea visitada en una única ocasión.

Sin embargo, con esto no es suficiente, ya que se necesita considerar una serie de restricciones a mayores para evitar que haya subciclos. Sea W un subconjunto de vértices del conjunto V considerado. Considerando los siguientes conjuntos,

$$A(W) = \{a_{ij} = (v_i, v_j) \in A : v_i, v_j \in W\}$$
$$\delta^-(W) = \{a_{ij} = (v_i, v_j) \in A : v_i \notin W, v_j \in W\}$$
$$\delta^+(W) = \{a_{ij} = (v_i, v_j) \in A : v_i \in W, v_j \notin W\}$$

se tiene que satisfacer la condición siguiente:

$$\sum_{(v_i,v_j)\in A(W)} x_{ij} \leq |W| - 1\,, \quad \forall\ W \subset V \quad \equiv \quad \sum_{v_i \in W, v_j \notin W} x_{ij} \geq 1\,, \quad \forall\ W \subset V$$

Las condiciones anteriores son equivalentes y denotan que en todo subconjunto de nodos debe haber al menos un arco que salga del mismo para así evitar subciclos.

En la Figura 4.6 se muestra una situación en la que aparecen dos subciclos. Como se puede ver, en este caso no se satisfacen las condiciones dadas con anterioridad para evitar que aparezcan. Si exigimos que se verifiquen, entonces esta situación no podría darse ya que se tendrían que cumplir las siguientes restricciones:

- **Para el subciclo $W = \{v_0, v_2\}$**: Se tiene que $x_{02} + x_{20} \leq 2 - 1 = 1$ o bien que $x_{01} + x_{03} + x_{05} \geq 1$ y $x_{21} + x_{23} + x_{25} \geq 1$.
- **Para el subciclo $W = \{v_1, v_3, v_5\}$**: Se tiene que $x_{13} + x_{35} + x_{51} \leq 3 - 1 = 2$ o bien que $x_{10} + x_{12} \geq 1$, $x_{30} + x_{32} \geq 1$ y $x_{50} + x_{52} \geq 1$.

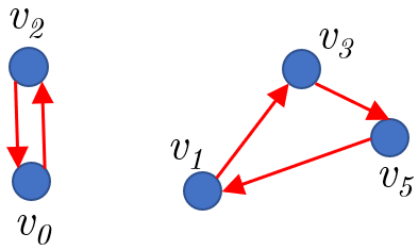


Figura 4.6: Subciclos en el TSP

De esta manera quedaría descrito el problema del viajante de comercio en términos matemáticos.

Definición 4.4 (TSP asimétrico). *En este caso G es un grafo dirigido. No se considera en el trabajo ya que no es de interés para resolver el problema de las trayectorias que lo motiva.*

Para resolver el problema del TSP se plantean varias alternativas. Una de ellas, es buscar un método o algoritmo que permita determinar la solución óptima del problema planteado mediante la enumeración de todas las posibles soluciones, pero, hasta el momento, esta posibilidad no ha resultado factible para el caso en que se tienen más de 20 vértices. Por lo tanto, se ha optado por una segunda alternativa que se basa en buscar un método con el que se obtenga una primera ruta factible, y tras ello, mejorar el coste de la misma intentando así estar lo más cerca posible de la solución óptima, esto es, se buscan soluciones aproximadas. Estos métodos se denominan heurísticos y se caracterizan por ser eficientes, buenos (solución próxima a la óptima) y robustos (probabilidad de que la solución no sea próxima a la óptima es baja). Así, mediante su uso se logra encontrar una solución factible o satisfactoria del problema de manera bastante rápida y eficiente.

4.6. Heurísticas para encontrar una ruta factible

Existen diversos métodos heurísticos usados para resolver el TSP que conviene mencionar. Se puede distinguir entre heurísticas constructivas, heurísticas de mejora, así como otros. Todos ellos buscan acercarse lo más posible a la solución óptima.

4.6.1. Heurísticas constructivas

Existen diversas heurísticas constructivas que se basan en obtener una solución factible teniendo en cuenta un determinado criterio. Se consideran las siguientes:

4.6.1.1. Heurística del vecino más próximo

Este método busca encontrar un circuito hamiltoniano de coste mínimo apoyándose en la idea de elegir siempre el vértice del grafo que esté más cercano del actual (aristas de menor coste) y no haya sido visitado. Una vez visitados todos los vértices se termina uniendo el último vértice visitado con el de partida. Por ir siempre en busca de lo mejor, se suele considerar un método ambicioso o voraz.

De manera genérica, supongamos que se tiene un grafo $G = (V, A)$ de n vértices descrito según la Sección 4.5. Los pasos a seguir son los siguientes:

Algoritmo 1 Vecino más próximo

1: Se selecciona un vértice v_i del grafo al azar donde $i \in \{0, ..., n\}$
2: Se considera $p = i$ y el subgrafo $W = V \setminus \{i\}$
3: **mientras** $W \neq \emptyset$ **hacer**
4: Se selecciona un vértice $v_i \in W$ tal que $d_{pi} = min\{d_{pj} : \forall\, j \in W\}$
5: Se une el vértice v_p con v_i
6: Se considera $W = W \setminus \{i\}$ y $p = i$
7: **fin mientras**

En la Figura 4.7 se muestra un ejemplo muy sencillo para ilustrar el funcionamiento del método anterior. Para ello, se indica en color amarillo el nodo de partida y de llegada.

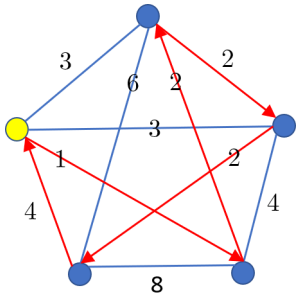
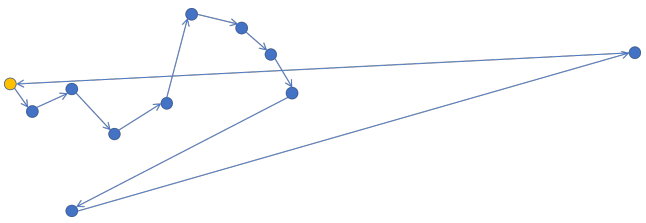


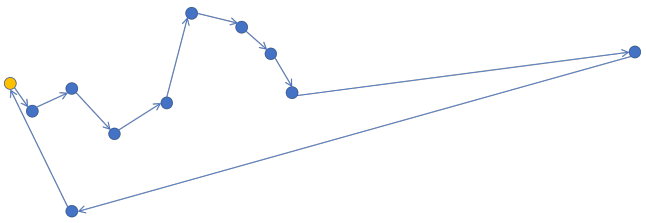
Figura 4.7: Ejemplo vecino más próximo

En el caso del ejemplo anterior el camino encontrado es el óptimo pero no siempre es así. Para situaciones en las que se tienen pocos nodos o ciudades resulta útil debido a su sencillez y rapidez pero, en general, no es recomendable pues lo habitual es tener una gran cantidad de nodos, y en este caso, suele obtener recorridos bastante malos con respecto a la solución óptima.

Una de las principales consecuencias por la que se obtienen recorridos lejanos al óptimo se debe a que cuando se realiza la elección de los nodos sólo se tiene en cuenta el nodo que se encuentra más próximo. Esto provoca que la distancia entre el último nodo considerado y el de partida sea demasiado grande, haciendo ineficiente dicho algoritmo. Si el objetivo fuera considerar un camino hamiltoniano en lugar de un circuito hamiltoniano dicho método resultaría muy adecuado, ya que, en la mayoría de los casos se encontraría la solución óptima o una muy próxima a la misma.



(a) Método del vecino más próximo



(b) Posible mejor solución

Figura 4.8: Heurística del vecino más próximo

En la Figura 4.8 se muestra el problema descrito con anterioridad. Se considera el coste de las aristas como la distancia existente entre pares de vértices, siendo el nodo amarillo el de partida. Es un ejemplo muy sencillo donde se puede ver cómo esta situación afecta de manera negativa para encontrar la ruta óptima.

4.6.1.2. Heurística de inserción

Son un conjunto de métodos que se basan en construir circuitos usando un determinado conjunto de vértices, y posteriormente, se van insertando uno a uno los restantes vértices en dicho circuito hasta formar un circuito hamiltoniano.

De manera genérica, supongamos que se tiene un grafo $G = (V, A)$ de n vértices descrito según la Sección 4.5. Los pasos a seguir son los siguientes:

Algoritmo 2 Heurística de inserción

1: Se selecciona un conjunto inicial de j vértices
2: Se considera el subgrafo $W = V \setminus \{\text{vértices seleccionados del conjunto}\}$
3: **mientras** $W \neq \emptyset$ **hacer**
4: Se considera un vértice $v_i \in W$ según un determinado método (se explican a continuación los más usados)
5: Se inserta dicho vértice v_i de manera que el coste del circuito se incremente lo menos posible
6: Se considera $W = W \setminus \{i\}$
7: **fin mientras**

Existen diferentes métodos de inserción según el criterio usado para añadir los nodos. Estos fueron descritos por Robacker y son los siguientes:

Definición 4.5 (**Inserción más cercana**). *Consiste en elegir la ciudad o vértice v_i más cercana a las ciudades del circuito actual, esto es, si W es el circuito actual, $d_{min}(v_i) = min\{d_{min}(v_j) : v_j \in W\}$.*

Definición 4.6 (**Inserción más lejana**). *Consiste en elegir la ciudad o vértice v_i más alejada de las ciudades del circuito actual, esto es, si W es el circuito actual, $d_{min}(v_i) = max\{d_{min}(v_j) : v_j \in W\}$.*

Definición 4.7 (**Inserción más aleatoria**). *Consiste en elegir la ciudad o vértice v_i al azar, esto es, sin seguir ningún criterio.*

Definición 4.8 (**Inserción más barata**). *Consiste en elegir la ciudad o vértice v_i que produce el menor incremento de coste posible, esto es, que mantiene el circuito existente lo más corto posible.*

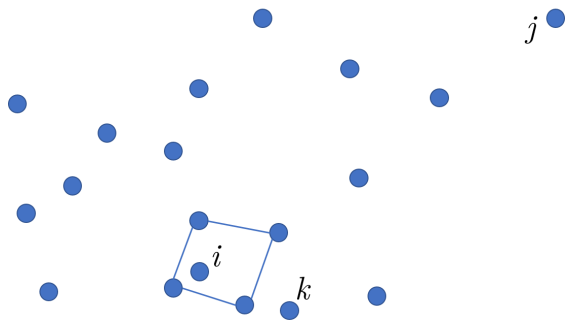


Figura 4.9: Elecciones según el método elegido

En la Figura 4.9 se muestran las diferentes elecciones de nodos según el método de inserción más lejano, más cercano y más barato para insertar al ciclo de 4 vértices actual. Para el caso de la inserción más lejana habría que añadir el nodo j al actual circuito, en el caso más cercano el nodo k y en el caso más barato el nodo i, pues con él se obtiene el circuito menor posible a partir del actual. Si se considera el tipo aleatorio, entonces se elegiría un nodo al azar entre los existentes.

4.6.1.3. Heurística de Christofides

Fue creada por Christofides en 1976 ([1, pág. 72]) y está muy relacionada con los árboles de coste mínimo, por ello, antes de explicar el método es necesario entender este tipo de árboles.

Un árbol de coste mínimo de un grafo es un subgrafo que es un árbol y, además, contiene todos los vértices del grafo inicial con el mínimo coste posible. La búsqueda de este tipo de árboles es un problema que se puede resolver en tiempo polinomial mediante algoritmos eficientes a diferencia de lo que ocurre con el TSP. Por ello, su uso es útil, ya que la solución de este tipo de problemas proporciona una cota del coste de la solución óptima para el TSP que es mínima.

Esto se debe a que si de un circuito que es solución eliminamos una arista se obtiene un árbol que posee un único camino de unión entre las distintas ciudades. Por ello, como la solución óptima debe tener una arista más que el árbol anterior, ya que debe ser un circuito cerrado, el coste de dicha solución óptima va a ser necesariamente mayor que el del árbol de mínimo coste.

El método de Christofides es un algoritmo que busca soluciones aproximadas a la óptima, de manera que si el coste de la solución aproximada es x y el de la solución óptima es y, entonces $x \leq \frac{3}{2}y$. Comienza buscando el árbol de mínimo coste L de un grafo G completo y etiquetado. Tras ello se elige el conjunto de vértices de grado impar del árbol L y se halla un apareamiento perfecto M (conjunto de aristas sin vértices en común) de mínimo peso en G sobre dichos vértices considerados. Tras ello se forma un multigrafo (en él dos nodos pueden estar conectados por más de una arista) mediante la combinación de las aristas de M y L. Finalmente, se obtiene un circuito euleriano en dicho multigrafo y, quitando los nodos ya visitados, se obtiene el circuito hamiltoniano buscado.

A continuación, en la Figura 4.10 se muestra un ejemplo de lo anteriormente comentado. El primer dibujo se corresponde con el grafo completo G del cual se quiere hallar la ruta óptima. Para ello, primero se busca el árbol de mínimo coste que se muestra en el segundo dibujo, y después, tras encontrar el par de vértices de grado impar se forma el emparejamiento M a partir del grafo G sobre esos vértices. Finalmente, el último dibujo se corresponde con el multigrafo unión de L y M que como resulta ser un circuito hamiltoniano de mínimo coste termina la búsqueda.

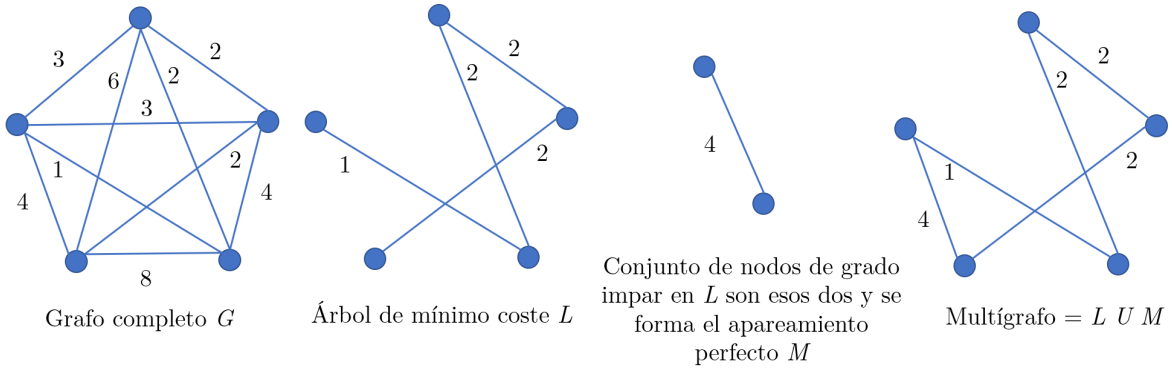


Figura 4.10: Ejemplo heurística de Christofides

4.6.2. Heurísticas de mejora

Se trata de una serie de métodos que buscan mejorar soluciones ya encontradas y, para ello, siguen diversas técnicas. Se puede distinguir entre métodos que se basan en realizar intercambios y otros que se basan en el azar o aleatoriedad.

4.6.2.1. Heurísticas de mejora k-opt o de Lin-Kernighan

Son procedimientos que consisten en intercambiar diversas aristas de una solución inicial de partida buscando mejorarla y conseguir una nueva solución más próxima a la óptima. Se conocen gracias a las primeras definiciones desarrolladas por Flood [1]. Dichos métodos se basan por lo tanto en realizar k-intercambios de aristas e ir generando rutas k-óptimas hasta que no sea posible mejorarlas más. Para entender estos métodos es necesario entender unos conceptos previos, por lo que antes de detallar cómo funcionan vamos a explicarlos.

El proceso de realizar un k-intercambio de aristas en una ruta inicial dada consiste en eliminar exactamente k aristas de dicha ruta y reemplazarlas por otras k aristas diferentes de manera que la nueva ruta obtenida sea mejor que la anterior, esto es, de menor coste. En ese caso, dicha ruta se conoce como k-óptima.

La complejidad de este tipo de métodos es $\mathcal{O}(n^k)$ (siendo n el número de nodos), ya que en cada paso, el número de posibles elecciones es $\binom{n}{k}$. Sin embargo, aunque a mayor valor de k mejores soluciones se esperan obtener, el número de operaciones a realizar crece mucho. Por ello, lo más usual es usar un valor de k no mayor que 3, pues en otro caso el coste temporal sería muy grande, no siendo recomendable.

Para que resulte más claro dicho procedimiento, se describe el caso en que $k = 2$. El proceso comienza con un ciclo hamiltoniano inicial y con el valor de la variable mejora = 1. Tras ello, mientras mejora valga 1, esto es, se encuentren soluciones mejores, se establece mejora a 0 y se van seleccionando los vértices que no han sido explorados. Para cada uno

de ellos se realizan todos los posibles movimientos de dos intercambios que incluyan a dicho vértice y uno sucesor. Si alguno de dichos intercambios reduce la distancia actual, se elige el mejor de ellos y mejora pasa a valer 1. Tras ello, dicho vértice se considera explorado y se sigue con el resto hasta que mejora vale 0 pues, en ese caso, ningún intercambio mejora la distancia actual. El coste computacional en cada paso no es grande siendo del orden de $\mathcal{O}(n^2)$.

A continuación, se muestra un ejemplo sencillo de este método que como vemos se basa en la idea de eliminar "cruces" entre aristas aunque, en la práctica, esto es difícil de visualizar. En el ejemplo de la Figura 4.11 se puede ver en el primer dibujo que las aristas en color rojo forman la ruta de partida con un coste de 24 unidades, mientras que en el segundo dibujo, al intercambiar las aristas de mayor coste (la de 8 y 6) por otras de menor coste (la de 4 y 2) se obtiene una mejor solución, siendo el coste de esta de 16 unidades.

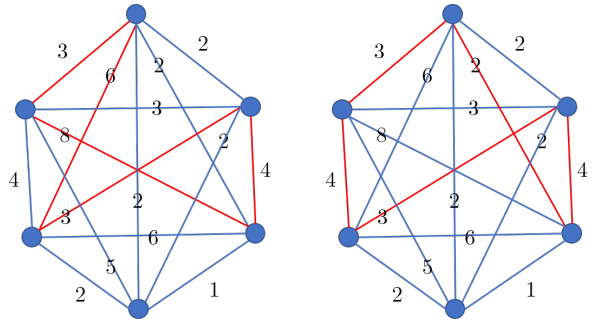


Figura 4.11: Ejemplo heurística *2-opt*

Una variante de esta heurística se denomina *V-opt* y difiere del *k-opt* en que las aristas que son eliminadas no están fijas, sino que, dicho número aumenta con el número de iteraciones que se hacen.

Dentro de esta heurística destaca el método *Lin-Kernighan*. Se trata de una de las mejores heurísticas que se conocen para resolver el problema del viajante. Consiste en ir intercambiando un número diferente de aristas según sea más conveniente en cada caso.

4.6.2.2. Heurísticas de mejora aleatorias

Estos métodos usan diversas técnicas para ir generando soluciones o rutas que estén cada vez más próximas a la ruta óptima logrando conseguir buenos resultados en un tiempo reducido.

Algoritmos Genéticos: Son métodos que se basan en simular los fenómenos naturales de evolución. Se parte de una población inicial generada de manera aleatoria (conjunto de nodos al azar) que sigue un proceso con las siguientes etapas:

- *Selección*: Consiste en elegir de la población actual aquellos descendientes que poseen las mejores características. Para ello, existe una función *fitness* que mide la calidad de cada una de las distintas alternativas.

- *Cruce*: Consiste en el traspaso de información genética entre cromosomas de los padres a los descendientes y, en el problema del TSP equivale a realizar saltos entre distintos estados del espacio de búsqueda.
- *Mutación*: Tras el cruce, cada uno de los nuevos individuos generados puede sufrir mutaciones con una determinada probabilidad p, que si es menor que la tasa de mutación (se elige en el rango $[0.001, 0.05]$), entonces se lleva a cabo dicha mutación.

Una vez realizadas estas etapas se eligen las mejores soluciones entre las existentes (las anteriores más las nuevas obtenidas) volviendo a realizar el proceso descrito. Así, se logra obtener diversas soluciones y a medida que aumentan las iteraciones están más próximas a la solución óptima. El procedimiento seguido en los algoritmos genéticos para buscar soluciones óptimas se muestra en la Figura 4.12 de manera gráfica.

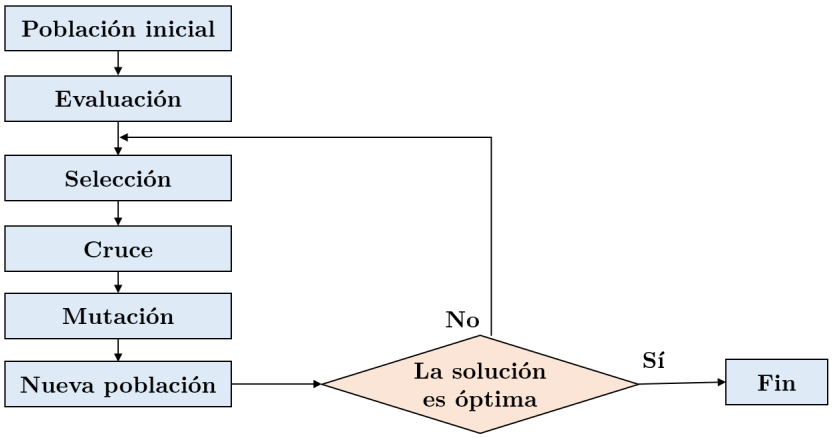


Figura 4.12: Procedimiento algoritmos genéticos

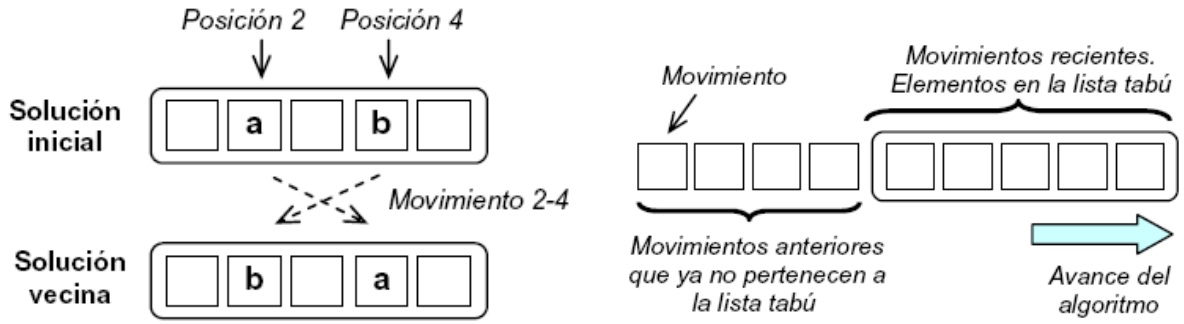
Búsqueda tabú: Es un algoritmo de búsqueda local desarrollado por Fred Glover [7] que trata de evitar que la búsqueda se quede bloqueada en óptimos locales no llegando a encontrar soluciones próximas a la óptima. Para ello, hace uso de estructuras de memoria que pueden ser a corto (denominado *lista tabú*) o largo plazo permitiendo moverse a soluciones que sean peores que la actual para poder escapar de esos óptimos locales.

En las de corto plazo, se almacenan las acciones o eventos más recientes, mientras que en las de largo plazo, se guardan los datos asociados a las frecuencias de ciertos eventos.

La memoria a corto plazo se suele denominar *lista tabú*, ya que en ella se almacenan los movimientos más recientes, denominados *movimientos tabú*, prohibiendo su elección. Así, se consigue salir de los óptimos locales evitando un ciclo repetitivo, pero para ello, se permiten peores soluciones. Estos movimientos considerados tabú pueden salir de dicha lista, cuando tras analizarlos, se observa que producen mejores resultados que los actuales. Esta técnica se conoce como *regla de aspiración*.

Por otro lado, la memoria a largo plazo es muy importante ya que trata de diversificar la búsqueda, permitiendo así, explorar zonas que no han sido visitadas con anterioridad.

En la Figura 4.13a se muestra un movimiento de intercambio, y en Figura 4.13b, cómo se van almacenando o eliminando de la *lista tabú* dichos movimientos.



(a) Movimiento de intercambio

(b) Elementos *lista tabú*

Figura 4.13: Búsqueda tabú

Colonia de hormigas: Al igual que los algoritmos genéticos se basa en imitar los procesos naturales. Consiste en estudiar el comportamiento que siguen las hormigas cuando salen del hormiguero para explorar otras regiones en busca de alimento. Después de realizar diversas observaciones, se pudo comprobar que éstas siguen el rastro de las feromonas que van dejando a lo largo del recorrido realizado.

Si se tiene un conjunto de n hormigas que salen del hormiguero, cada una de ellas hace su propio recorrido marcando cuál es el camino seguido mediante el rastro de feronomas que dejan. Como a medida que pasa el tiempo el rastro de feromonas se evapora, los caminos que son más largos tienen menos probabilidades de ser seguidos pues en ellos se reduce la fuerza de atracción que mueve a las hormigas en su elección. Este proceso de evaporación es útil ya que permite que se detenga convergiendo en óptimos locales. Por ello, cuando se encuentra un camino que es bueno, esto es, de menor distancia hay más posibilidades para que el resto de hormigas lo sigan.

Este hecho fue considerado para aplicarlo a la resolución del TSP buscando encontrar en un grafo completo el camino hamiltoniano de menor coste. En este caso, el agente que se mueve entre las ciudades juega el papel de la hormiga. Se considera que varias hormigas parten de distintas ciudades, cada una de las cuales realiza un recorrido entre las mismas, visitando una única vez cada ciudad y volviendo a la de partida. Tras ello, se van dejando feromonas en el camino, de manera que cuando el recorrido realizado es corto se potencia el rastro de feromonas en el mismo para la siguiente iteración y si es largo no. Esto hace que se evapore dicho rastro en el caso de caminos largos. Así, a medida que se realizan las iteraciones se logra potenciar los recorridos con distancias más cortas logrando estar cada vez más cerca del óptimo.

En la Figura 4.14 se muestra de manera gráfica el proceso seguido por dicho método, donde se puede ver como la mayoría de las hormigas eligen el recorrido más corto, ya que es el que tiene mayor rastro de feromonas.

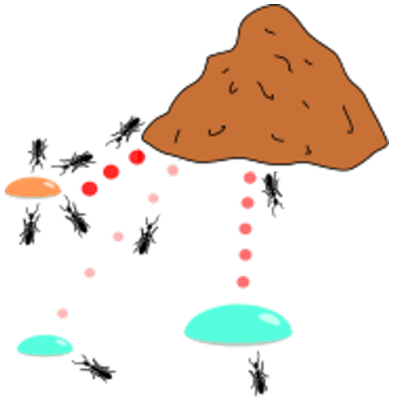


Figura 4.14: Método colonia de hormigas

Simulated Annealing: Dicho método fue descrito de manera independiente por Scott Kirkpatrick, C. Daniel Gelatt y Mario P. Vecchi, así como por Vlado Černý en los años 80 ([1, pág. 86]) y está relacionado con el campo de la termodinámica. Como dicho método es el objeto de estudio del presente trabajo, se detalla con mayor formalidad a continuación.

El algoritmo *Simulated Annealing* es una heurística que tiene como propósito encontrar un valor lo más próximo posible al valor óptimo de una función objetivo determinada. Por lo tanto, se trata de un método de optimización global que tiene la ventaja de resolver problemas de optimización genéricos, de manera rápida y eficiente, en espacios de estados (o soluciones) grandes.

A continuación, se introduce la notación que se va a usar para describir el algoritmo:

$S = \{$ soluciones posibles del problema a optimizar$\}$, siendo finito y denominado espacio de soluciones.

f: Es la función objetivo a optimizar que va del espacio de soluciones posibles a la recta real, esto es,

$$f : S \to \mathbb{R}$$

(S, f): Es un par que simboliza una determinada instancia del problema de optimización combinatoria. Para cada solución o estado $i \in S$ se considera el conjunto $S_i \subseteq S$ como el conjunto de las soluciones próximas a i, esto es, el entorno de i. Se tiene que $j \in S_i \Leftrightarrow i \in S_j$.

$S_{opt} = \{$ soluciones óptimas del problema a optimizar$\}$.

f_{opt}: Es el valor de la función objetivo en una solución óptima del problema de optimización. Por ejemplo, para el caso del problema del viajante se debe buscar $i_{opt} \in S_{opt}$ tal que $f(i_{opt}) \leq f(i)$, $\forall\ i \in S$ donde i_{opt} es una solución global óptima de dicho problema. (pues se trata de un problema de minimización)

c_k: Es el valor del parámetro de control en la iteración k o paso k-ésimo.

L_k: Es el número de transacciones o movimientos a realizar en la iteración k o paso k-ésimo.

Definición 4.9. *Una transición es un proceso formado por dos etapas donde, en la primera se aplica un mecanismo para generar una solución o estado sucesor del actual, y tras ello, se aplica el criterio de aceptación (Definición 4.10) para ver si se sigue con la solución o estado actual, o se elige el sucesor.*

Una vez que se tienen los estados, espacio de soluciones, función objetivo, etc. es necesario determinar cómo se va a llevar a cabo la aceptación de los diferentes estados. Dicho proceso sigue el siguiente criterio.

Definición 4.10 (**Criterio de aceptación**)**.** *Sea (S, f) una instancia de un problema de optimización combinatoria y sean $i, j \in S$ dos soluciones con valores o costes asociados $f(i)$, $f(j)$ respectivamente. Se define el criterio de aceptación para pasar del estado i al estado j mediante la siguiente probabilidad:*

$$\mathbb{P}_c\,(\text{aceptar j}) = \begin{cases} 1 & \text{si } f(j) \leq f(i) \\ exp\left(\frac{f(i)-f(j)}{c}\right) & \text{si } f(j) > f(i) \end{cases}, \tag{4.1}$$

donde $c \in \mathbb{R}^+$ es denominado el parámetro de control.

Es importante señalar que, a la hora de determinar la convergencia o no del algoritmo, el comportamiento de dicho parámetro c va a tener gran importancia.

Una vez que ya se tienen todos los conceptos y criterios anteriores, se describe el proceso seguido para llevar a cabo la aplicación del algoritmo *Simulated Annealing*.
Dada (S, f) una instancia de un problema de optimización combinatoria, para encontrar un valor próximo al valor óptimo de la función objetivo f, se siguen los siguientes pasos:

Algoritmo 3 Simulated Annealing

1: Se inicializa el estado inicial $i_{inicial}$, el valor inicial del parámetro de control c_0 y el número de transiciones o movimientos a realizar en la iteración inicial, esto es, L_0
2: Se fija $k = 0$, $i = i_{inicial}$
3: **mientras** no se cumpla el criterio de parada **hacer**
4: **para** $l = 1, ..., L_k$ **hacer**
5: Se genera un nuevo estado $j \in S_i$
6: **si** $f(j) \leq f(i)$ **entonces**
7: **devolver** $i = j$
8: **si no**
9: **si** $exp\left(\frac{f(i)-f(j)}{c_k}\right) > random[0, 1)$
10: **devolver** $i = j$
11: **fin si**
12: $k = k + 1$
13: Se calcula el valor L_k
14: Se calcula el valor c_k
15: **fin mientras**

Se tiene que random[0,1) es una variable aleatoria uniforme en [0,1), el paso 5 es el mecanismo de generación de nuevos estados (se eligen estados dentro del entorno del estado actual) y los pasos 6, 7, 8, 9, 10 y 11 se corresponden con el mecanismo llevado a cabo para aceptar o no dichos estados generados.

El criterio de parada usado para finalizar dicho proceso, puede venir dado fijando un número máximo de iteraciones posibles a realizar, o bien cuando tras un número de iteraciones, no se consigue mejorar el estado o solución actual.

Por lo tanto, dicho algoritmo se trata de una heurística de búsqueda local que permite llevar a cabo movimientos que lleven del estado actual a otros peores durante el inicio del proceso, y a medida que se va descendiendo de manera gradual el valor del parámetro de control c, dicha probabilidad se reduce para evitar así alejarse del valor óptimo de la función objetivo.

El interés de permitir aceptar estados peores que el actual en las primeras etapas, es lo que permite al algoritmo escapar de óptimos locales, permitiendo así explorar todo el espacio de estados S. Además, si la disminución de c se realiza lentamente, se puede garantizar que el algoritmo encuentra el óptimo global con una probabilidad cercana a 1.

Como consecuencia de ello, supone una mejora con respecto a otros algoritmos de búsqueda local, como es el caso del Hill Climbing. Este algoritmo presenta el inconveniente de quedarse bloqueado en óptimos locales por no permitir ir a soluciones peores que la actual.

En la Figura 4.15 que se muestra a continuación, se observa el inconveniente de la búsqueda local determinista, ya que por ejemplo, Hill Climbing quedaría bloqueado en el mínimo local sin llegar a alcanzar el global.

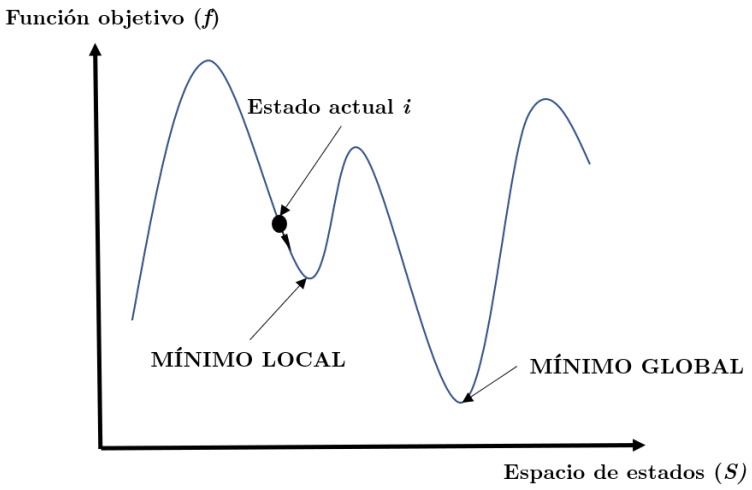


Figura 4.15: Problema de la búsqueda determinista

La demostración de la convergencia asintótica del algoritmo *Simulated Annealing* de una función f al conjunto de soluciones óptimas S_{opt} se basa en probar lo siguiente:

$$\lim_{k\to\infty} \mathbb{P}(X_k \in S_{opt}) = 1, \tag{4.2}$$

siendo X_k la variable estocástica correspondiente al estado k-ésimo del proceso estocástico de Markov considerado. Un proceso estocástico es sucesión de observaciones X_1, X_2,... (variables estocásticas) cuyos valores no se pueden predecir exactamente, esto es, son aleatorios, pero sin embargo, sí es posible especificar las probabilidades para los distintos posibles valores en cada instante determinado.

Para ello, se requiere definir y demostrar una serie de resultados que no se detallarán, aunque sí se describirá de manera resumida el procedimiento para probar la convergencia del algoritmo. Se basa en lo siguiente:

Definición 4.11. *Dada (S, f) una instancia de un problema de optimización combinatoria y una estructura de estados vecinos adecuada con valor de c fijo, se considera la distribución cuyas componentes vienen dadas por:*

$$q_i(c) = \frac{1}{N_0(c)} exp\left(\frac{-f(i)}{c}\right) \quad y \quad N_0(c) = \sum_{j\in S} exp\left(\frac{-f(j)}{c}\right), \tag{4.3}$$

siendo $N_0(c)$ una constante de normalización.

A partir de la definición anterior, se puede demostrar lo siguiente:

$$\lim_{c\to 0} q_i(c) = \frac{1}{|S_{opt}|}\chi_{S_{opt}}(i) \quad donde \quad \chi_{S_{opt}}(i) = \begin{cases} 1 & si\ i \in S_{opt} \\ 0 & si\ i \notin S_{opt} \end{cases}. \tag{4.4}$$

El resultado que se obtiene de la expresión (4.4) anterior tiene mucha importancia, ya que cuando el valor de c decrece a 0, la distribución $\boldsymbol{q}$ se comporta de manera adecuada. Este comportamiento se debe a que dicha distribución en el límite es uniforme sobre el conjunto de soluciones óptimas S_{opt} y, por ser un conjunto finito, se podrá obtener la expresión dada en (4.2).

En el algoritmo *Simulated Annealing* dado un estado $i \in S$, se busca otro estado $j \in S_i$, esto es, dentro del espacio de estados o soluciones próximas a la actual. Por lo tanto, se puede ver que la generación de un nuevo estado sólo depende del estado anterior. Como las cadenas de Markov sirven para modelizar las transiciones que se llevan a cabo en un determinado sistema de estados, debido a su analogía con el proceso *Simulated Annealing*, dicho algoritmo puede describirse matemáticamente usando cadenas de Markov. (Para más información sobre cadenas de Markov consultar los libros de Kai Lai Chung [19] y Isaacson, D. and R. Madsen [20]).

Finalmente, usando cadenas de Markov, bien homogéneas (aquellas en las que la probabilidad de ir del estado i al estado j en un determinado paso no depende del tiempo en

el cual se encuentre la cadena), o no homogéneas (en este caso si depende del momento de tiempo en el que estemos) se prueba que tienen como única distribución estacionaria la dada en la Definición 4.11. En consecuencia, usando la expresión (4.4) junto con otros resultados se prueba la expresión dada en (4.2), esto es, la convergencia asintótica del algoritmo hacia el conjunto de soluciones óptimas S_{opt} de una función objetivo dada f (bajo un cierto comportamiento asintótico del parámetro de control c_k).

Para más información sobre los resultados matemáticos consultar los libros de P.J.M. Van Laarhoven y Emile Aarts [3] y Emile Aarts y Jan Korst [4]. También se puede consultar el Trabajo de Fin de Grado de Matemáticas realizado que se encuentra en el Repositorio Institucional de la Universidad de Valladolid.

Como consecuencia de que converge hacia el conjunto de soluciones óptimas, resulta ser un algoritmo adecuado para resolver el objetivo perseguido. Como ya se comentó, dicho objetivo se basa en determinar aquellas trayectorias que siguen las aeronaves con una alineación temporal correcta y corregir aquellas que presentan alguna anomalía en su trayectoria. Así, mediante su uso, se podrá obtener recorridos próximos al óptimo mejorando las trayectorias de partida.

Por todo ello, y por ser un método simple y general, es muy usado debido a su gran potencial a la hora de buscar soluciones próximas a la solución óptima. Además, no consume mucha memoria, produce muy buenos resultados para problemas de tamaño grande y tiene la ventaja de que no se bloquea en óptimos locales, ya que permite considerar soluciones peores a la actual, sobretodo, en las primeras etapas del algoritmo.

En la Figura 4.16 que se muestra a continuación, se puede observar lo comentado con anterioridad, así como la interpretación del criterio de aceptación descrito en (4.1).

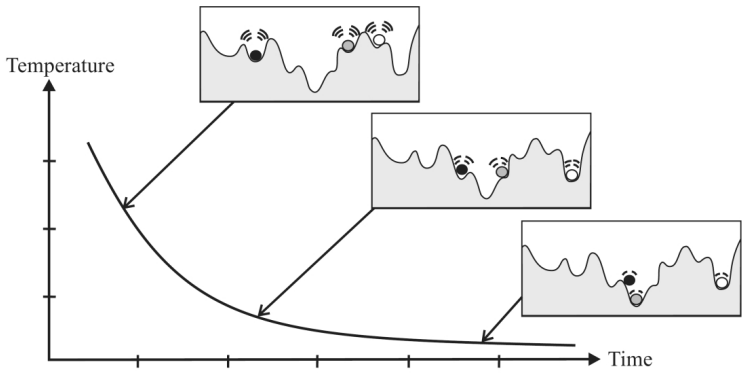


Figura 4.16: Variación de la temperatura

Durante las primeras iteraciones, la temperatura (representada por el parámetro de control c) es bastante elevada y la probabilidad de aceptar soluciones peores a las actuales es muy alta. Esta probabilidad va descendiendo a medida que decrece la temperatura, de manera que en las últimas etapas del algoritmo, la probabilidad de aceptar soluciones peores es prácticamente nula, lo cual, tiene como objetivo evitar alejarnos de la solución óptima.

4.7. Fundamento teórico de la solución

El objetivo principal del trabajo es reconstruir las trayectorias dadas con el objetivo de mejorarlas, esto es, para obtener recorridos más cercanos a los que realmente fueron volados por las aeronaves durante el vuelo, y para ello, será necesario aplicar un conjunto de algoritmos de resolución. Por lo tanto, el propósito perseguido consiste en resolver el problema de los caminos hamiltonianos de mínima distancia.

Luego, si se tiene un conjunto de n datos, la resolución de dicho problema consiste en encontrar el recorrido más corto que una el punto o estado 1 con el punto o estado n (siendo estos fijados al principio), de manera que sólo se pase por cada estado en una única ocasión. Como la formulación de este problema no coincide exactamente con la del problema del TSP descrita en este capítulo, es necesario realizar una adaptación del mismo. Para ello, se ha considerado la adición de un estado o punto auxiliar en la trayectoria que tiene distancia cero con el resto de estados considerados, y por lo tanto, permite obtener un camino cerrado.

Lo anteriormente descrito se puede visualizar en la Figura 4.17 que se muestra a continuación. Así, mediante la adición de dicha ciudad abstracta (marcada en color amarillo), se puede formar un circuito hamiltoniano de coste mínimo que es el objetivo para resolver el problema del viajante o TSP. En ella, los datos aparecen ordenados en función del tiempo de recepción de los mismos y viene indicado justo encima de cada punto.

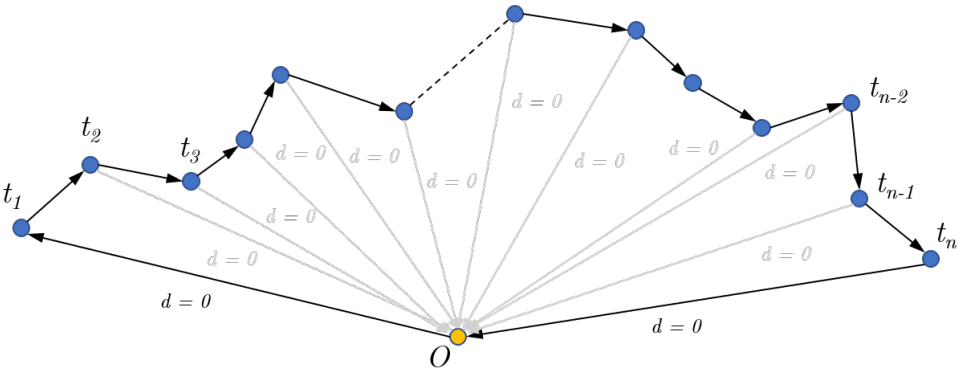


Figura 4.17: Recorrido en el que se añade una ciudad abstracta

Así, resolver el problema del camino hamiltoniano de mínima distancia entre el estado inicial 1 y el estado final n (considerando un conjunto de n datos y siendo estos estados fijados al comienzo), equivale a encontrar el circuito hamiltoniano de mínima distancia considerando la adición del estado auxiliar, siendo este el objetivo del propio problema TSP. Para que quede más claro, supongamos que se tiene el conjunto $D = \{d_i : i \in \{1, ..., n\}\}$, y que se añade el punto auxiliar denotado por O. Como la distancia entre dicho punto y el resto de puntos del conjunto D es 0, esto es, $dist(d_i, O) = 0,\ \forall\ i \in \{1, ...n\}$, se puede aplicar el problema del TSP a dicho conjunto buscando encontrar el circuito hamiltoniano de coste mínimo. Así, si se fija como punto inicial d_1 y como final d_n, se obtendrá un circuito de

mínima distancia formado por $n+1$ puntos, por ejemplo vendrá dado según la secuencia de puntos siguiente: $d_1 d_4 d_2 d_3 .... d_i O d_{i+2} ... d_{n-2} d_n$. Ahora, para encontrar el camino hamiltoniano de mínima distancia entre los puntos inicial y final fijados, basta eliminar la arista que une los puntos d_i con O y la que une los puntos O con d_{i+2}, pues al ser la distancia entre ellos 0, el coste de la solución no se verá alterado. Luego, en este caso se obtendrá el camino hamiltoniano siguiente: $d_1 d_4 d_2 d_3 .... d_i d_{i+2} ... d_{n-2} d_n$ cuya distancia será la misma que en el caso anterior. Este camino es la solución al problema del camino hamiltoniano de mínima distancia.

Para la resolución del presente problema se han considerado los siguientes algoritmos: *Inserción más cercana, más lejana, más barata y aleatoria, Vecinos más próximo y Vecinos más próximo repetitivo, 2-opt, Lin-Kernighan, Concorde* y *Simulated Annealing*.

Todos los métodos mencionados con anterioridad han sido descritos o considerados en este capítulo a excepción del método *Vecinos más próximo repetitivo*. Dicho método aplica el método vecinos más próximo partiendo de cada una de las diferentes ciudades, para así, quedarse con el recorrido que da el mejor resultado. También conviene decir, que el método *Concorde* es un algoritmo bastante avanzado y preciso para resolver el TSP y se basa en técnicas de ramificación y corte. Por ello, es de esperar obtener muy buenos resultados al ser aplicado al problema de estudio.

Además, se plantean dos posibilidades de resolución diferentes, las cuales son: aplicar los algoritmos usando ventanas de tiempo o sin usarlas.

El motivo de poder elegir la opción de ejecutar los algoritmos usando ventanas de tiempo se debe a que al reducir el número de datos a los cuales se aplican los algoritmos, se mejoran los resultados obtenidos en distancia recorrida y tiempo de ejecución. Para el caso del *Simulated Annealing* como el número de iteraciones para cada ventana es fijo, el tiempo de ejecución es algo mayor cuando se consideran ventanas que cuando no se hace. Sin embargo, este aumento de tiempo no es relevante con respecto a la mejora en distancia obtenida. Además, se considera la posibilidad de elegir un solapamiento entre las distintas ventanas para así evitar aislar los datos entre pares de ventanas consecutivas. Dicho solapamiento se realiza entre los últimos elementos de una ventana y los primeros de la siguiente. También, es posible elegir tamaños de ventana y de solapamiento diferentes en las zonas del aeropuerto y en la zona central del vuelo, lo cual, se debe a que en las zonas del aeropuerto las trayectorias no son tan rectas como en la zona central, presentando diversos quiebros y zig-zag. Por lo tanto, en ocasiones, es conveniente reducir el tamaño de las ventanas en el aeropuerto, así como el número de datos de solapamiento para obtener mejores resultados.

1ª) Sin usar ventanas de tiempo: Esta posibilidad, permite resolver el problema considerado con el algoritmo indicado, el cual, es aplicado a todo el conjunto de datos del dataset.

2ª) Usando ventanas de tiempo: Con esta posibilidad, se consideran subconjuntos del conjunto de datos original, a los cuales, se les aplica el algoritmo indicado. Para ello,

será necesario considerar ventanas de tiempo de un cierto tamaño y con un determinado solapamiento entre ellas.

Formalizándolo en términos matemáticos: Sea $D = \{d_i : i \in \mathbb{N}\}$ un conjunto tal que d_i es la información asociada al mensaje ADS-B i recibido de un vuelo, de manera que $d_1 < d_2 < ... < d_n < ...$, esto es, los mensajes están ordenados de manera creciente en función del tiempo de recepción (*timestamp*). Se entiende por ventana de tiempo de tamaño t a un subconjunto de datos $D^{'} \subset D$ tal que $|D^{'}| = t < |D|$ (considerando que hay más de una ventana de tiempo pues sino $|D^{'}| = t = |D|$).
Si se tienen n ventanas de tiempo consecutivas de tamaño t con solapamiento s entre ellas siendo $s \leq t$, entonces se tiene que: $D_1, ..., D_n \subset D$ de manera que $|D_1| = ... = |D_n| = t$ y $|D_i \cap D_{i+1}| = s$, $\forall\, i = 1, ..., n$, esto es, coinciden los s últimos elementos de la ventana D_i con los s primeros de D_{i+1}. Luego, si $D_i = \{d_{(i-1)t}, ..., d_{it-1}\}$ y $D_{i+1} = \{d_{it}, ..., d_{(i+1)t-1}\}$, entonces $d_{(it-1)-s} = d_{it}, ..., d_{it-1} = d_{it+s}$.

En la Figura 4.18 se detalla de manera gráfica lo explicado con anterioridad.

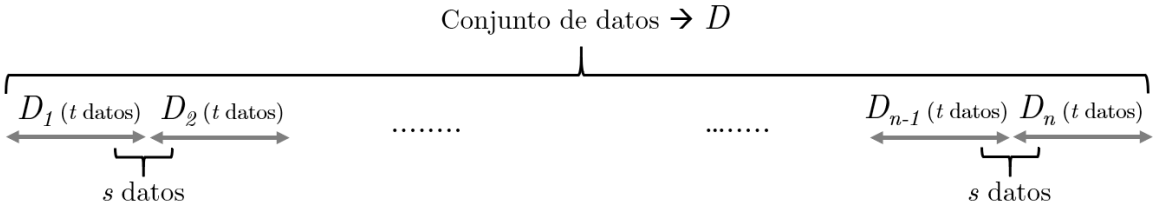


Figura 4.18: Uso de ventanas de tiempo

Cuando se usen ventanas de tiempo se van a considerar dos posibles casos:

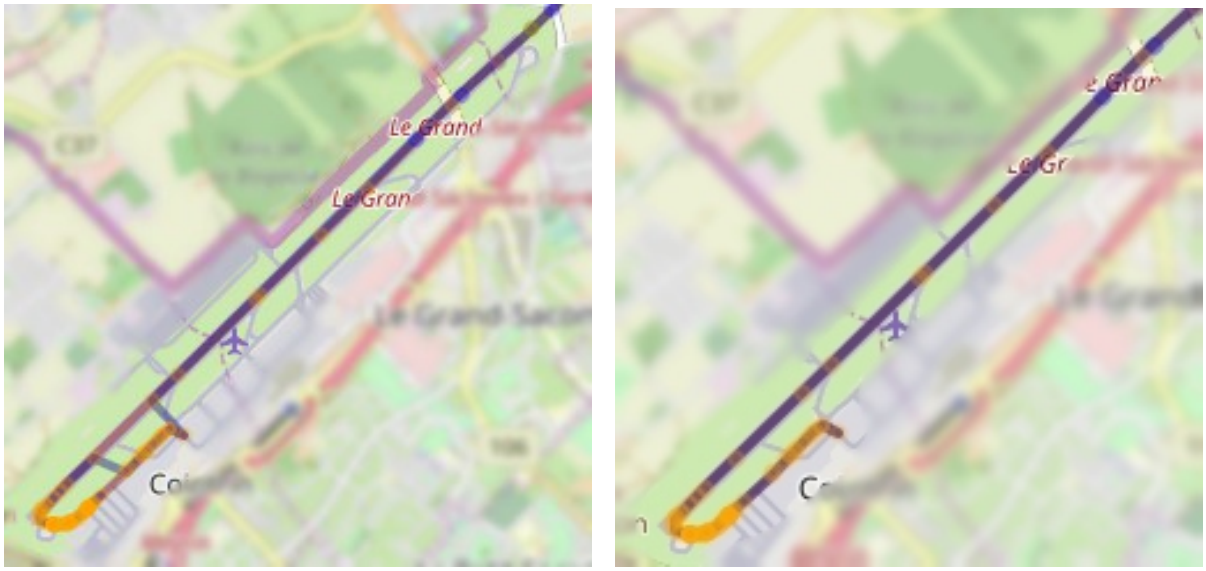
· Todas las ventanas son del mismo tamaño t con solapamiento s → En dicho caso, la aplicación del algoritmo debe realizarse sobre cada una de las ventanas consideradas, donde el número de ejecuciones vendrá dado por:

$$f(t, s) = 1 + \left\lfloor \frac{|D| - s - 1}{t - s} \right\rfloor$$

· Se consideran diferentes tamaños de ventana → En este caso sólo se van a tener en cuenta dos tamaños diferentes, donde uno se corresponderá con las zonas de los aeropuertos, y el otro con la zona central del vuelo, siendo el motivo de esta diferenciación el comentado anteriormente. Se denota por t_v el tamaño de ventana en vuelo, t_a el tamaño de ventana en el aeropuerto, s_v el solapamiento en vuelo y s_a el solapamiento en el aeropuerto. Así, el algoritmo se aplica a cada uno de los diferentes trozos: en los dos trozos del aeropuerto, se aplica $f(t_a, s_a)$ veces y, en el trozo del vuelo se aplica $f(t_v, s_v)$ veces. Por lo tanto, se obtiene que el número de veces que se aplica el algoritmo es $2f(t_a, s_a) + f(t_v, s_v)$.

Dependiendo de si un vuelo presenta más problemas debido al fallo en el alineamiento temporal o menos, será recomendable aplicar una alternativa u otra. A continuación se muestran los resultados obtenidos de una serie de ejemplos realizados con diferentes vuelos y consideraciones.

Las dos imágenes siguientes 4.19a y 4.19b, sirven para ilustrar como en determinados vuelos cuyas trayectorias cerca de los aeropuertos presentan ciertos quiebros y zig-zag es útil usar ventanas de tiempo. La presencia de dichas irregularidades se debe a que en las fases del aterrizaje y el despegue las aeronaves suelen realizar más maniobras que en la zona del vuelo, siendo en el vuelo las trayectorias más rectas. Se observa como en la imagen 4.19b se corrige la desviación presente en la trayectoria de la imagen 4.19a como consecuencia de usar ventanas de tiempo en las zonas del aeropuerto.



(a) Sin usar ventanas (b) Usando ventanas con $t_a = 10$ y $s_a = 2$

Figura 4.19: Trayectoria ordenada sin usar ventanas de tiempo o usándolas

Así, mediante muchas de estas pequeñas correcciones, se logra reducir la distancia en km obtenida en total, y por lo tanto, estar más próximos de la solución óptima que es el objetivo perseguido.

En las Figuras 4.20 y 4.21 que se muestran a continuación, se observa un ejemplo de desajuste temporal en la recepción de ciertos mensajes ADS-B. En la Figura 4.20 se muestra en color rojo la trayectoria reconstruida a partir de los mensajes obtenidos, y en la Figura 4.21, se ve en color azul cual es la solución dada por el algoritmo de resolución *Inserción más cercana* del problema del TSP adaptado a nuestro problema particular.
Se puede ver como la distancia en km de la trayectoria se reduce de manera considerable al aplicar dicho algoritmo de resolución.

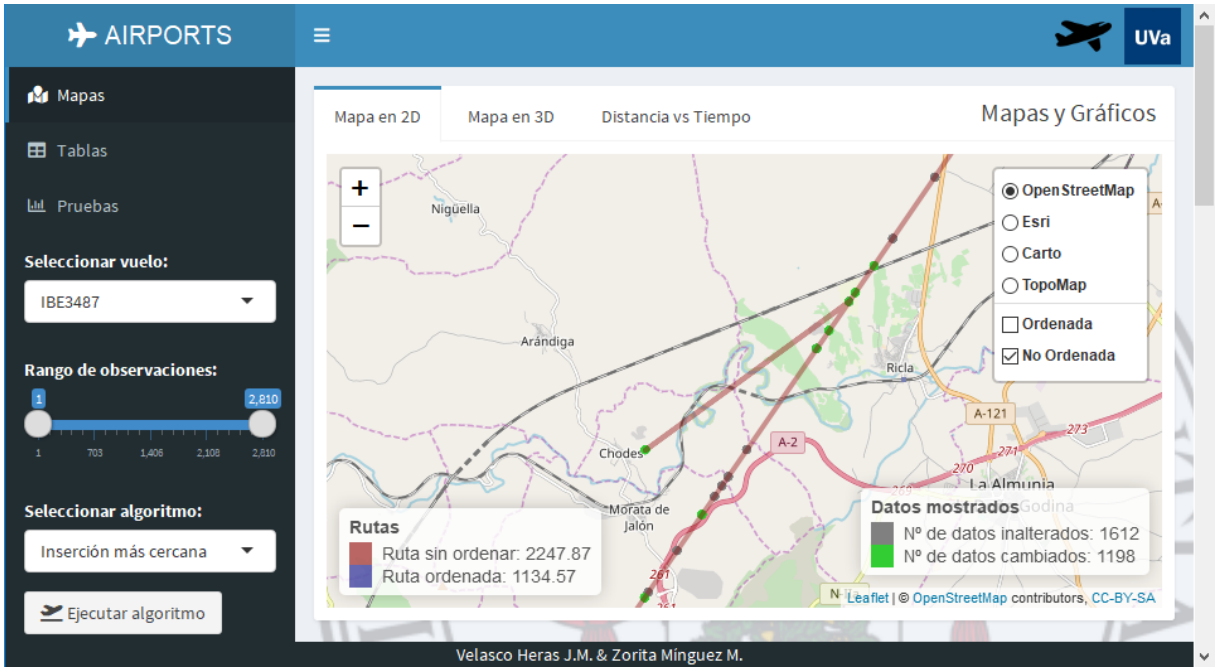


Figura 4.20: Ruta del vuelo *IBE3487* sin ordenar

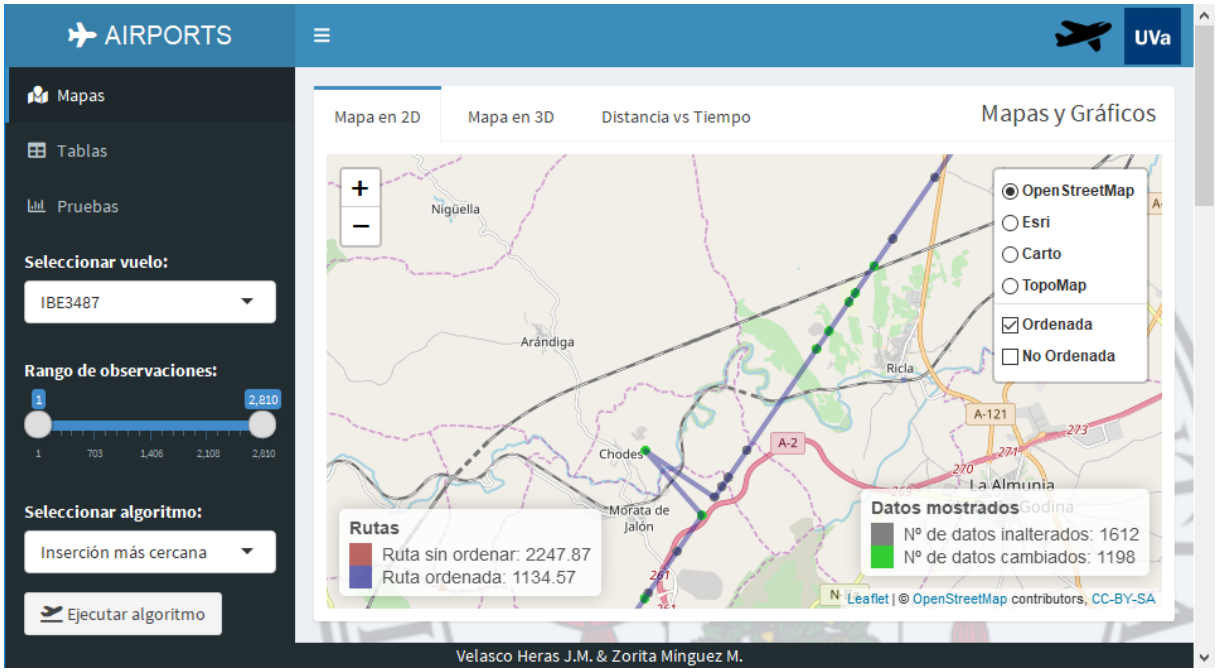


Figura 4.21: Ruta del vuelo *IBE3487* ordenada

Una vez explicado el problema del viajante en detalle, los distintos algoritmos de resolución y la relación entre el famoso problema del viajante y el problema de estudio, en las siguientes secciones se realiza un estudio sobre el comportamiento de los algoritmos cuando son aplicados para resolver el problema planteado, así como la descripción del entorno y las herramientas necesarias para su realización.

Capítulo 5

Estudio preliminar de los algoritmos en *R-Studio*

Durante este capítulo se describe cómo se ha realizado el dashboard en el entorno de desarrollo *R-Studio*. Los objetivos de su realización son los siguientes: por un lado, obtener las trayectorias ordenadas tras la aplicación de los distintos algoritmos a los vuelos proporcionados, y por el otro, realizar la reasignación de tiempos a los datos que han sido alterados en la reordenación usando para ello un modelo de interpolación lineal. Con ello, se podrá llevar a cabo un estudio sobre el comportamiento de los diferentes algoritmos, para así realizar una comparativa de los mismos en términos de distancia recorrida y tiempo de ejecución. Dicho estudio se realiza en el Capítulo 6.

Por lo tanto, antes de comenzar a describir las herramientas, entorno, etc. usado y la implementación realizada vamos a describir las funcionalidades que perseguimos obtener con su realización. Esto es, describiremos los requisitos funcionales del dashboard a desarrollar en *R-Studio*. Dichos requisitos serán los siguientes:

RF-01: El dashboard permitirá al usuario elegir entre visualizar mapas, visualizar la tabla de datos de un vuelo y realizar una batería de pruebas o experimentos.

RF-02: El dashboard permitirá al usuario elegir entre un conjunto de vuelos.

RF-03: El dashboard permitirá al usuario elegir el rango de datos a considerar para el vuelo elegido.

RF-04: El dashboard mostrará un mapa en 2 dimensiones (2D) teniendo en cuenta los datos considerados sobre el vuelo elegido para describir la trayectoria de vuelo.

RF-05: El dashboard mostrará un mapa en 3 dimensiones (3D) teniendo en cuenta los datos considerados sobre el vuelo elegido para describir la trayectoria de vuelo.

RF-06: El dashboard permitirá al usuario cambiar la *capa o layer de visualización* del mapa en 2D.

RF-07: El dashboard mostrará un minimapa en el mapa en 2D, para que en caso de hacer zoom, se pueda conocer la zona donde se encuentra.

RF-08: El dashboard mostrará unas gráficas de velocidad frente a tiempo y de altura frente a tiempo del vuelo elegido y el rango de datos indicado.

RF-09: El dashboard permitirá al usuario visualizar en el mapa en 2D la información asociada a un determinado mensaje.

RF-10: El dashboard permitirá al usuario seleccionar los datos a visualizar en los mapas y gráficos según el proveedor indicado.

RF-11: El dashboard permitirá al usuario elegir entre la opción de visualizar en los mapas los datos por colores en función del tipo de proveedor o sin colores.

RF-12: El dashboard mostrará al usuario una tabla con los datos del vuelo indicado teniendo en cuenta el rango de datos considerado.

RF-13: El dashboard permitirá al usuario filtrar los datos que desea buscar en la tabla.

RF-14: El dashboard permitirá al usuario elegir un algoritmo de resolución entre un conjunto de ellos.

RF-15: El dashboard permitirá al usuario elegir para el caso del algoritmo *Simulated Annealing*, un algoritmo previo de resolución, el número de iteraciones a realizar, así como la temperatura considerada.

RF-16: El dashboard permitirá al usuario ejecutar el algoritmo indicado mediante tres posibilidades diferentes: sin usar ventanas de tiempo, usando ventanas de tiempo del mismo tamaño y usando ventanas de tiempo de distinto tamaño en las zonas del aeropuerto y la zona central del vuelo.

RF-17: El dashboard permitirá en el caso de usar ventanas de tiempo, indicar al usuario los datos asociados a cada una de las zonas consideradas, así como el tamaño de la ventana en el aeropuerto y en el vuelo, y el solapamiento de ventanas en cada caso.

RF-18: El dashboard mostrará al usuario en los mapas la información necesaria para poder distinguir entre los datos que han sido alterados y los que no tras la reordenación.

RF-19: El dashboard permitirá al usuario elegir visualizar la trayectoria ordenada y/o no ordenada en el mapa en 2D y 3D.

RF-20: El dashboard mostrará al usuario la información asociada a la distancia total en km de la trayectoria original y la ordenada, así como el número de puntos que han sido alterados o no tras la reordenación.

RF-21: El dashboard mostrará al usuario tras ejecutar un algoritmo cuatro columnas nuevas en la tabla (*distancia*, *time_nuevo*, *tag_orig* y *tag_ord*).

RF-22: El dashboard mostrará al usuario, tras ejecutar un determinado algoritmo, una gráfica donde se representa el tiempo de cada punto tras realizar la interpolación de los erróneos frente a la distancia de cada uno de ellos con respecto al primero.

RF-23: El dashboard permitirá visualizar en la gráfica anterior un gradiente en función de la altura de cada uno de los puntos considerados.

RF-24: El dashboard permitirá al usuario realizar una batería de pruebas o experimentos de manera dinámica.

RF-25: El dashboard permitirá al usuario elegir un conjunto de vuelos, de algoritmos y las opciones de ventanas deseadas y realizar su ejecución.

RF-26: El dashboard mostrará al usuario tres tablas sobre los resultados obtenidos en términos de distancia, eficacia y tiempo de ejecución, así como unas métricas asociadas (media aritmética, median absolute desviation y desviación estándar muestral).

RF-27: El dashboard permitirá al usuario descargar los datos mostrados en las tablas en formato *csv* para poder guardar dicha información.

5.1. Descripción del entorno y herramientas usadas

El procedimiento seguido para la realización del dashboard ha sido el siguiente:

1. Instalación de *R-Studio*: Para ello se ha descargado la versión gratuita de la página web https://www.rstudio.com/products/rstudio/download/.

Se ha usado dicho entorno de desarrollo o IDE dado que, principalmente está dedicado a la computación estadística, el diseño de gráficos y por ser el más usado para crear aplicaciones en *R*. La apariencia de dicho entorno se muestra en la Figura 5.1. En ella, se puede apreciar que consta de 4 pantallas (el editor de texto, la consola, una zona de visualización de gráficas, ayuda, paquetes, etc. y por último el entorno de trabajo junto con el historial).

El lenguaje de programación usado en este entorno es *R*, el cual, es utilizado principalmente para el análisis estadístico, aunque también es muy usado en investigación científica, minería de datos, investigación biomédica, bioinformática y matemáticas financieras. Entre sus características y ventajas podemos destacar las siguientes:

- Es multiplataforma.
- Es un proyecto colaborativo y abierto, lo que permite el desarrollo de nuevas librerías y la realización de mejoras.
- Es un lenguaje interpretado, esto es, funciona mediante comandos.
- Proporciona una gran variedad de herramientas estadísticas para el análisis de datos y la generación de gráficos de alta calidad (de ahí su gran uso en el trabajo).
- Permite manejar una gran cantidad de datos (usado en Data Science).
- Funciona con diferentes tipos de hardware y software (Linux, Windows, etc.)
- Con el uso de las librerías y paquetes se puede ampliar su configuración básica. Estas serán detalladas a continuación, pero cabe destacar que han sido de gran utilidad para la realización de la interfaz del dashboard aquellas que se basan en la visualización de

gráficos. También han sido muy útiles aquellos paquetes que contienen los algoritmos o métodos que han sido aplicados a la ordenación de las trayectorias obtenidas.

Por lo tanto, se trata de un lenguaje de programación muy usado, lo cual hace que su mejora sea continua, estando en la actualidad en constante avance y crecimiento.

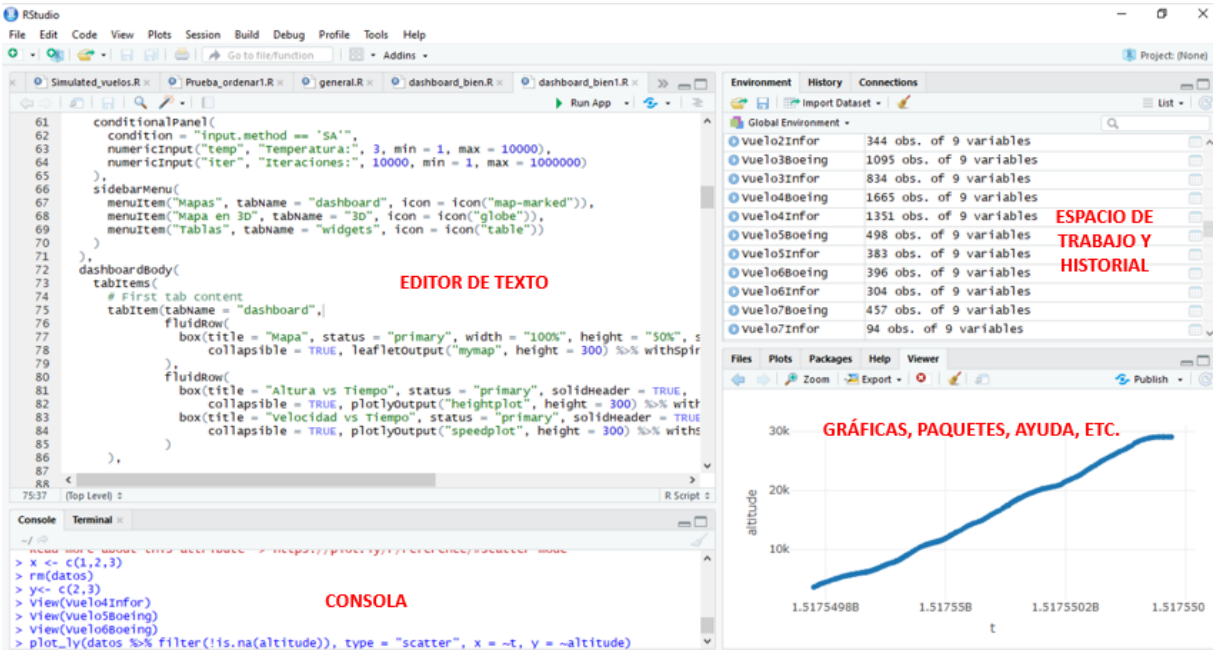


Figura 5.1: Entorno de *R-Studio*

2. Instalación de librerías de *R*: Una vez instalado el entorno de *R-Studio*, es necesario instalar una serie de librerías de *R* para poder realizar el dashboard. Como *R* posee muchas librerías, sólo detallaré aquellas que han sido usadas en la creación del dashboard.

Para instalar nuevos paquetes existen básicamente dos formas diferentes:

1) Usar la consola de comandos poniendo directamente *install.packages("nombre_paquete").*
2) Usar la zona denominada gráficas, paquetes, ayuda, etc. de la Figura 5.1.
   Para ello, tras seleccionar la pestaña paquetes se puede instalar un nuevo paquete o actualizar alguno de los existentes.

Una vez que se han instalado los paquetes, para hacer uso de ellos es necesario poner la sentencia *library(paquete)* para cada uno de los paquetes que se necesiten. Todas estas sentencias se ponen al comienzo del script usado para implementar el dashboard y sirven para cargar los paquetes requeridos para su ejecución.

Debido a las características que presenta el tipo de proyecto a realizar, ha sido necesario usar la librería *Shiny*. Se trata de una librería de *R* muy útil, ya que facilita la creación de aplicaciones web interactivas directamente desde *R*, pudiendo además incluirse temas

css adicionales, acciones de JavaScript, etc.

Además, ha sido necesario instalar otras librerías para poder desarrollar el dashboard como son las siguientes:

- **png**: Dicho paquete debe instalarse para poder instalar el paquete Leaflet.
- **leaflet**: Dicho paquete permite manejar mapas, por lo que su uso es imprescindible para visualizar las trayectorias voladas por las aeronaves.
- **shinydashboard**: Dicho paquete facilita el uso del dashboard creado usando *Shiny* mediante el uso de diversos paneles de control.
- **shinycssloaders**: Dicho paquete nos ha permitido instalar una serie de spinners que se visualizan mientras se cargan los datos necesarios para dibujar los mapas, gráficas, visualizar las tablas, etc.
- **shinyalert**: Dicho paquete sirve para mostrar al usuario diversos mensajes de alerta en el caso de que haya introducido algún dato erróneo en los campos.
- **ggplot2**: Dicho paquete permite realizar gráficos.
- **plotly**: Dicho paquete lo que hace es mejorar el aspecto y la apariencia de las gráficas obtenidas con ggplot2. Se ha usado para la realización de todos los gráficos y mapas del dashboard.
- **geosphere**: Dicho paquete se ha usado para calcular la distancia del Haversiano en la esfera.
- **TSP**: Dicho paquete permite resolver el problema del viajante o TSP mediante el uso de diversos métodos.
- **dplyr**: Dicho paquete ha sido usado para realizar operaciones de manera sencilla y rápida sobre los datasets.
- **stats**: Dicho paquete se ha usado para realizar cálculos estadísticos y generar números aleatorios. Además, contiene la función *optim()* que se ha usado para resolver el problema del viajante o TSP mediante el método *Simulated Annealing*.
- **DT**: Dicho paquete proporciona una interfaz usable y manejable además de bonita para la visualización de las tablas de datos en *R*. Permite mostrar los datos como tablas en páginas HTML y DataTables, proporciona filtrado, paginación, clasificación y muchas otras características.
- **readr**: Dicho paquete permite leer datos de diferentes formatos de archivos como *csv*, *tsv* y *fwf*.
- **shinyBS**: Dicho paquete se ha usado para mostrar ventanas emergentes informativas al usuario.
- **yasp**: Dicho paquete permite aplicar ciertas funciones para el manejo de Strings.
- **tibble**: Dicho paquete proporciona un manejo sencillo de dataframes (conjuntos de datos).
- **stringr**: Dicho paquete permite realizar diversas manipulaciones sobre caracteres de datos (Strings).

5.2. Implementación del dashboard

Para desarrollar el dashboard se ha usado el lenguaje de programación *R* pues como ya se comentó es muy usado en la actualidad. Para crear el dashboard se ha hecho uso de la librería conocida como *Shiny*. Dicha librería ha sido de gran utilidad para el desarrollo del proyecto ya que permite crear con cierta facilidad aplicaciones web visuales e interactivas para el usuario. Esto se logra gracias a que en su código fuente se integra código de otros lenguajes como HTML, CSS o JavaScript sin necesidad de hacer uso directo de ellos.

Shiny implementa la programación reactiva que consiste en vincular los valores de entrada con los de salida, esto es, cuando una entrada (*input*) cambia, el servidor reconstruye cada salida (*output*) que depende de ella (también si la dependencia es indirecta) de manera automática. Además, dispone de widgets y herramientas que permiten hacer aplicaciones bonitas y visuales además de interactivas. Si se quiere obtener más información se puede consultar la página web: http://shiny.rstudio.com/

Antes de comenzar a desarrollar el dashboard es necesario instalar el paquete de *Shiny* mediante el comando siguiente: *install.packages("shiny")*.
Los dashboards desarrollados haciendo uso de *Shiny* constan básicamente de dos partes, que son las siguientes:

1. **Interfaz de usuario (ui.R)** → En esta parte se encuentra todo el código asociado a la interfaz gráfica del dashboard creado, esto es, asociado a los componentes que se quieren mostrar.

   Está distribuido en tres partes, las cuales, conforman el cuadro de mando: el encabezado, el menú lateral y el cuerpo o Body. Viene definido de la forma siguiente:

   ```
   #Librerías básicas para la interfaz de usuario.
   library(shiny)
   library(shinydashboard)

   ui = dashboardPage(
       #Encabezado, menú lateral y cuerpo (body).
       dashboardHeader(),
       dashboardSidebar(),
       dashboardBody()
   )
   ```

2. **Servidor (server.R)** → En esta parte se encuentran todas las funciones necesarias para realizar los análisis, operaciones y gráficos que se muestran en la interfaz de usuario del dashboard, esto es, contiene la lógica de la programación.

Por lo tanto, la estructura básica de un dashboard en *Shiny* viene dada como:

```
library(shiny)
library(shinydashboard)
```

```
# Interfaz de usuario
ui <- fluidPage()

# Parte del servidor
server <- function(input, output) {}

# Ejecuta el dashboard
shinyApp(ui = ui, server = server)
```

Finalmente, en la Figura 5.2 se muestra la arquitectura básica del dashboard desarrollado en *Shiny*. Como vemos consta de una parte cliente y otra parte servidor que interactúan entre sí.

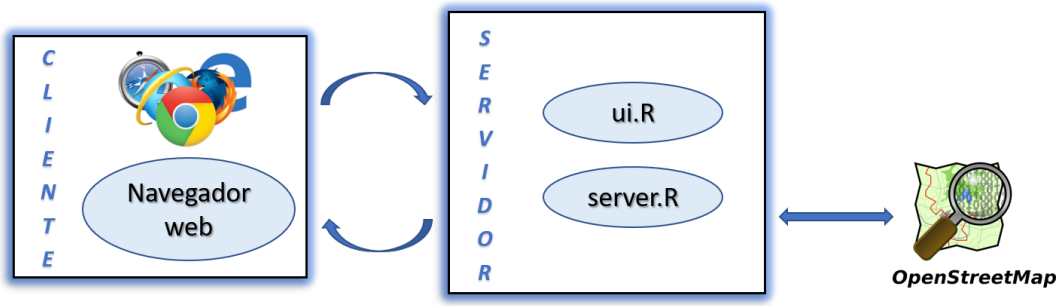


Figura 5.2: Arquitectura de *Shiny* [16]

5.3. Manual de usuario del dashboard creado

Como ya se ha comentado, el dashboard desarrollado se basa principalmente en la visualización de trayectorias, recorridos (tanto ordenados como no ordenados), datos de vuelos, resultados obtenidos de la ejecución de los distintos métodos, etc. para así poder realizar una comparativa de los mismos en términos de distancia recorrida y tiempo de ejecución. Además, una vez ordenadas las trayectorias, se realiza la reasignación de *timestamps* para los datos que han sufrido cambios, así como la representación gráfica de todos ellos.

Ahora, se explica el funcionamiento del dashboard y para que sea más visual y se entienda mejor, se muestran una serie de imágenes de la interfaz asociada a cada una de las explicaciones dadas. Existen diversas funcionalidades en el dashboard, las cuales son:

Dashboard Mapa 2D: Es la pestaña que se visualiza al iniciar el dashboard. En ella se puede visualizar un mapa con la trayectoria descrita por el vuelo que aparece indicado (por defecto el primero del selector *"Seleccionar vuelo"*), siendo estos leídos de una carpeta denominada Vuelos. Dicha trayectoria se construye con los mensajes ADS-B obtenidos de cada proveedor, cada uno de los cuales se indica con un determinado color (existe la posibilidad de que aparezcan todos con el mismo color pulsando en la opción *"Con colores"* que se visualiza en la parte derecha del mapa y de eliminar la visualización de los datos de un determinado proveedor pulsando el checkbox correspondiente) y vienen

unidos mediante una línea roja. En el caso en que se eliminen los datos de todos los proveedores de un vuelo, se visualizará la trayectoria de la ruta de partida. En la imagen de la Figura 5.3 se muestra el caso en que sólo se consideran los datos asociados al proveedor OpenSky. Además, existe la posibilidad de visualizar la información de un determinado mensaje ADS-B pulsando sobre el mismo. También, se puede cambiar la *"capa o layer de visualización"* del mapa (hay varias posibilidades) y ver un minimapa para saber en caso de aplicar zoom al mapa, la zona donde nos encontramos.

Justo debajo de dicho mapa se muestran dos gráficas que representan el tiempo frente a la altura y la velocidad de los datos del correspondiente vuelo. Es posible cambiar desde el menú lateral izquierdo el vuelo considerado, así como restringir los datos con el slider *"Rango de observaciones"*. Todos estos cambios se realizan sin tener que recargar la página, y los mapas y gráficos se muestran de manera inmediata.

Además, como se ve en la Figura 5.3 el filtrado por proveedores también se visualiza en las gráficas de velocidad y altura.

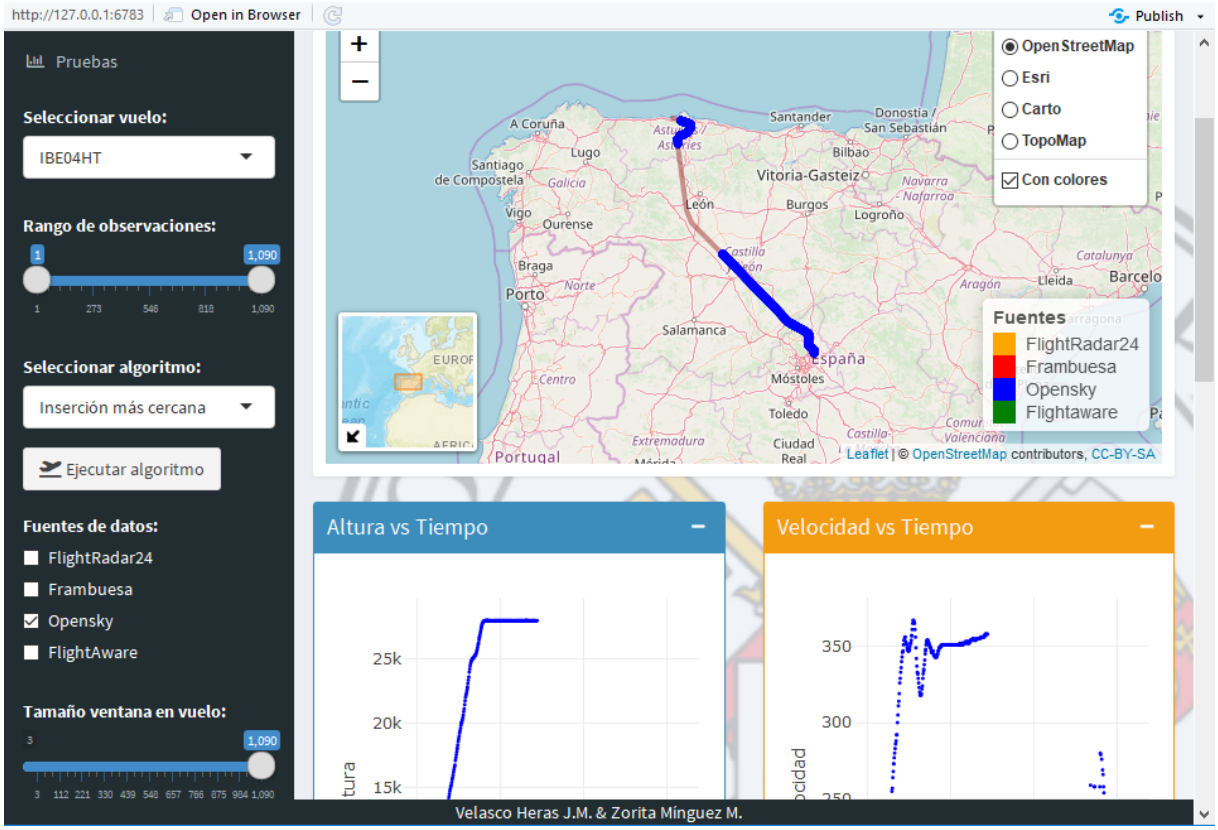


Figura 5.3: Parte superior pestaña mapa en 2D con filtrado de proveedores

En las imágenes de las interfaces asociadas a las Figuras 5.4 y 5.5 se visualizan las explicaciones dadas con anterioridad. La interfaz de la Figura 5.5 es común para la visualización del Mapa en 2D, 3D y la gráfica Distancia vs Tiempo cambiando sólo la parte superior ya que el menú lateral izquierdo también se mantiene.

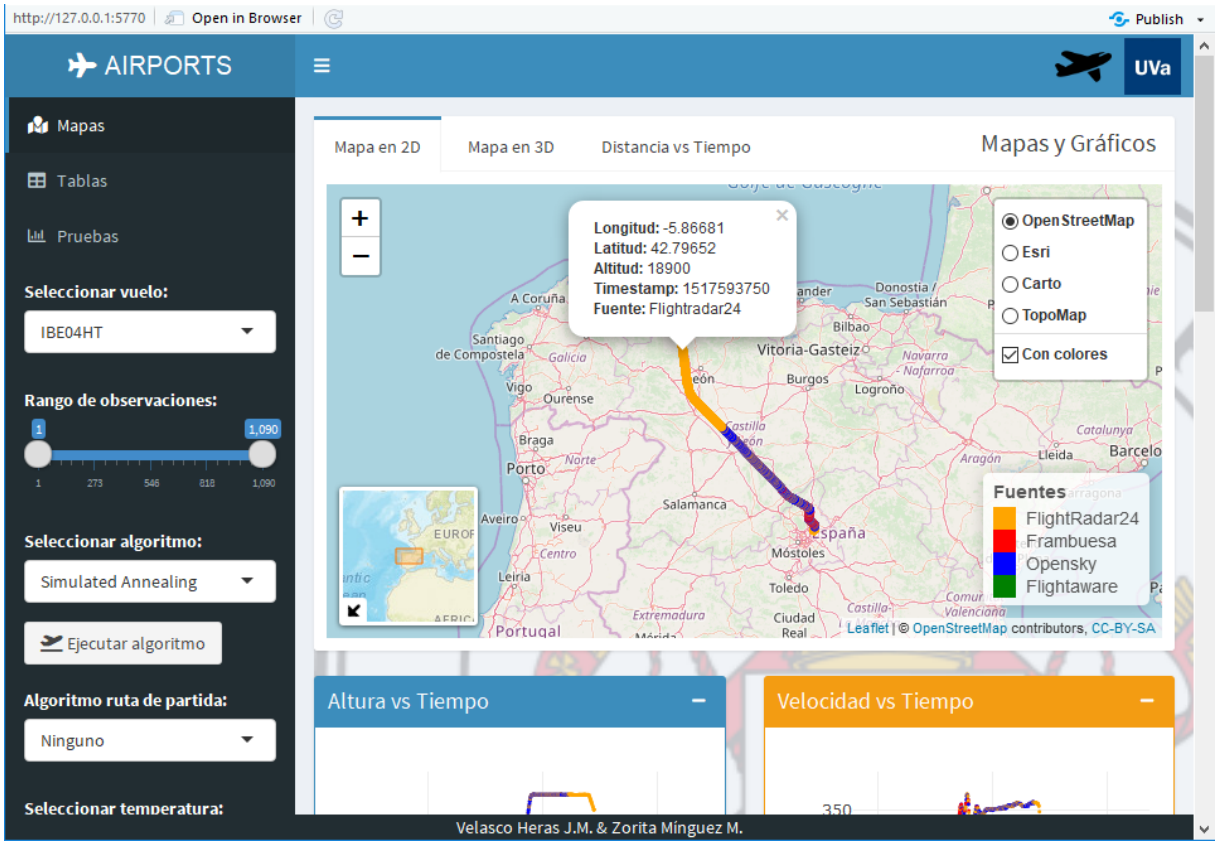


Figura 5.4: Parte superior pestaña mapa en 2D

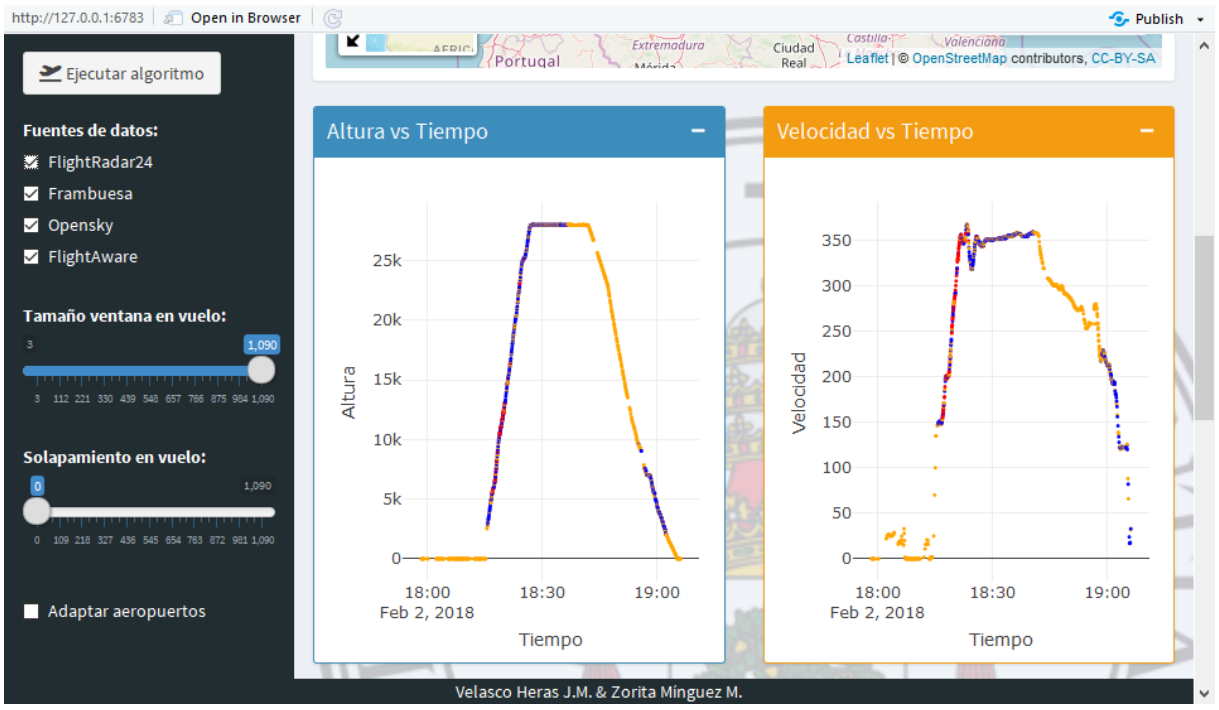


Figura 5.5: Parte inferior pestaña mapa en 2D, en 3D y Distancia vs Tiempo

Seleccionar método: Además, existe en el menú lateral la opción de elegir el método de resolución que se desee mediante el selector *"Seleccionar algoritmo"*. Existen unos cuantos métodos, los cuales son: *Inserción más cercana, más lejana, más barata y aleatoria, Vecinos más próximo y Vecinos más próximo repetitivo, 2-opt, Lin-Kernighan, Concorde* y *Simulated Annealing*. Todos ellos se encuentran en el paquete *TSP*, a excepción del *Simulated Annealing*, para el que se ha usado el método *optim()* del paquete *stats*.

Hay que tener en cuenta que si se elige el método *Simulated Annealing* se visualizarán a mayores en el menú lateral tres nuevos selectores que permiten elegir un algoritmo de resolución previo (selector *"Algoritmo ruta de partida"*), la temperatura (selector *"Seleccionar temperatura"*) y el número de iteraciones a realizar (selector *"Número de iteraciones"*). Esto se puede visualizar en la Figura 5.6. Además, en el caso de que los valores elegidos para el selector de temperatura y número de iteraciones no sean enteros se mostrará un mensaje de alerta al usuario informándole del error.

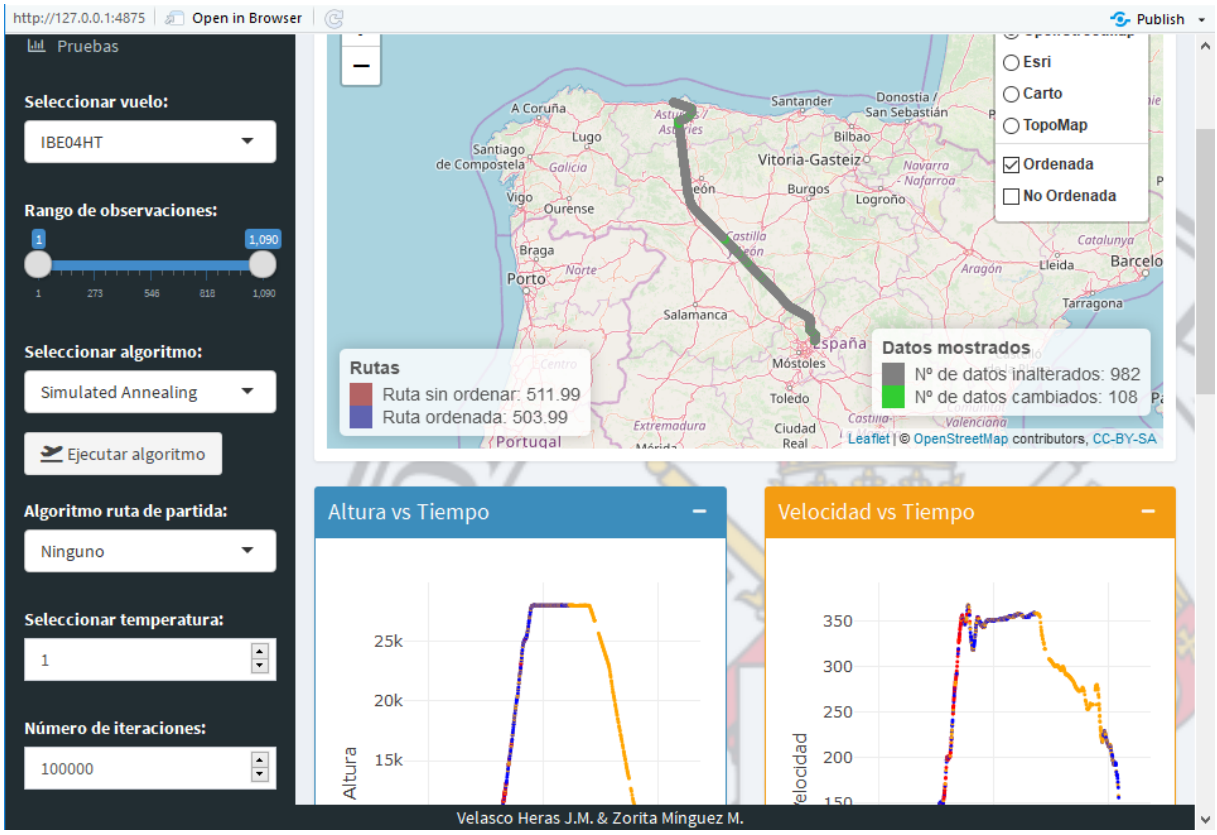


Figura 5.6: Opción Seleccionar método *Simulated Annealing*

Una vez elegido el método e indicado los parámetros (en caso de que sea necesario), pulsando el botón *"Ejecutar algoritmo"* se procederá a su resolución. Además, como se observa en la Figura 5.5 existe la posibilidad de elegir para ejecutar un algoritmo, un determinado tamaño de ventana en vuelo así como el solapamiento entre ellas. También, seleccionando el checkbox *"Adaptar aeropuertos"* se puede indicar un determinado tamaño de ventana en las zonas del aeropuerto así como un solapamiento entre ellas.

Para delimitar el número de datos que forman parte de la zona central del vuelo o de las zonas del aeropuerto existen dos formas: la primera es hacerlo de manera manual mediante el slider denominado *"Segmentos ventana"* y la otra opción es introducir en el input denominado *"Seleccionar altura"* un valor, de manera que el slider se ajusta automáticamente. Esto se muestra a continuación en la interfaz de la Figura 5.7.

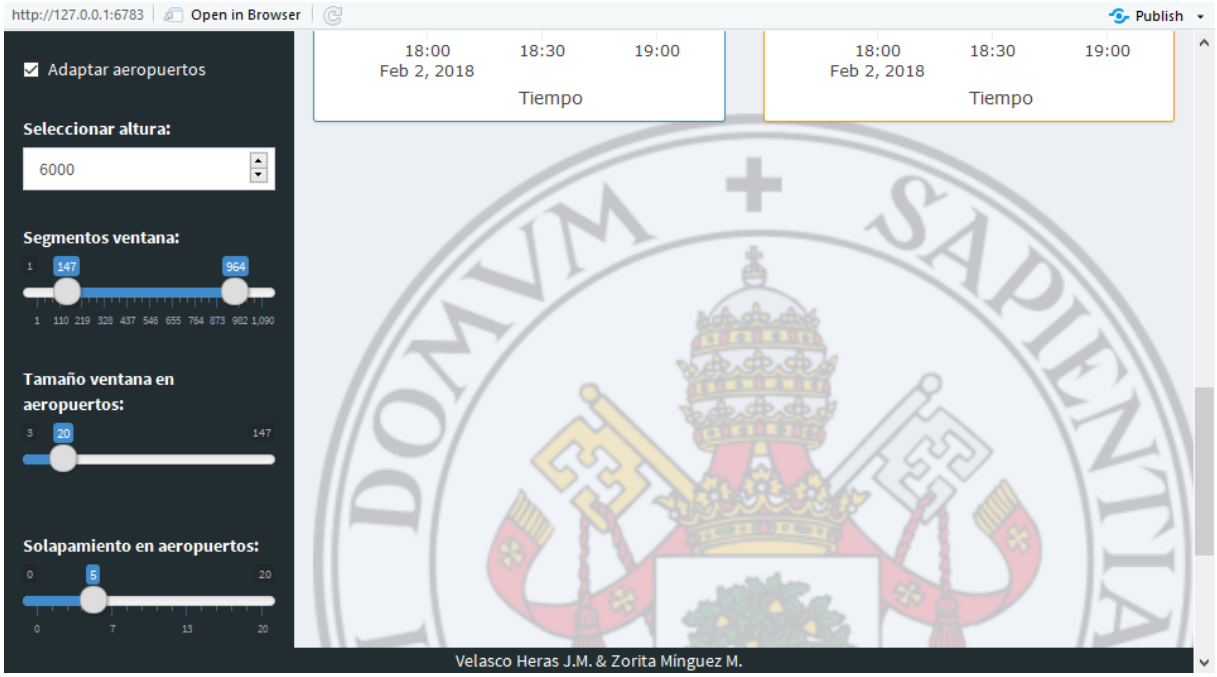


Figura 5.7: Adaptación en las zonas del aeropuerto

Como se puede observar, en todas las interfaces mostradas aparece el mismo estilo, de manera que se ha considerado como fondo la imagen del escudo de la Universidad de Valladolid, junto con una serie de iconos y símbolos en las cabeceras y botones.

Una vez ejecutado el algoritmo elegido, cambiará la apariencia del mapa, mostrando en este caso los mensajes ADS-B de dos colores diferentes (gris para los puntos que no han sido alterados y verdes para los que si han sufrido un cambio de orden). Además, se puede visualizar tanto la ruta no ordenada (la de partida que vendrá marcada en color rojo) como la ordenada (obtenida al aplicar el algoritmo y vendrá marcada en color azul). Esta información junto con la distancia en km de cada ruta aparecerá visible en la leyenda situada en la parte inferior izquierda. En el caso en que se eliminen los datos de todos los proveedores, se mostrará la ruta ordenada. En la Figura 5.8 puede visualizarse el caso en que sólo aparecen los datos asociados al proveedor OpenSky.

Para visualizar o dejar de visualizar dichas rutas basta clicar en los checkbox que aparecen en el mapa con el nombre *"Ordenada"* y *"No Ordenada"*. Además, aparece otra leyenda en la parte inferior derecha que indica el número de puntos alterados al ejecutar el algoritmo, así como aquellos que no han sufrido cambios. Esto se puede ver en la Figura 5.6 mostrada con anterioridad y la Figura 5.8 que se muestra a continuación.

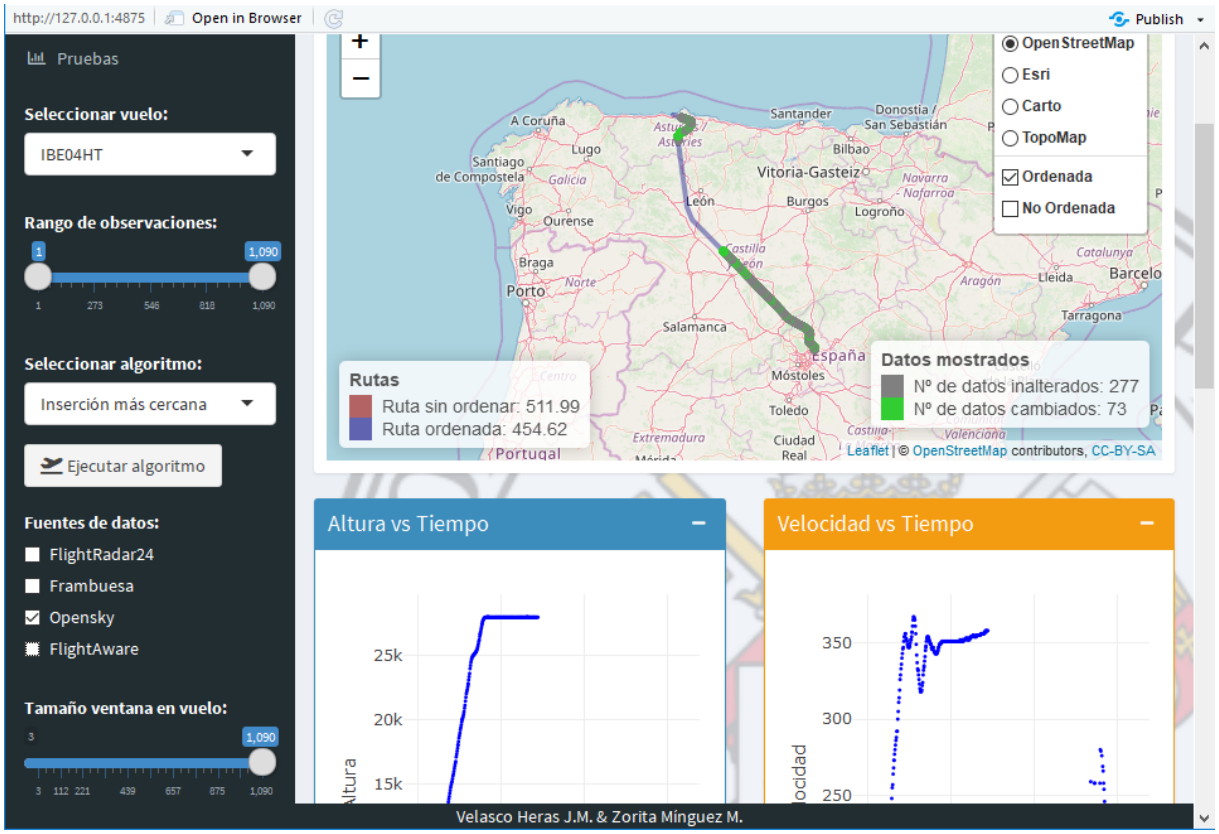


Figura 5.8: Ejecución algoritmo con filtrado por proveedor

Dashboard Mapa 3D: Desde la página principal si se quiere visualizar el mapa en 3 dimensiones (longitud, latitud y altitud) se debe seleccionar la pestaña *"Mapa en 3D"*. Tanto las gráficas inferiores como el menú lateral no varían con respecto a lo anterior.

En dicha gráfica también podemos visualizar los datos de los distintos proveedores y realizar un filtrado según si se quieren visualizar los de todos, o sólo los de ciertos proveedores. Por defecto la ruta no se visualiza para evitar confusiones con tantos colores, pero se puede ver pulsado en la parte derecha sobre el texto *"Ruta"*. Igualmente, pulsando de nuevo sobre dicho texto, podemos dejar de visualizarla. Además, si se pulsa sobre el texto *"Mensajes ADS-B"*, se deja de visualizar tanto la ruta como los datos asociados a dichos mensajes. También si nos situamos sobre un determinado mensaje podemos ver cierta información asociada al mismo (Ver la Figura 5.9). Si se eliminan los datos de todos los proveedores se visualiza la trayectoria de la ruta de partida como se ve en la Figura 5.10.

Al igual que en el mapa en 2D si se ejecuta un método, cambia la gráfica pasándose a visualizar como en la Figura 5.11. En ella, a mayores se puede visualizar la ruta ordenada obtenida tras aplicar el método seleccionado, así como los puntos que han sido alterados (color verde) y los que no han variado (color gris). En el caso en que se eliminen los datos de todos los proveedores se muestra la trayectoria de la ruta ordenada. Esto se puede ver en la interfaz de la Figura 5.12.

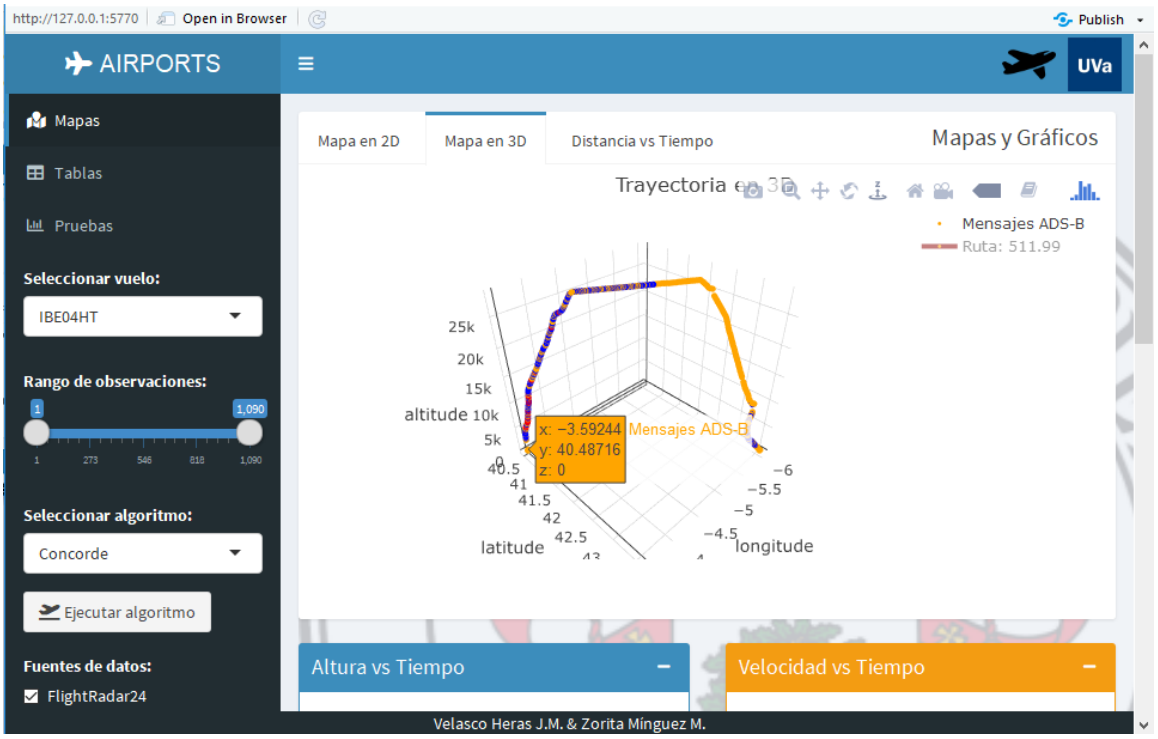


Figura 5.9: Parte superior pestaña mapa en 3D sin ejecutar algoritmo

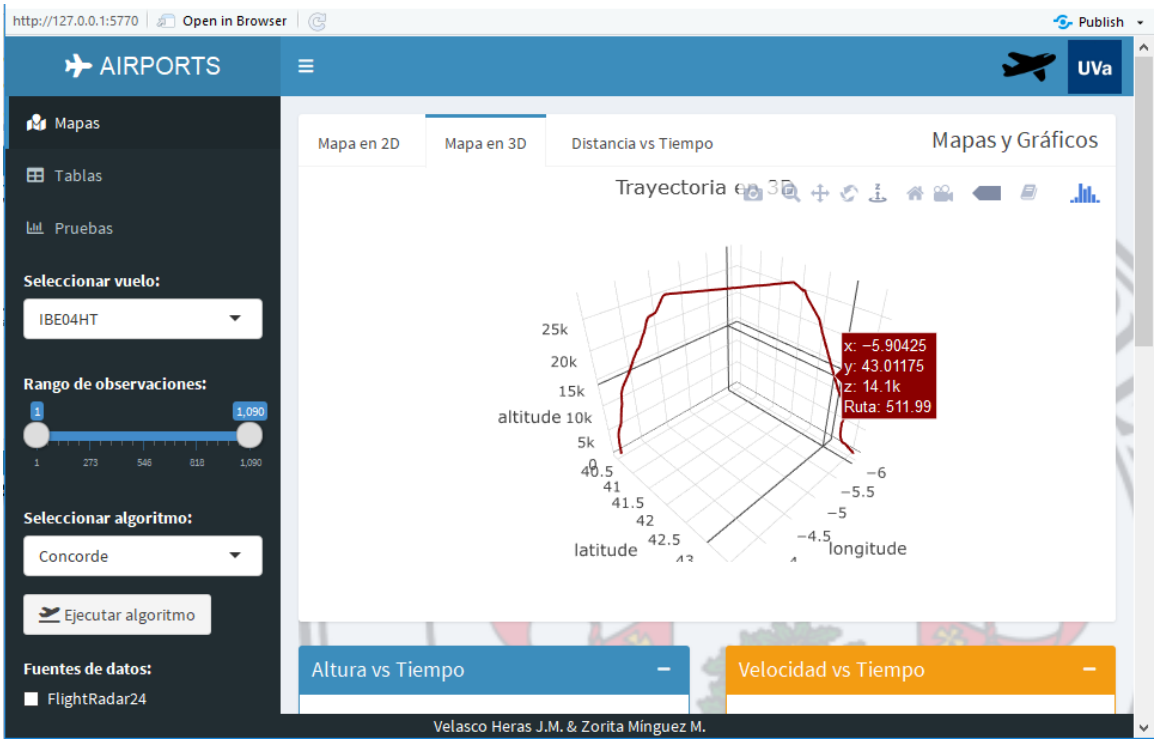


Figura 5.10: Mapa en 3D sin ejecutar algoritmo y con filtrado de proveedores

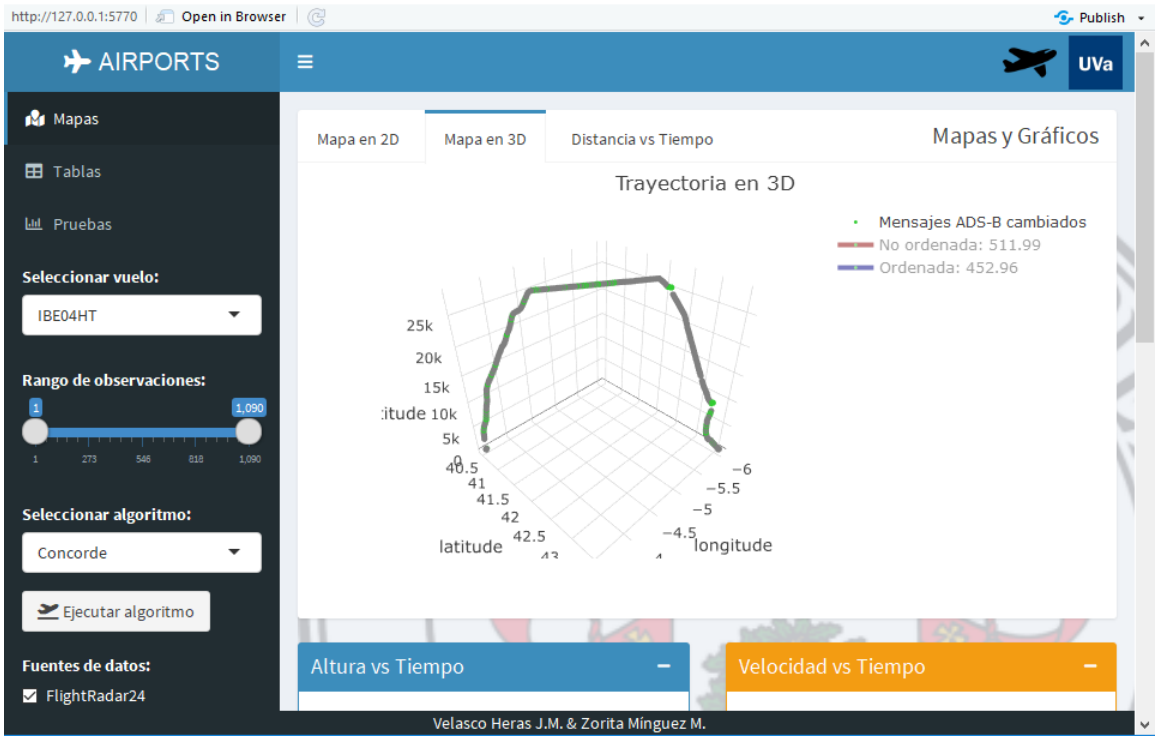


Figura 5.11: Parte superior pestaña mapa en 3D tras ejecutar algoritmo

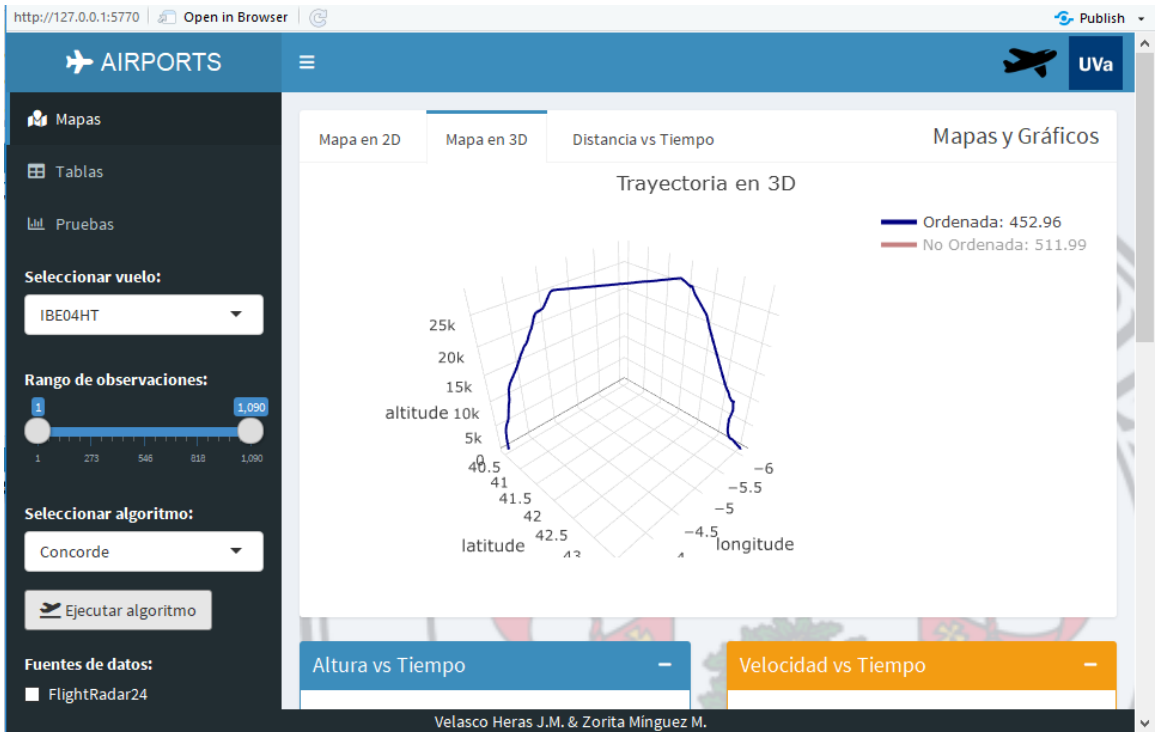


Figura 5.12: Mapa en 3D tras ejecutar algoritmo y con filtrado de proveedores

Dashboard Distancia vs Tiempo: En esta pestaña se muestra la representación gráfica de la reasignación de tiempos o *timestamps* para aquellos puntos que han sido alterados tras ejecutar un determinado algoritmo de resolución. En el caso en que no se haya ejecutado ningún algoritmo aparece el siguiente mensaje: "Es necesario realizar una ejecución para poder visualizar este gráfico" y en el caso en que se hayan eliminado los datos de todos los proveedores aparece el mensaje: "Es necesario disponer de los datos asociados a un proveedor como mínimo para poder visualizar el gráfico". Los puntos alterados se visualizan como en el resto de mapas de color verde y los que no han sufrido cambios de color gris. En el caso en que todos los puntos hayan alterado su posición, no se podrá realizar la interpolación y se mostrará el siguiente mensaje: "No se han podido corregir los *timestamps* porque todos los puntos se han visto alterados por la reordenación".

En el caso en que haya al menos un punto bueno (no ha sufrido cambios tras la reordenación), se ha usado un modelo de programación lineal que consiste en lo siguiente:

1. Se comprueba si el primer o el último punto son malos (han cambiado de orden tras la ejecución) pues en ese caso son los primeros que se corrigen. En caso de que sean malos y haya columna de velocidades en el dataset considerado, se calcula la velocidad de dicho punto como la media aritmética de las velocidades hasta el primer punto bueno (para el caso del primer punto se mira de él en adelante y para el último punto de él hacia atrás, excluyendo las velocidades nulas). Si no hay velocidades, se supone que esta es proporcional a la altura para hallar la velocidad de dicho punto. Tras ello, se obtiene la velocidad $v_{primero}$ y $v_{último}$.

2. Una vez se obtiene el valor de $v_{primero}$ se estima el valor del tiempo para el primer punto según la fórmula:

$$t_{primeroNuevo} = t_{primerBueno} - \frac{d_{primerBueno}}{v_{primero} + 0.1}$$

   donde $t_{primerBueno}$ es el tiempo del primer punto bueno (no ha sufrido cambios de orden) siguiente al punto 1 y $d_{primerBueno}$ es la distancia existente entre dicho punto con respecto al primero tras la ordenación. Tras ello, el punto 1 se considera bueno.

3. Una vez se obtiene el valor de $v_{último}$ se estima el valor del tiempo para el último punto según la fórmula:

$$t_{últimoNuevo} = t_{últimoBueno} + \left(\frac{d_{últimoNuevo} - d_{últimoBueno}}{v_{último} + 0.1} \right)$$

   donde $t_{últimoBueno}$ es el tiempo del punto bueno (no ha sufrido cambios de orden) anterior al último punto, $d_{últimoBueno}$ es la distancia existente entre dicho punto con respecto al primero y $d_{últimoNuevo}$ es la distancia existente entre el último punto con respecto al primero tras la ordenación. Tras ello el último punto se considera bueno.

4. Luego se hace algo similar para los puntos centrales que son malos, y por lo tanto, tienen que ser interpolados. Para cada uno de los que sean malos, se considera el anterior bueno denotado por $p_{anteriorBueno}$ y el posterior bueno denotado por $p_{posteriorBueno}$. En el caso

que haya columna de velocidades en el dataset considerado, se calcula el valor de los pesos que se denota por p_0 y p_1. El valor de p_0 se obtiene como la media aritmética de las velocidades desde el anterior punto bueno hasta el punto a interpolar excluyendo los puntos nulos. De la misma manera, el valor de p_1 se obtiene como la media aritmética de las velocidades desde el punto a interpolar hasta el siguiente punto bueno excluyendo los puntos nulos. Si no hay velocidades, se supone que esta es proporcional a la altura para hallar los valores de los pesos p_0 y p_1 del punto a interpolar.

5. Finalmente, para hallar el valor del *timestamp* que le corresponde al punto que se desea interpolar, se usa la siguiente expresión:

$$t_{puntoInterpolar} = \frac{(d_1 p_1 + 1) t_{anteriorBueno} + (d_0 p_0 + 1) t_{posteriorBueno}}{d_1 p_1 + d_0 p_0 + 2}$$

donde d_1 es la distancia existente entre $p_{posteriorBueno}$ y el punto a interpolar y d_0 es la distancia existente entre $p_{anteriorBueno}$ y el punto a interpolar.

En las Figuras 5.13 y 5.14 se muestra dicha representación gráfica, donde el eje horizontal se corresponde con el *timestamp* de cada punto y el eje vertical con la distancia entre cada punto y el punto inicial. En este caso se ha usado el método de ejecución *Concorde*. En la imagen de la Figura 5.14 se muestra dicha representación de forma ampliada, y en ella se puede ver que la interpolación llevada a cabo es lineal. El estudio y análisis realizado ha permitido observar que donde se produce un mayor número de alteraciones es en las zonas de los aeropuertos, esto es, en el despegue y aterrizaje. Como ya se comentó, es debido a que en estas zonas las trayectorias seguidas tienen más quiebros y zig-zag, mientras que en la zona del vuelo son más rectas.

Figura 5.13: Corrección de *timestamps* para los puntos alterados

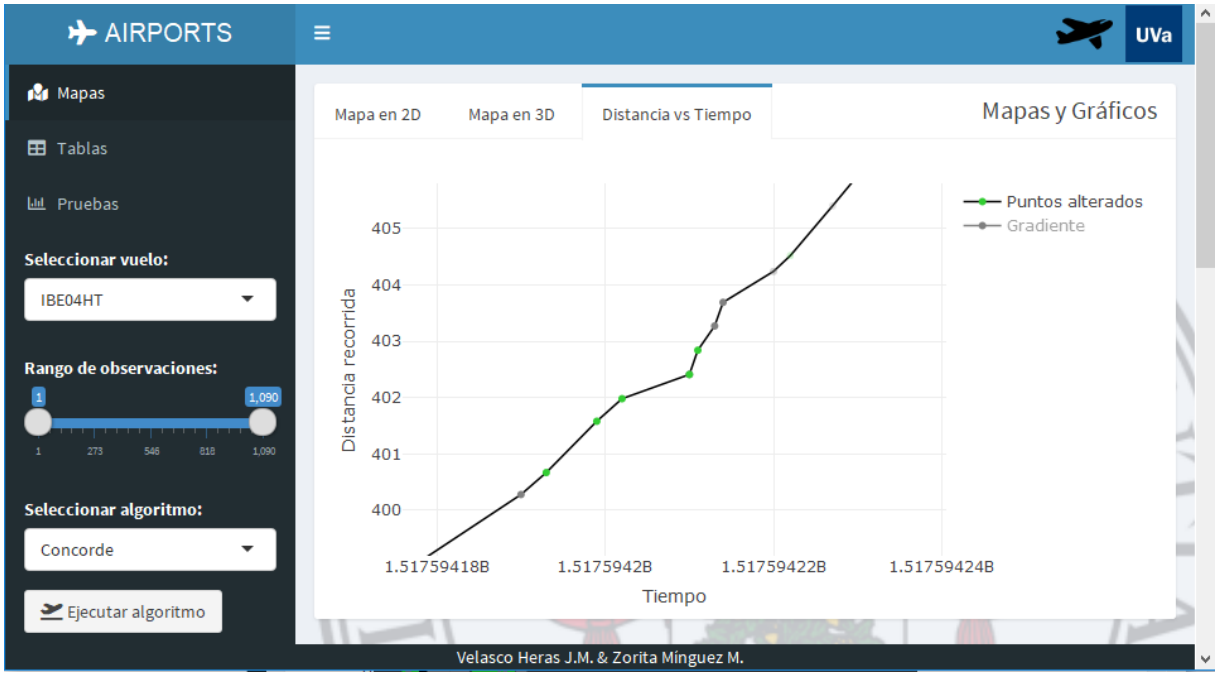


Figura 5.14: Corrección de *timestamps* para los puntos alterados (ampliado)

Además, se ofrece la posibilidad de visualizar en la representación un gradiente de colores que depende de la velocidad de cada punto. Esto ha ayudado a determinar dónde hay más o menos errores en las distintas trayectorias ya que en las zonas de los aeropuertos la velocidad es menor que en el vuelo. Esto se puede ver en la imagen de la Figura 5.15 donde la zona del aeropuerto es la que se corresponde con el color amarillo.

Figura 5.15: Gradiente en función de la velocidad de cada punto

Dashboard Table: También, es posible visualizar un listado con los datos de un determinado vuelo (el indicado en el selector *"Seleccionar vuelo"*), y para ello, se debe seleccionar en el menú lateral izquierdo la opción *"Tablas"*. Esto mostrará una apariencia como la de la interfaz de la Figura 5.16. Desde ahí usando el filtrador podemos buscar aquellas filas que tengan alguna coincidencia con la información indicada. También podemos ir visualizando las diferentes páginas de datos mediante un paginador al igual que desplazarnos en horizontal por la tabla con un scroll. Por último, con el slider asociado al rango podemos restringir los datos a visualizar en la tabla, así como se podía hacer para la visualización de los mapas y gráficas.

Conviene mencionar que tras ejecutar un método se añaden cuatro columnas más a la tabla: *"tag_original"* que indica la posición de los datos antes de la ordenación, *"tag_ordenado"* que almacena su posición tras haber sido ordenados con el método indicado, *"distancia"* que calcula la distancia de cada punto con respecto al primero y por último *"timestamp_ordenado"* que almacena el tiempo reasignado tras la ordenación (cambia para los puntos que han sido alterados). El uso de las columnas *"tag_original"* y *"tag_ordenado"* permite conocer los puntos que han sido alterados y los que no al aplicar el algoritmo indicado, para así poder visualizarlos en los mapas y asignarles el *"timestamp_ordenado"*. Esto puede verse en la interfaz de la Figura 5.17.

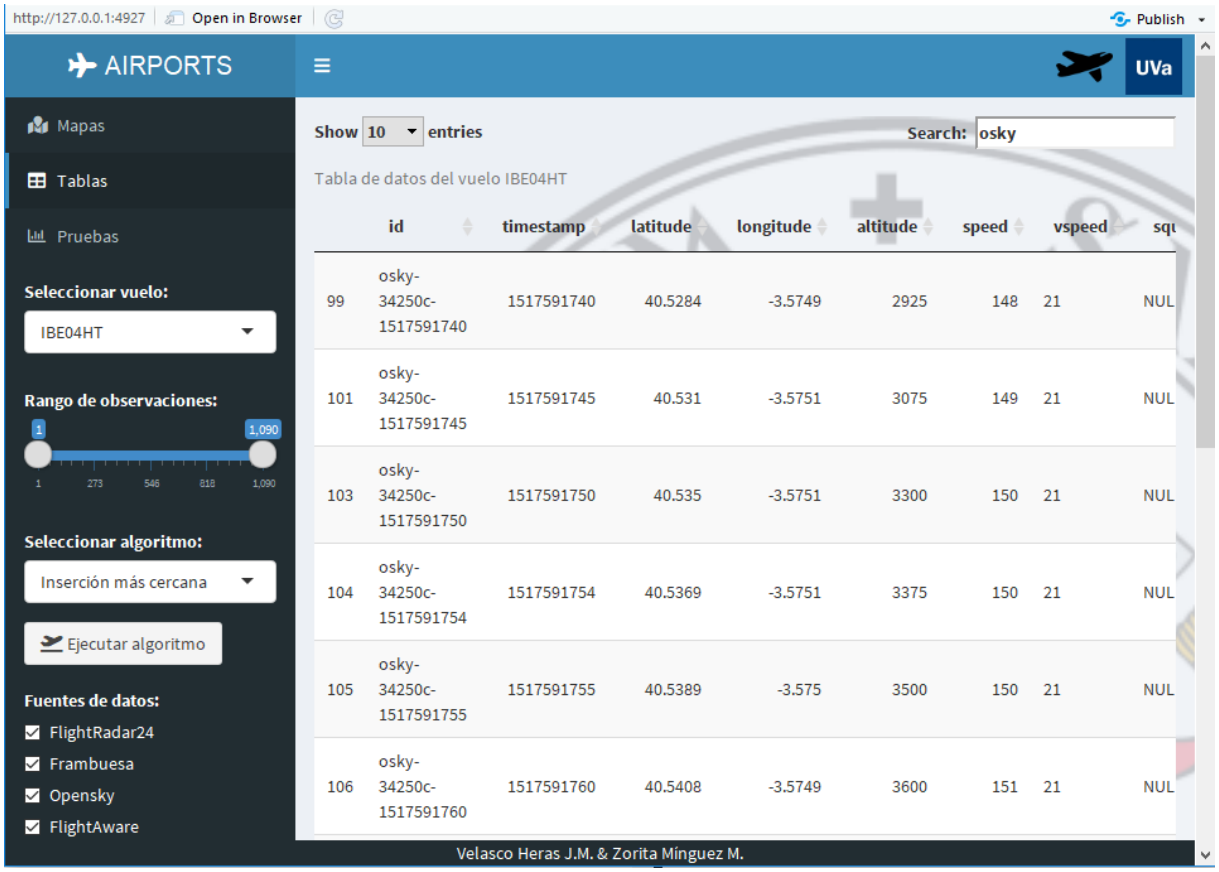


Figura 5.16: Tabla de datos con filtrado por proveedor osky

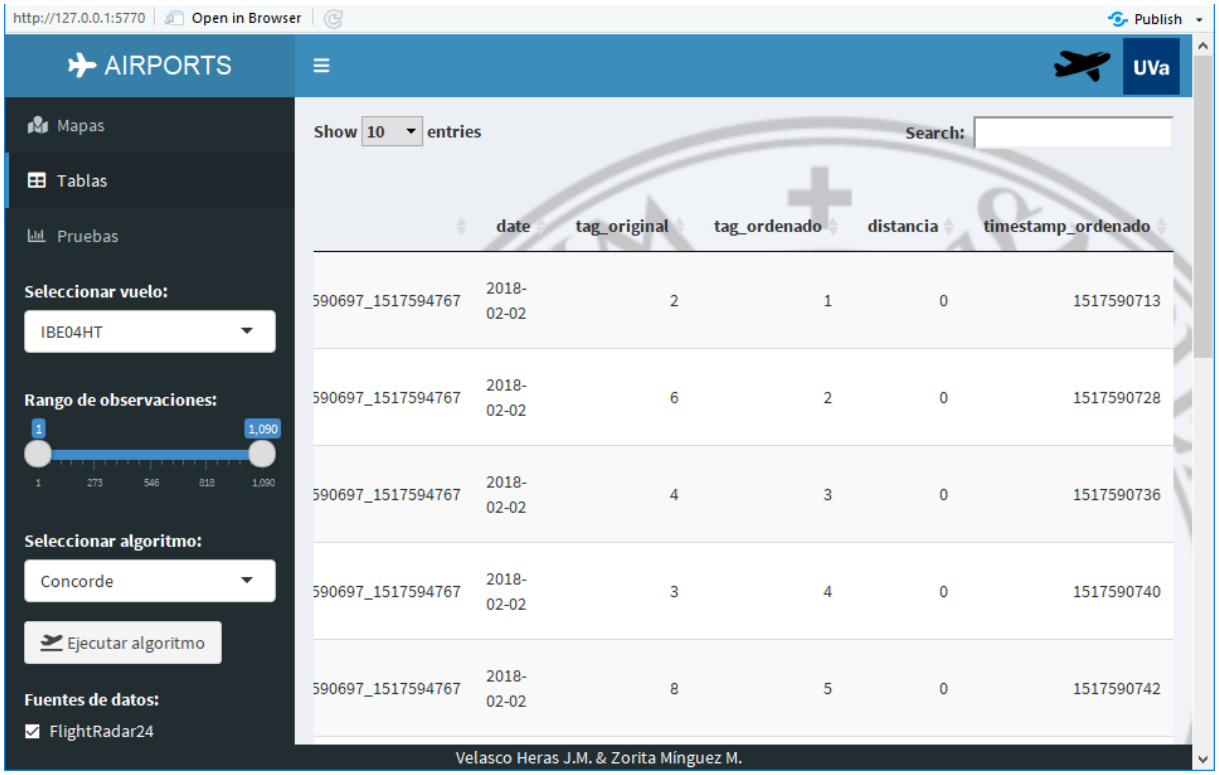


Figura 5.17: Tabla de datos con columnas añadidas tras ejecutar un algoritmo

En la imagen que se muestra en la Figura 5.18 se puede ver también el efecto que produce realizar el filtrado por proveedores. En esta imagen se ha dejado sólo el proveedor Frambuesa y como en dicho vuelo no había datos de ese proveedor, no aparece ningún dato en la tabla.

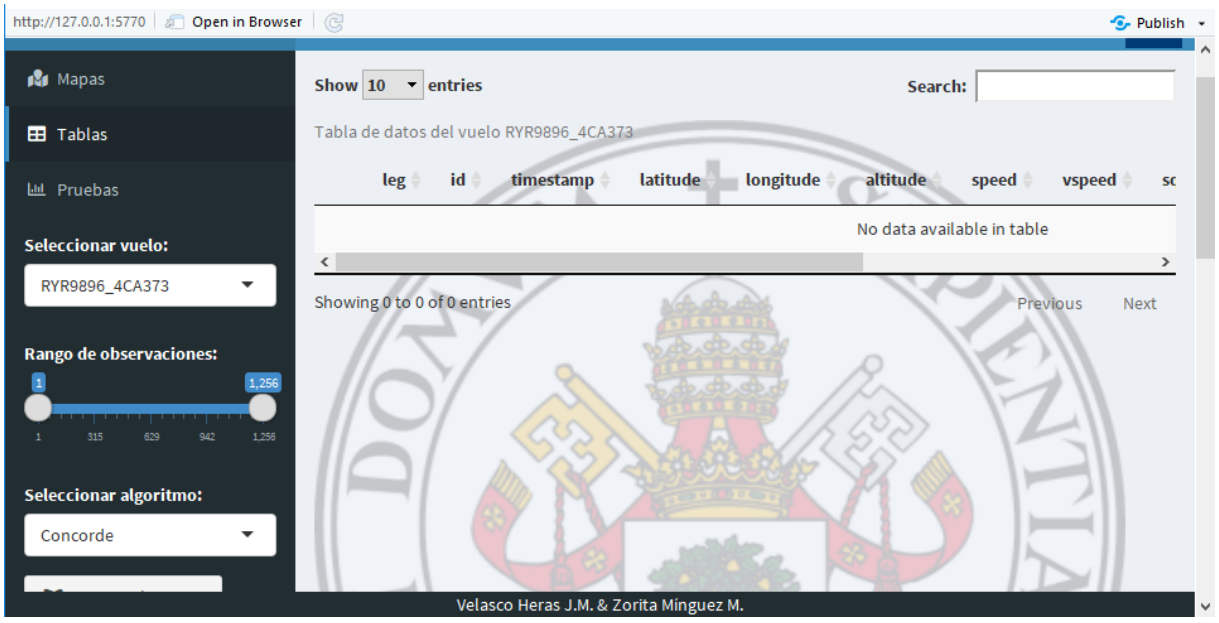


Figura 5.18: Tabla de datos con filtrado por proveedores

Dashboard Batería de Pruebas: Otro de los aspectos interesantes del dashboard es que permite realizar una batería de pruebas para las opciones deseadas. Para ello, se debe seleccionar la pestaña *"Pruebas"*, y tras ello, indicar en el selector *"Elegir vuelos"* los vuelos deseados, en el selector *"Elegir métodos"* los métodos a aplicar (ver Figura 5.19) y en el checkbox *"Ventanas"* el tipo de ventanas que se quiere usar para su resolución. Al igual que en la parte de los mapas, en función de la opción elegida aparecerán los parámetros u opciones necesarios para su ejecución. También en el caso de que algún valor sea erróneo, se mostrará un mensaje informativo al usuario (ver Figura 5.20).

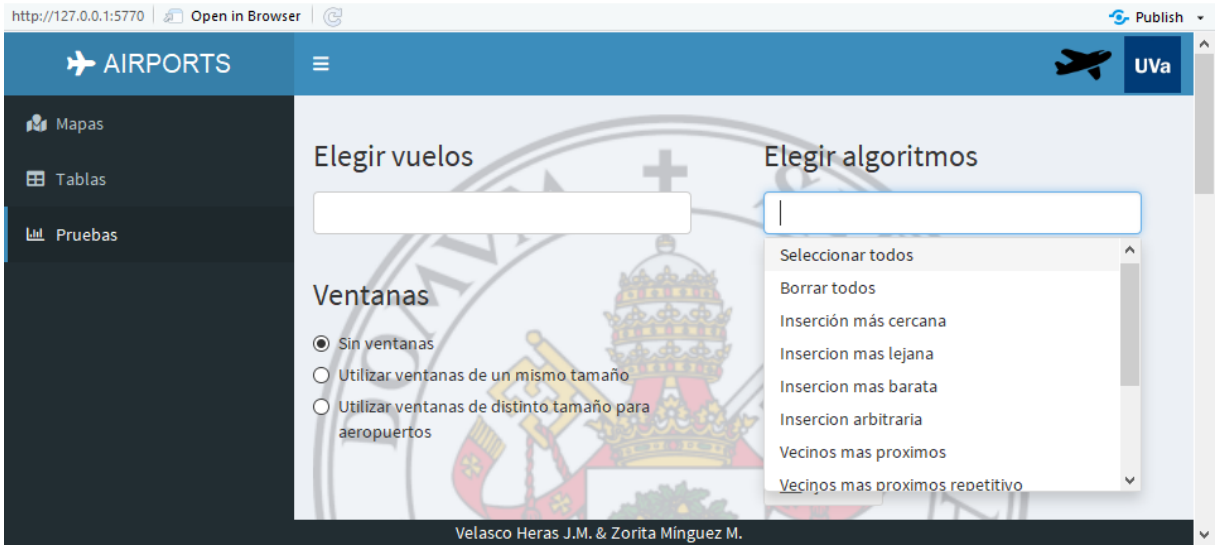


Figura 5.19: Selector para elegir algoritmos

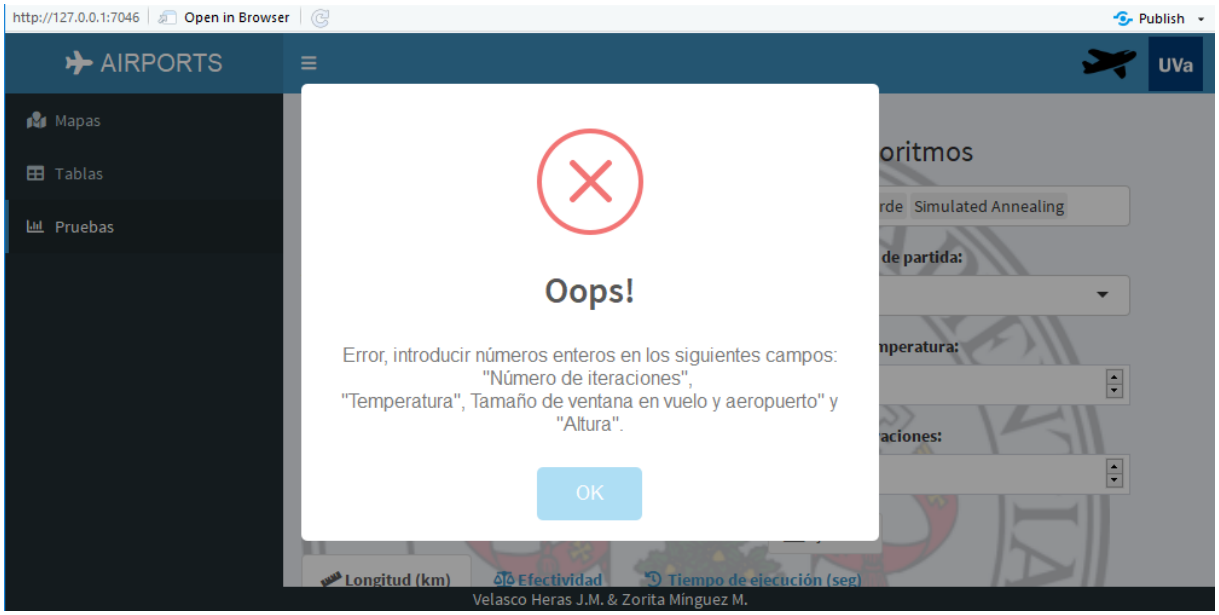


Figura 5.20: Mensaje de error en campo numérico

A continuación, en la Figura 5.21, se ha elegido la opción que tiene más parámetros y opciones a determinar.

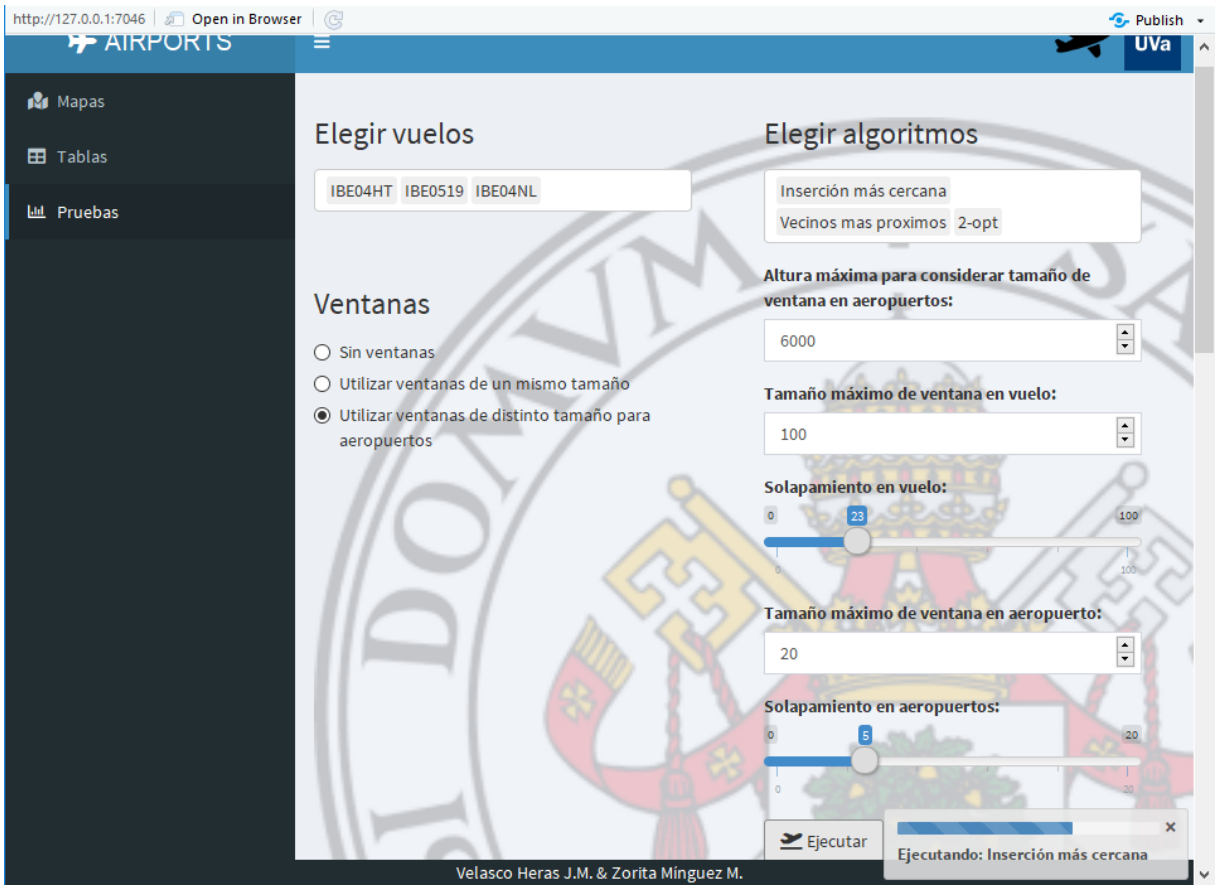


Figura 5.21: Ejecutando una batería de pruebas

Tras realizar todas las ejecuciones indicadas, se visualiza una tabla de resultados donde aparece la longitud obtenida (Figura 5.22), la eficacia obtenida (Figura 5.23) y el tiempo de ejecución (Figura 5.24) para cada uno de los métodos aplicados a cada uno de los vuelos elegidos. Además, para todos los resultados se aplican diversas métricas con el objetivo de poder visualizar de forma rápida, que algoritmo es el más adecuado. Las métricas usadas son la media aritmética, la desviación absoluta media conocida como MAD y la desviación estándar muestral. Se tiene que $MAD = Mediana(|x_i - Mediana(X)|)$ siendo X el conjunto de valores considerados, esto es, $X = \{x_i : i \in \mathbb{N}^+\}$.

El valor de la efectividad o eficacia, se ha calculado restando a la distancia original la distancia obtenida y dividiendo este resultado por la distancia original. Todo ello se multiplica por 100 obteniendo un valor en tanto por ciento. Así, aquellos métodos que obtengan un mayor valor serán los más eficientes. Además, se pueden obtener valores negativos en el caso en que la aplicación del método aumente la longitud de la ruta de partida. En este caso dichos valores aparecen en color rojo como se observa en la Figura 5.23. Por último, existe la posibilidad de guardar los resultados obtenidos de las ejecuciones en formato *csv*, y para ello, basta con pulsar el botón *Descargar* que aparece justo debajo de cada tabla.

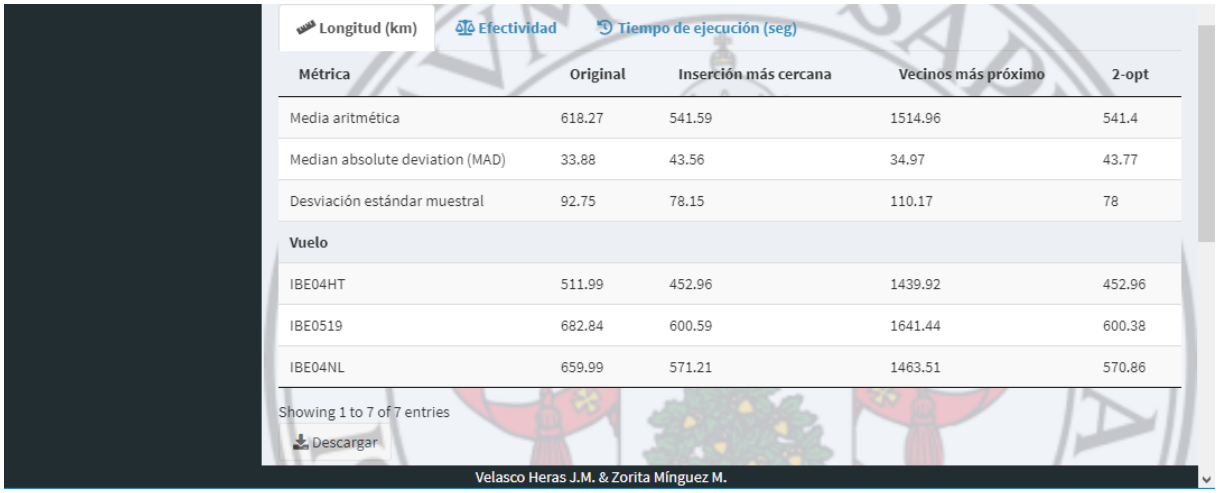


Figura 5.22: Resultado de la batería de pruebas para la longitud

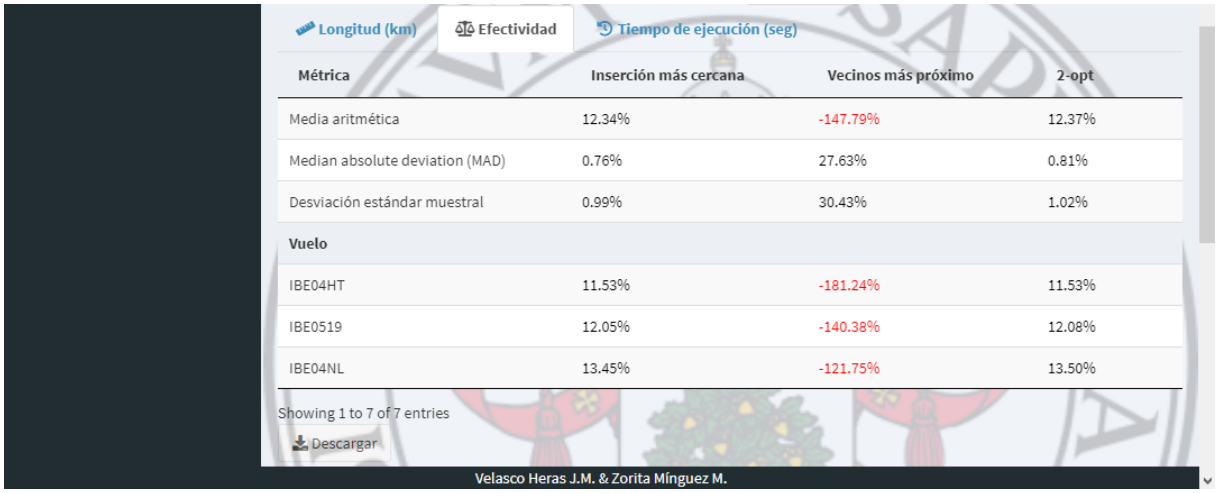


Figura 5.23: Resultado de la batería de pruebas para la efectividad

Figura 5.24: Resultado de la batería de pruebas para el tiempo de ejecución

Capítulo 6

Análisis de los resultados obtenidos

En este capítulo, se comenzará realizando un análisis de los resultados obtenidos al ejecutar el método de resolución *Simulated Annealing* a un conjunto de 96 vuelos proporcionados por los tutores. Se aplicará dicho método con todas las posibles configuraciones de ventanas (sin usar ventanas de tiempo, usando ventanas de tiempo del mismo tamaño y con ventanas de tiempo de distinto tamaño en la zona del aeropuerto).

Después se realizará un estudio general de los diferentes algoritmos de resolución descritos en la Sección 4.6 del Capítulo 4, y tras ello, se llevará a cabo una comparativa de los mismos en función de los resultados obtenidos en cuanto a distancia recorrida y tiempo de ejecución. Esto permitirá comparar nuestro algoritmo de estudio con el resto.

Para su realización se va a hacer uso del dashboard implementado en el entorno *R-Studio*, en concreto se usará la parte asociada a la batería de pruebas, ya que permite ejecutar todos los vuelos de una sola vez.

6.1. *Simulated Annealing* usando diferentes configuraciones de ventanas

Durante el desarrollo de esta sección se va a realizar un análisis y estudio sobre el comportamiento del algoritmo *Simulated Annealing* para ver bajo qué configuraciones de ventanas o situaciones obtiene mejores resultados. Para ello, se va a usar un conjunto de 96 vuelos.

Así, usando la batería de pruebas de *R-Studio*, obtendremos de manera dinámica diferentes tablas de resultados similares a las de las interfaces de las Figuras 5.22, 5.23 y 5.24. Dichas tablas nos permitirán comparar los resultados obtenidos, y en este caso, al tener un conjunto grande de vuelos, nos fijaremos en los resultados de las distintas métricas consideradas (media aritmética, median absolute deviation (MAD) y la desviación estándar muestral). Una vez se tengan dichos resultados, se analizarán para decidir bajo qué configuraciones se comporta mejor o peor el algoritmo de estudio.

La máquina usada para realizar las distintas pruebas es un ordenador portátil HP Pavilion x360 Convertible con un procesador Intel Core i5-7200U, cuya velocidad varía entre 2.5 GHz y 2.7 GHz, con memoria RAM de 8 GB y Sistema Operativo Windows 10 Home.

Siguiendo la notación del Capítulo 4 se considerará:

- **Para ventanas del mismo tamaño**: En este caso, t será el tamaño de ventana y s el solapamiento tanto en la zona del vuelo como en los aeropuertos.
- **Para ventanas de distinto tamaño en la zona del aeropuerto y en vuelo**: En este caso, t_v será el tamaño de ventana en vuelo, s_v el solapamiento en vuelo, t_a el tamaño de ventana en aeropuerto y s_a el solapamiento en aeropuerto.

Para realizar las pruebas se ha considerado un tamaño de ventana de 100 con solapamiento de 20 cuando las ventanas son del mismo tamaño, y para el caso en que se hace distinción en la zona de los aeropuertos, se ha considerado un tamaño de ventana de 20 y solapamiento de 5 en dichas zonas. Como ya se comentó, la disminución del tamaño de ventana y de solapamiento en estas zonas con respecto a la zona central del vuelo es debido a que al realizar el avión más maniobras, las trayectorias suelen presentar más quiebros y zig-zag, siendo más adecuado en estos casos reducir el número de datos considerados para realizar la ejecución del algoritmo.

A continuación, se muestran los resultados obtenidos de las distintas ejecuciones realizadas usando un determinado valor de temperatura y número de iteraciones. Para las distintas configuraciones de ventanas usadas se muestran los resultados obtenidos sobre la distancia de la trayectoria ordenada (en kilómetros), la eficacia obtenida (en porcentaje) y el tiempo de ejecución (en segundos) respectivamente.

Prueba 1: 10000 iteraciones y temperatura 3 $\Longrightarrow$

Configuración usada	Media aritmética	MAD	Desviación estándar muestral
Sin aplicar algoritmo de resolución	1868.65 km	948.83 km	1265.18 km
***Simulated Annealing* → Distancia tras reordenar (km)**			
Sin usar ventanas	1852.44 km	948.92 km	1246.33 km
$\mathbf{t = 100}$ **y** $\mathbf{s = 20}$	1635.42 km	871.94 km	960.58 km
$\mathbf{t_v = 100}$, $\mathbf{s_v = 20}$, $\mathbf{t_a = 20}$ **y** $\mathbf{s_a = 5}$	1631.99 km	860.44 km	956.80 km
***Simulated Annealing* → Eficacia obtenida (%)**			
Sin usar ventanas	0.54 %	0.03 %	3.03 %
$\mathbf{t = 100}$ **y** $\mathbf{s = 20}$	7.53 %	4.20 %	13.15 %
$\mathbf{t_v = 100}$, $\mathbf{s_v = 20}$, $\mathbf{t_a = 20}$ **y** $\mathbf{s_a = 5}$	7.66 %	3.98 %	12.92 %
***Simulated Annealing* → Tiempo de ejecución (sg)**			
Sin usar ventanas	2.81 sg	1.23 sg	1.17 sg
$\mathbf{t = 100}$ **y** $\mathbf{s = 20}$	23.48 sg	13.38 sg	12.50 sg
$\mathbf{t_v = 100}$, $\mathbf{s_v = 20}$, $\mathbf{t_a = 20}$ **y** $\mathbf{s_a = 5}$	33.52 sg	14.93 sg	14.05 sg

Tabla 6.1: *Simulated Annealing* con 10000 iteraciones y temperatura 3

Prueba 2: 10000 iteraciones y temperatura 1 $\Longrightarrow$

Configuración usada	Media aritmética	MAD	Desviación estándar muestral
Sin aplicar algoritmo de resolución	1868.65 km	948.83 km	1265.18 km
***Simulated Annealing* → Distancia tras reordenar (km)**			
Sin usar ventanas	1852.65 km	948.80 km	1246.48 km
$\mathbf{t = 100}$ **y** $\mathbf{s = 20}$	1589.61 km	818.75 km	939.68 km
$\mathbf{t_v = 100}$, $\mathbf{s_v = 20}$, $\mathbf{t_a = 20}$ **y** $\mathbf{s_a = 5}$	1589.63 km	823.59 km	937.04 km
***Simulated Annealing* → Eficacia obtenida (%)**			
Sin usar ventanas	0.54 %	0.05 %	2.96 %
$\mathbf{t = 100}$ **y** $\mathbf{s = 20}$	9.93 %	6.72 %	12.99 %
$\mathbf{t_v = 100}$, $\mathbf{s_v = 20}$, $\mathbf{t_a = 20}$ **y** $\mathbf{s_a = 5}$	9.93 %	6.76 %	12.92 %
***Simulated Annealing* → Tiempo de ejecución (sg)**			
Sin usar ventanas	6.04 sg	2.82 sg	2.44 sg
$\mathbf{t = 100}$ **y** $\mathbf{s = 20}$	24.31 sg	12.74 sg	12.31 sg
$\mathbf{t_v = 100}$, $\mathbf{s_v = 20}$, $\mathbf{t_a = 20}$ **y** $\mathbf{s_a = 5}$	42.35 sg	19.27 sg	17.68 sg

Tabla 6.2: *Simulated Annealing* con 10000 iteraciones y temperatura 1

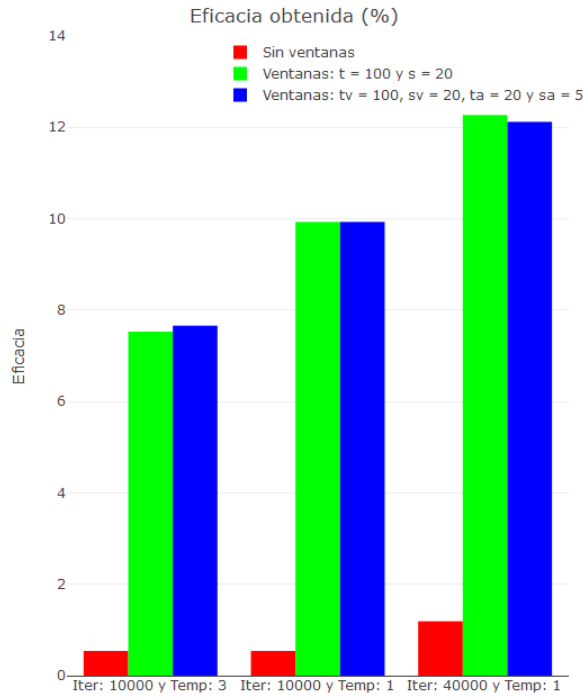
Prueba 3: 40000 iteraciones y temperatura 1 $\Longrightarrow$

Configuración usada	Media aritmética	MAD	Desviación estándar muestral
Sin aplicar algoritmo de resolución	1868.65 km	948.83 km	1265.18 km
***Simulated Annealing* → Distancia tras reordenar (km)**			
Sin usar ventanas	1832.83 km	945.80 km	1222.41 km
$\mathbf{t = 100}$ **y** $\mathbf{s = 20}$	1537.40 km	803.01 km	905.29 km
$\mathbf{t_v = 100}$, $\mathbf{s_v = 20}$, $\mathbf{t_a = 20}$ **y** $\mathbf{s_a = 5}$	1540.18 km	800.00 km	905.18 km
***Simulated Annealing* → Eficacia obtenida (%)**			
Sin usar ventanas	1.19 %	0.10 %	5.24 %
$\mathbf{t = 100}$ **y** $\mathbf{s = 20}$	12.27 %	8.93 %	14.31 %
$\mathbf{t_v = 100}$, $\mathbf{s_v = 20}$, $\mathbf{t_a = 20}$ **y** $\mathbf{s_a = 5}$	12.12 %	8.78 %	14.18 %
***Simulated Annealing* → Tiempo de ejecución (sg)**			
Sin usar ventanas	11.35 sg	5.24 sg	4.75 sg
$\mathbf{t = 100}$ **y** $\mathbf{s = 20}$	100.17 sg	58.18 sg	50.72 sg
$\mathbf{t_v = 100}$, $\mathbf{s_v = 20}$, $\mathbf{t_a = 20}$ **y** $\mathbf{s_a = 5}$	132.18 sg	62.55 sg	55.95 sg

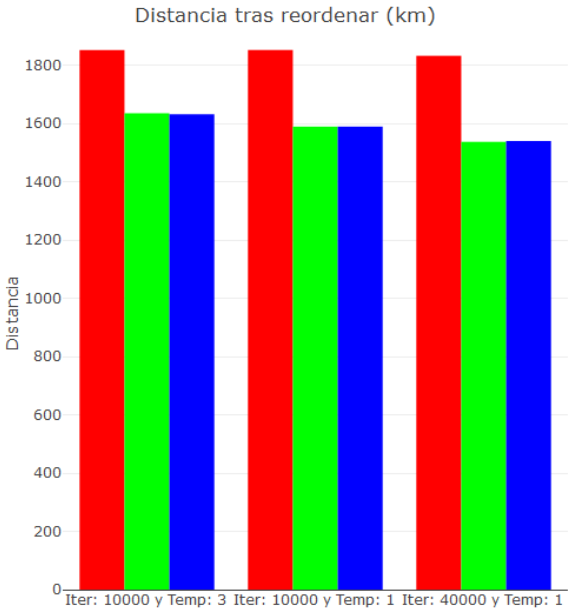
Tabla 6.3: *Simulated Annealing* con 40000 iteraciones y temperatura 1

Ahora, en la Figura 6.1 se muestran una serie de gráficas donde se representa el valor de la media aritmética obtenida para las distintas pruebas realizadas, teniendo en cuenta cada una de las configuraciones de ventanas consideradas (sin usar ventanas, usando ventanas del mismo tamaño y solapamiento en la zona del vuelo y en los aeropuertos, así como usando ventanas con distinto tamaño y solapamiento dependiendo de si se considera la zona del vuelo o de los aeropuertos).

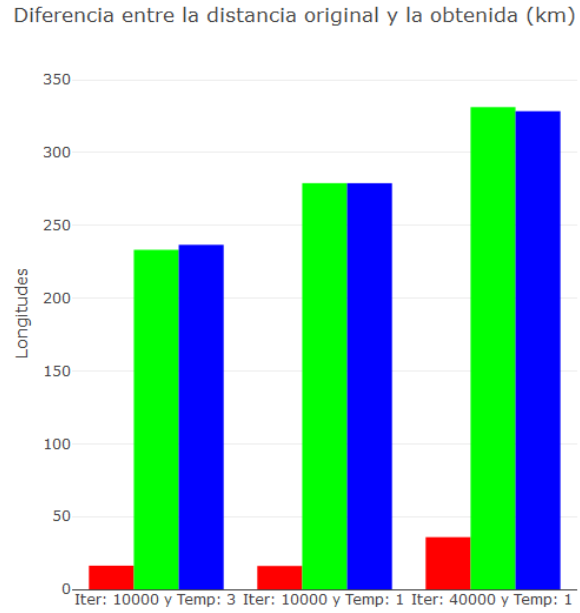
La Figura 6.1a se corresponde con la eficacia conseguida (en porcentaje), la Figura 6.1b con la distancia recorrida (en kilómetros), la Figura 6.1c con la diferencia entre la distancia original con respecto a la obtenida (en kilómetros) y la Figura 6.1d con el tiempo empleado (en segundos).



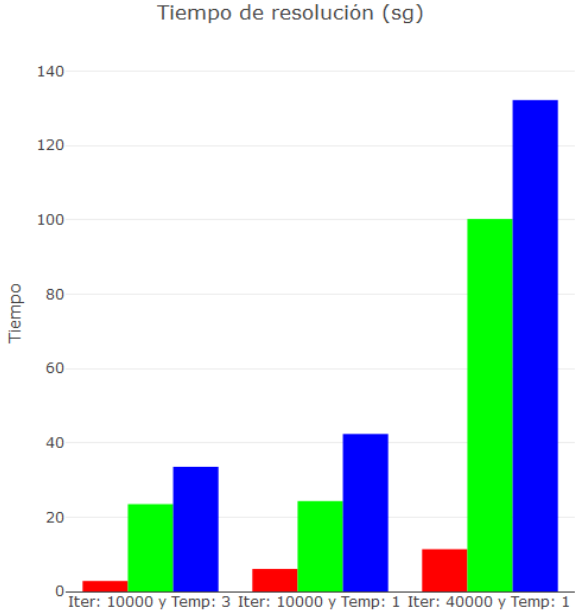
(a) Eficacia obtenida (%)



(b) Distancia recorrida (km)



(c) Diferencia entre la distancia original y la obtenida (km)



(d) Tiempo de resolución (sg)

Figura 6.1: Resultados del algoritmo *Simulated Annealing*

Tras los resultados obtenidos de las distintas ejecuciones realizadas en el dashboard implementado, se puede decir sobre el comportamiento de dicho algoritmo lo siguiente:

1. El algoritmo arroja mejores resultados cuando el valor de la temperatura se elige pequeño siendo el mejor valor el 1. Para ello, basta observar los datos de las Tablas 6.1 y 6.2, pues elegido el mismo número de iteraciones, se obtiene una efectividad inferior usando 3 como valor de la temperatura que si se usa el valor 1. Por lo tanto, la eficacia disminuye a mayor valor de temperatura elegido en el parámetro correspondiente del método *optim()*.

2. El algoritmo arroja mejores resultados cuando el número de iteraciones es mayor, sin embargo, esto causa que el tiempo de ejecución aumente de manera significativa. Por ello, dado que alcanzado un cierto límite de iteraciones la mejora no es muy relevante y el tiempo de ejecución crece demasiado, lo conveniente es buscar un equilibrio para que los resultados obtenidos sean aceptables tanto en términos de eficacia como en tiempo de ejecución. Un buen número de iteraciones a elegir se encuentra en el rango de 30000 a 50000, pues en el caso de ser inferior la mejora es pequeña y si es superior el tiempo de ejecución es muy grande. Esto se puede observar en los datos de las Tablas 6.2 y 6.3, ya que en ellas se ve como al aumentar de 10000 a 40000 iteraciones mejora la eficacia aumentando con ello el tiempo de ejecución.

3. Otro aspecto a tener en cuenta de las ejecuciones realizadas es ver como influye en los resultados obtenidos el uso de ventanas de tiempo. A partir de los datos de las 3 tablas y gráficos anteriores, podemos concluir que para dicho algoritmo usar ventanas de tiempo es recomendable pues mejora la eficacia obtenida, esto es, se reduce la distancia de la ruta ordenada. El inconveniente de su uso es que el tiempo de ejecución es algo mayor, sin embargo, dado que el aumento del tiempo no es demasiado y la mejora en eficacia es muy significativa, es de gran utilidad usar ventanas de tiempo, y por lo tanto, conviene usarlas. El aumento del tiempo de ejecución, como ya se comentó, se debe a que el número de iteraciones del algoritmo es fijo para cada ejecución (el cálculo del número de ejecuciones necesarias viene explicado en la Sección 4.7) y el número de ejecuciones del algoritmo crece dado que hay que considerar diferentes ventanas de tiempo, y además, un determinado solapamiento entre ellas.

   Para el caso en que se usen ventanas del mismo tamaño o de distinto tamaño la eficacia obtenida es similar, donde las mínimas diferencias obtenidas son consecuencia del tipo de vuelos considerados, esto es, si presentan más irregularidades en las zonas de los aeropuertos o menos. El comportamiento será similar a menos que los vuelos considerados sean muy irregulares en los aeropuertos ya que en ese caso el uso de ventanas de distinto tamaño arrojará mejores resultados. En nuestro caso eso no ocurre y por ello las diferencias son mínimas. En cambio, si nos fijamos en el tiempo de ejecución si se aprecian claras diferencias ya que podemos ver como cuando se usan ventanas de distinto tamaño este tiempo aumenta. Se debe a que como en las zonas del aeropuerto se reduce el tamaño de ventana y de solapamiento entre ellas, el número de ejecuciones también aumenta, y en consecuencia, el tiempo de resolución.

En conclusión, se puede decir que el algoritmo *Simulated Annealing* es adecuado para resolver el TSP, y por lo tanto, el problema planteado sobre el fallo en el alineamiento temporal, ya que en la mayoría de las ocasiones, logra mejorar la solución de partida aproximándose a la óptima (siempre que la elección de los parámetros sea la adecuada).

6.2. Comparativa de los métodos de resolución

Ahora, vamos a realizando una comparativa de todos los métodos de ejecución considerados, para así poder sacar conclusiones sobre cuáles arrojan mejores o peores resultados.

Conviene mencionar que esta parte del trabajo se ha realizado de manera conjunta con Juan Manuel Velasco Heras, esto es, ambos hemos considerado los mismos vuelos proporcionados por los tutores para realizar las pruebas. Por lo tanto, los resultados obtenidos son los mismos en ambos trabajos y las conclusiones de cada uno similares. La máquina usada para realizar las distintas pruebas es un ordenador portátil ACER Aspire 5 A515-51G-751G con un procesador Intel Core i7-7500U, cuya velocidad varía entre 2.7 GHz y 3.5 GHz, con memoria RAM de 8 GB y Sistema Operativo Windows 10 Home.

A continuación, en las Tablas 6.4 y 6.5 se realiza una comparativa de los distintos algoritmos comentados en términos de distancia recorrida y tiempo de ejecución. Conviene mencionar que para estas ejecuciones no se han usado ventanas de tiempo. Los vuelos considerados para realizar dicho estudio son 5 y son los siguientes:

- ***IBE04HT***: Vuelo de Madrid a Asturias que tiene 1090 mensajes ADS-B.
- ***IBE04NL***: Vuelo de Asturias a Madrid que tiene 1151 mensajes ADS-B.
- ***IBE0519***: Vuelo de A Coruña a Madrid que tiene 1105 mensajes ADS-B.
- ***IBE05DK***: Vuelo de Madrid a A Coruña que tiene 1079 mensajes ADS-B.
- ***RYR9KY_4CA97C***: Vuelo de Ibiza a Barcelona que tiene 533 mensajes ADS-B.

Algoritmo de resolución	***IBE04HT***		***IBE04NL***		***IBE0519***	
	Distancia	**Tiempo**	**Distancia**	**Tiempo**	**Distancia**	**Tiempo**
Sin aplicar algoritmo	511.99 km	-	659.99 km	-	682.84 km	-
Inserción más cercana	453.22 km	5.46 sg	588.89 km	7.24 sg	639.72 km	5.60 sg
Inserción más lejana	533.07 km	5.41 sg	570.51 km	6.14 sg	600.42 km	5.97 sg
Inserción más barata	452.95 km	3.76 sg	570.42 km	4.03 sg	600.38 km	3.89 sg
Inserción aleatoria	529.76 km	0.05 sg	576.98 km	0.05 sg	873.01 km	0.04 sg
Vecinos más próximo	518.98 km	0.05 sg	888.53 km	0.05 sg	1227.06 km	0.05 sg
Vecinos más próximo repetitivo	453.49 km	66.22 sg	570.76 km	67.54 sg	602.57 km	55.41 sg
2-opt	452.95 km	0.42 sg	570.42 km	0.63 sg	600.38 km	0.50 sg
Lin-Kernighan	452.97 km	20.85 sg	570.39 km	18.58 sg	600.38 km	22.14 sg
Concorde	452.97 km	8.08 sg	570.39 km	10.44 sg	600.38 km	8.05 sg
***SA* (40000 iteraciones)**	505.91 km	5.04 sg	657.75 km	5.08 sg	677.75 km	5.22 sg
***SA* (100000 iteraciones)**	503.99 km	12.20 sg	656.83 km	14.06 sg	664.79 km	12.56 sg
***SA* (1000000 iteraciones)**	482.97 km	141.86 sg	611.94 km	133.66 sg	641.93 km	120.99 sg

Tabla 6.4: Comparativa en distancia (km) y tiempo de resolución (sg)

Algoritmo de resolución	***IBE05DK***		***RYR9KY_4CA97C***	
	Distancia	**Tiempo**	**Distancia**	**Tiempo**
Sin aplicar algoritmo	602.47 km	-	346.98 km	-
Inserción más cercana	573.73 km	5.59 sg	300.85 km	0.78 sg
Inserción más lejana	801.13 km	4.97 sg	360.63 km	0.80 sg
Inserción más barata	558.69 km	3.04 sg	299.99 km	0.49 sg
Inserción aleatoria	653.80 km	0.05 sg	330.80 km	0.02 sg
Vecinos más próximo	913.47 km	0.04 sg	332.44 km	0.01 sg
Vecinos más próximo repetitivo	558.77 km	51.36 sg	310.66 km	6.75 sg
2-opt	558.69 km	0.28 sg	299.99 km	0.03 sg
Lin-Kernighan	558.69 km	16.97 sg	299.99 km	3.62 sg
Concorde	558.69 km	8.40 sg	299.99 km	2.49 sg
***SA* (40000 iteraciones)**	601.86 km	5.59 sg	340.84 km	3.65 sg
***SA* (100000 iteraciones)**	590.95 km	11.98 sg	327.98 km	9.17 sg
***SA* (1000000 iteraciones)**	578.84 km	120.21 sg	309.50 km	89.64 sg

Tabla 6.5: Comparativa en distancia (km) y tiempo de resolución (sg)

De los resultados obtenidos observamos que los mejores métodos en cuanto a distancia recorrida son el *Lin-Kernighan* y el *Concorde*. Además, observamos como el *2-opt* arroja muy buenos resultados y es debido a que en este caso la solución de partida no es muy lejana a la óptima, esto es, las trayectorias no son muy malas (en otro caso los resultados serían algo peores). Con respecto a los algoritmos que obtienen buenos resultados, el *2-opt* presenta la ventaja de tener un tiempo de ejecución bastante menor. También se observa que otros algoritmos como *Inserción más barata* y *Vecinos más próximo repetitivo* arrojan muy buenos resultados.

Por otro lado, para algoritmos como *Inserción más cercana, más lejana* y *Vecinos más próximos* vemos que el resultado depende de la ejecución realizada, de manera que los valores mostrados en las Tablas 6.4 y 6.5 podrían mejorarse si se realizan una serie de ejecuciones y se elige la mejor de ellas. Este es uno de los inconvenientes que presentan dichos algoritmos, pues en función del dato de partida considerado se obtienen mejores o peores resultados, aunque para el caso del algoritmo *Vecinos más próximos*, la distancia obtenida suele estar en la mayoría de las ocasiones lejos de la óptima. Además, para el algoritmo *Inserción aleatoria*, como la inserción de los nodos se realiza al azar, habrá diferencias en los resultados dependiendo de la ejecución realizada pues cada vez será diferente, arrojando en algunos casos mejores resultados que en otros. También se observa que debido al procedimiento seguido por el algoritmo *Vecinos más próximo repetitivo*, los resultados obtenidos son mejores que los del algoritmo *Vecinos más próximo*, sin embargo, para ello se necesita un tiempo de ejecución bastante mayor.

Por último vemos como el *Simulated Annealing* también obtiene buenos resultados, logrando aproximarse bastante a la solución óptima, pero para ello, es necesario realizar unas cuantas iteraciones lo que provoca un aumento del tiempo de ejecución. En este caso, para realizar las ejecuciones del algoritmo no se ha usado un algoritmo previo.
Aunque el *Simulated Annealing* mejora la solución de partida, como se ha podido ver,

existen otros algoritmos que son más adecuados para resolver el problema planteado, entre ellos el *Concorde* y las heurísticas de mejora local como es el caso del *2-opt* y *Lin-Kernighan*. El motivo de que se adapten mejor a dicho problema se debe principalmente a que han sido creados específicamente para resolver el problema del viajante o TSP. En cambio, no ocurre lo mismo para el caso del algoritmo *Simulated Annealing*, ya que a diferencia del resto, puede ser aplicado para resolver múltiples problemas, pues partiendo de un espacio de estados inicial, una función a optimizar, una función generadora de estados y una medida de la diferencia de coste entre soluciones o estados, dicho algoritmo logra optimizar cualquier función objetivo dada.

A continuación, se muestran dos gráficas en las Figuras 6.2 y 6.3, donde se representa para cada uno de los algoritmos de las Tablas 6.4 y 6.5 la diferencia entre la distancia original de cada uno de los vuelos considerados con respecto a la distancia tras aplicar un determinado algoritmo de resolución. Además, para cada uno de los vuelos considerados, se indica cuál es la ciudad de origen y de destino.

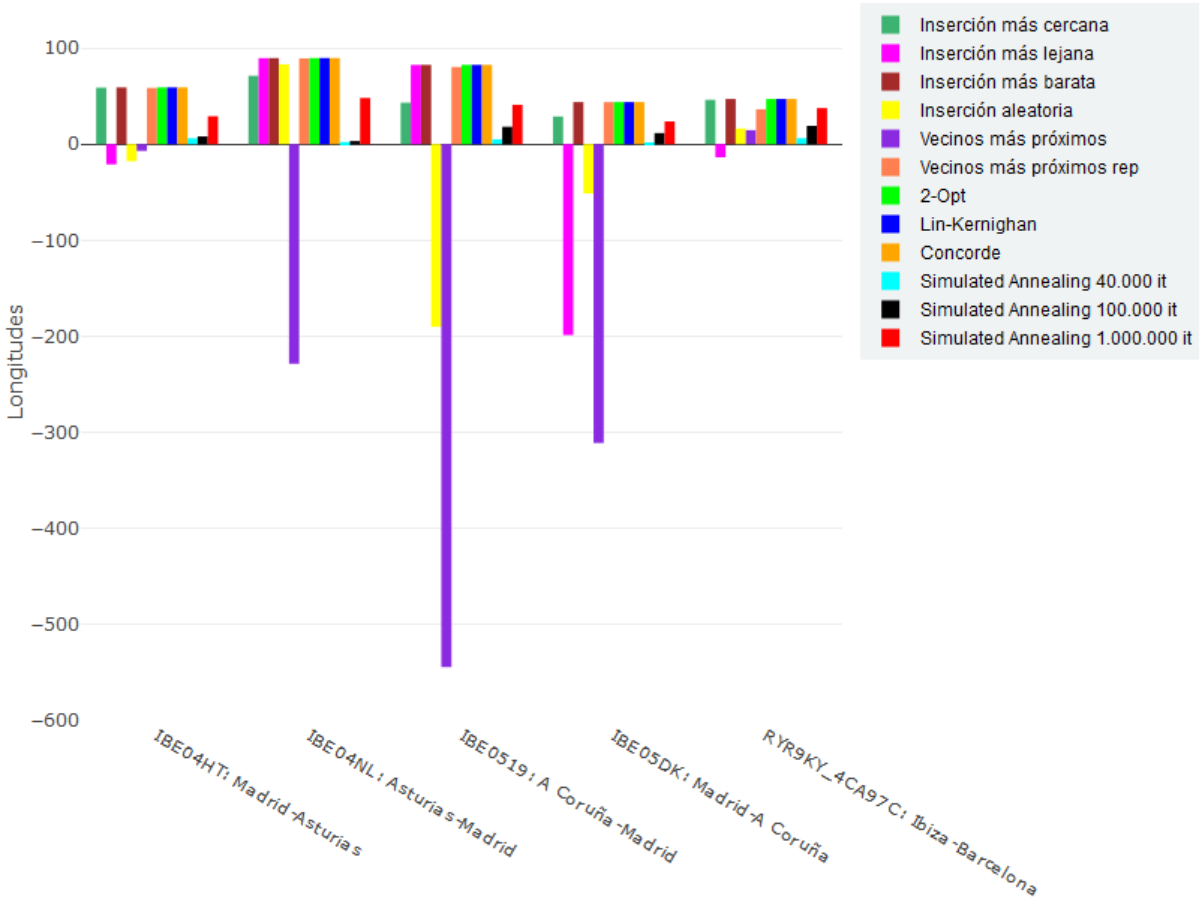


Figura 6.2: Comparativa de todos los algoritmos de resolución

En esta primera gráfica (ver Figura 6.2) se puede ver como para algunos algoritmos se obtienen diferencias negativas, ya que en ocasiones, la ejecución de dichos algoritmos arroja resultados peores que los de partida. Por otro lado, en la segunda gráfica (ver Figura 6.3),

se muestra la representación asociada al método *Concorde*, las heurísticas de mejora local como son los algoritmos *2-opt* y *Lin-Kernighan* y las heurísticas de mejora basadas en el azar o aleatoriedad como es el caso del algoritmo *Simulated Annealing*.

Se puede observar como para el caso del *Simulated Annealing* si se usa un adecuado número de iteraciones la diferencia con respecto a los otros algoritmos no es elevada. Además, como ya se mostró en la sección anterior, los resultados de dicho algoritmo se aproximan mucho más a la solución óptima cuando se considera el uso de ventanas de tiempo, lo cual, no fue considerado en este caso. Por lo tanto, si se hubiera tenido en cuenta, la diferencia con el resto de algoritmos se reduciría.

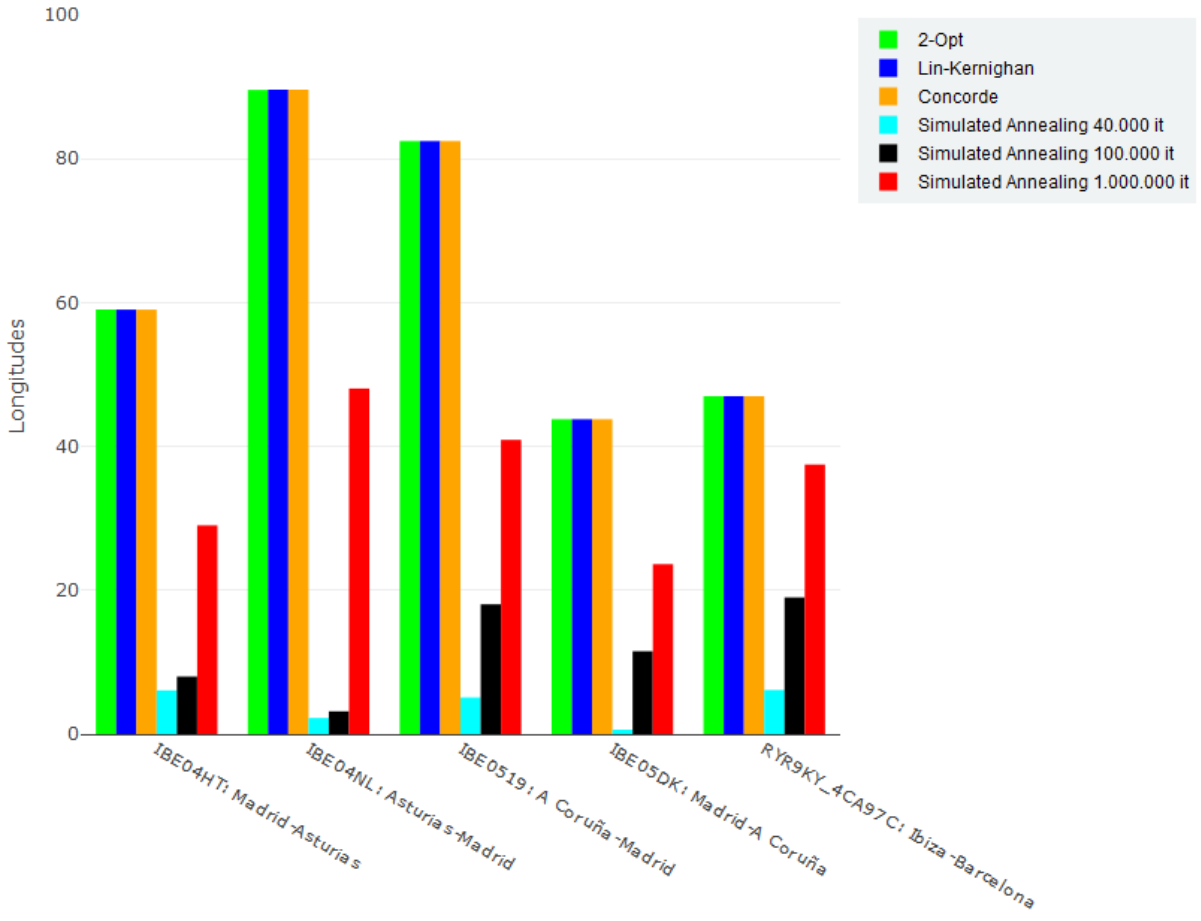


Figura 6.3: Comparativa de algunos algoritmos de resolución

Cabe decir que se han elegido vuelos en los que existe una cierta mejora entre la ruta de partida y la obtenida cuando se aplican los algoritmos considerados, para que así se pueda llevar a cabo la comparación de los mismos en función de los resultados que obtienen en términos de distancia y tiempo de resolución. Por lo general, en vuelos muy cortos la mejora es prácticamente nula para algunos algoritmos (los que mejor funcionan), mientras que en vuelos largos las mejoras en cuanto a distancia suelen ser más apreciables.

Capítulo 7

Implementación del *Simulated Annealing* usando *MapReduce*

Durante este capítulo se describe cómo se ha realizado la implementación del algoritmo *Simulated Annealing* usando el modelo de programación *MapReduce*.

La idea de dicha implementación surge porque, como el propio nombre del trabajo indica, la reconstrucción de las trayectorias se iba a hacer desde el principio usando dicho algoritmo. Tras realizar una serie de pruebas en el dashboard creado, los resultados obtenidos sí mejoraban los de partida. Por todo ello, se decidió implementar el algoritmo *Simulated Annealing* usando el modelo de programación *MapReduce* para poder procesar en paralelo grandes conjuntos de datos procedentes de diferentes vuelos, y así, dar una solución escalable a la vez que distribuida al problema planteado en el proyecto llevado a cabo.

7.1. Descripción del entorno

Para realizar la implementación del algoritmo *Simulated Annealing* usando el modelo de programación *MapReduce* se va a usar como entorno Big Data *Cloudera*.

Cloudera CDH (Cloudera Distribution Hadoop) es la plataforma de código abierto de *Cloudera* siendo esta la distribución más popular de *Apache Hadoop* (base tecnológica sobre la que se desarrolla *Cloudera*). Pero, además de incluir el núcleo de *Hadoop*, integra diversos proyectos de la fundación de Apache como HBase, Mahout, Pig, Hive, etc. *Hadoop* es un framework muy potente y uno de los más usados en el contexto tecnológico actual, gracias a su importancia en el desarrollo de proyectos que usan tecnologías de tipo Big Data ya que ofrece computación fiable, escalable y distribuida, apoyándose para ello en la técnica de *MapReduce* y en el sistema distribuido de archivos HDFS. Por lo tanto, *Hadoop* es capaz de ejecutar programas en *MapReduce* escritos en Java como es nuestro caso. Dichos programas se caracterizan por su naturaleza paralela, lo cual resulta de gran utilidad para analizar grandes conjuntos de datos usando distintas máquinas distribuidas en clusters, fáciles de escalar.

En la Figura 7.1 se muestra la arquitectura básica de *Hadoop*. Por un lado, se muestra el sistema de archivo HDFS que se comunica con la infraestructura de programación *MapReduce*, siendo ésta la encargada de realizar todas las operaciones y procesamientos distribuidos. Por otro lado, Hive, HBase, Pig, etc. son herramientas que usa *MapReduce* para realizar operaciones sobre los datos.

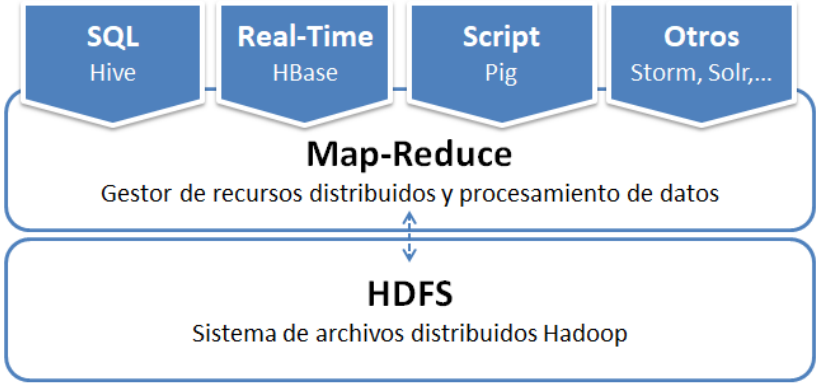


Figura 7.1: Arquitectura básica de *Hadoop* [21]

El procedimiento seguido para la implementación ha sido el siguiente:

1. Instalación de *X2Go Client*: Como la implementación se ha realizado usando la máquina virtual *Cloudera*, se puede: bien acceder a ella vía ssh o bien usando *X2Go Client* que es un software de código abierto que proporciona un escritorio remoto muy visual que facilita el trabajo en *Cloudera*. Su descarga se puede hacer de forma gratuita en la página web `https://wiki.x2go.org/doku.php`.

Una vez instalado, se requiere realizar una serie de configuraciones para poder usarlo, como son las siguientes: indicar el nombre de la sesión, el host, el usuario, el puerto de conexión SSH y el tipo de sesión usada.

2. Uso de *Eclipse Luna* y *NetBeans 8.2* para la implementación: Primero, se comenzó programando el algoritmo en *NetBeans* usando Java y tras ver que su funcionamiento era el esperado, se usó *Eclipse Luna* para añadir la funcionalidad asociada usando el modelo de programación *MapReduce* dentro de la máquina virtual de *Cloudera*.

Por un lado, *NetBeans 8.2 IDE* es un entorno de desarrollo integrado libre, gratuito y multiplataforma que usa el lenguaje de programación Java, y por otro lado, *Eclipse Luna* es una plataforma de software que consta de un conjunto de herramientas de programación, las cuales son de código abierto, multiplataforma y que también usan Java.

Antes de detallar cómo se ha realizado la implementación, conviene explicar las tecnologías de *Apache HDFS* y el modelo de programación *MapReduce* ya que se han usado para su realización.

Apache HDFS (Hadoop Distributed File System): Es un framework de software que soporta aplicaciones distribuidas bajo una licencia libre. Es adecuado para aplicaciones que tienen grandes conjuntos de datos y se encarga del almacenamiento distribuido de dichos datos en un determinado clúster.

De manera interna, un archivo se divide en uno o más bloques, donde todos los bloques tienen el mismo tamaño a excepción del último y se almacenan en un conjunto de *DataNodes* (explicado posteriormente). Estos bloques son replicados para evitar posibles fallos. Tanto el tamaño de los bloques como el número de réplicas es configurable, y por lo general el tamaño suele ser grande, permitiendo reducir el tiempo de acceso asociado a la lectura de los datos. *HDFS* es compatible con el patrón "write once read many", que resulta de gran utilidad para las aplicaciones que escriben datos una sola vez pero los leen una o más veces a determinadas velocidades. Luego, lo que se tiene es un archivo que se almacena mediante un determinado número de bloques que son replicados entre los distintos servidores del clúster de *Hadoop*. En la Figura 7.2 se muestra un ejemplo de una réplica donde se tienen 5 bloques, cada uno de los cuales son replicados dos veces.

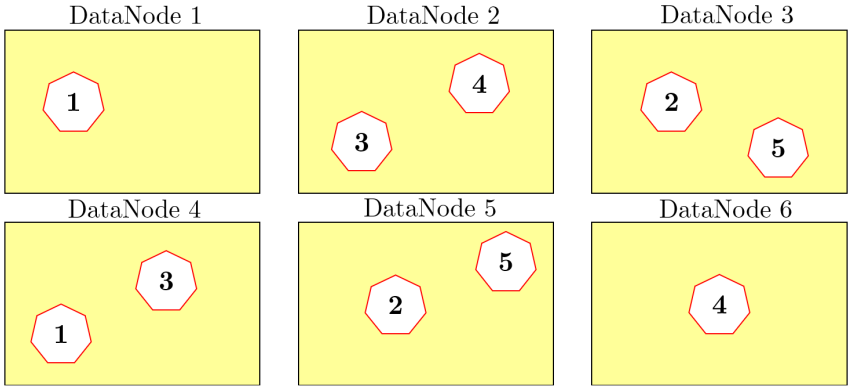


Figura 7.2: Replicamiento de bloques [18]

HDFS tiene una arquitectura de tipo maestro/esclavo, donde el nodo maestro se conoce como *NameNode* y el resto como *DataNodes*. El *NameNode* es un servidor que se encarga de administrar el espacio de nombres del sistema de archivos (realiza operaciones como abrir, cerrar y renombrar tanto archivos como directorios) y regular el acceso de los clientes a los mismos. Además, asigna los bloques a los *DataNodes*, los cuales, se encargan de atender las solicitudes de lectura y escritura de los clientes del sistema de archivos, así como crear, eliminar y replicar los bloques según las órdenes dadas por el *DataNode*. Por lo tanto, son los encargados de ejecutar las funciones *Map* y *Reduce* sobre los datos que se almacenan de manera local en cada uno de los nodos del clúster.

En resumen, mientras que el *NameNode* se encarga de saber el estado de los datos almacenados por el clúster, los *DataNodes* se encargan de realizar dicho almacenamiento físico de los datos en el clúster asociado. Además, existe otro nodo denominado *Jobtracker* que agrupa las tareas de *MapReduce* a nodos específicos en el clúster (suelen ser los que tienen los datos, o al menos los que están en el mismo rack).

En la Figura 7.3 se muestra la arquitectura *HDFS* comentada con anterioridad.

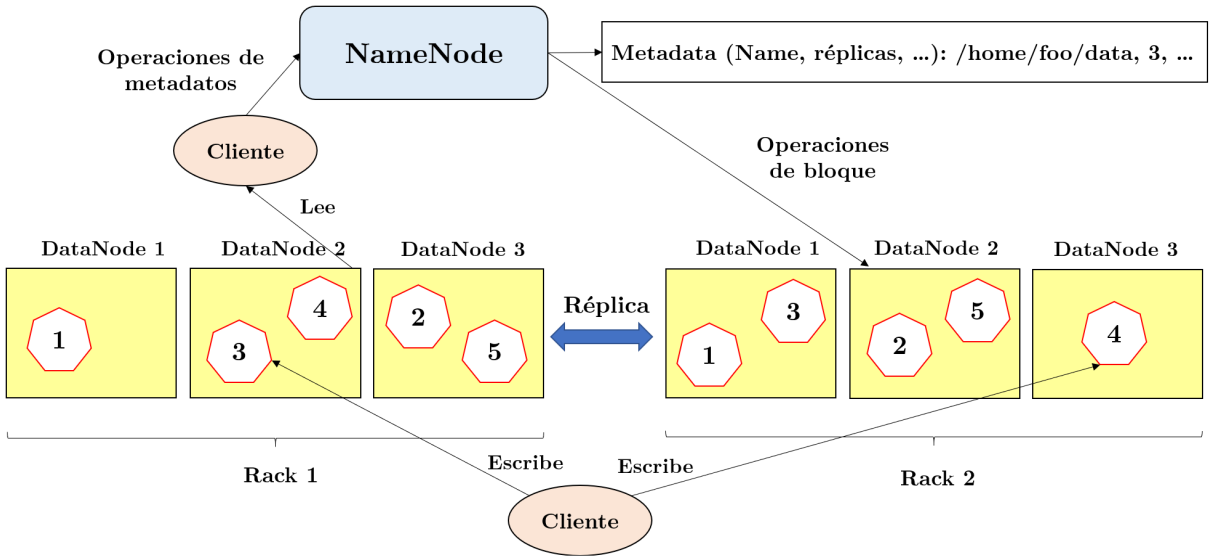


Figura 7.3: Arquitectura HDFS [18]

MapReduce: Es un entorno de desarrollo o modelo de programación diseñado para procesar grandes cantidades de datos en paralelo dentro de sistemas que se encuentran distribuidos. Se encarga de repartir las tareas entre los diversos nodos del clúster. *MapReduce* no sirve para resolver cualquier tipo de problema, siendo principalmente usado con problemas asociados a grandes cantidades de datos que suelen ejecutarse sobre el sistema de ficheros *HDFS*. Entre sus principales características podemos destacar las siguientes:

· Distribución y paralelización automáticas.

· Presenta tolerancia frente a fallos y redundancias.

· Dispone de herramientas de monitorización.

· Con funcionamiento interno y mantenimiento transparente para los desarrolladores.

· Posee gran escalabilidad horizontal, pues en caso de ser necesario más cómputo de programación, basta con añadir más nodos al clúster.

· Permite la localización de los datos mediante el desplazamiento del algoritmo a ellos.

El esquema de funcionamiento de *MapReduce* se puede ver en la Figura 7.4. Como se puede apreciar, existen varias fases. La primera de ellas es la fase del *Mapping* que consta de un método (*map*) que recibe como parámetros de entrada un par (clave, valor) por cada línea del fichero de entrada. Se encarga de realizar el tratamiento sobre cada uno de esos pares (clave, valor) devolviendo cero o más pares (clave, valor) en cada llamada. Tras ello, está la fase intermedia del *Shuffling* & *Sort* en la que se ordenan los resultados obtenidos del *Mapper* por la clave y se combinan en una lista aquellos que tienen la misma clave. Finalmente, en la fase del *Reducer*, se define el método *reduce* que, tras recibir las claves y

los valores asociados a cada una de ellas, realiza las operaciones indicadas emitiendo uno o más pares de (clave, valor).

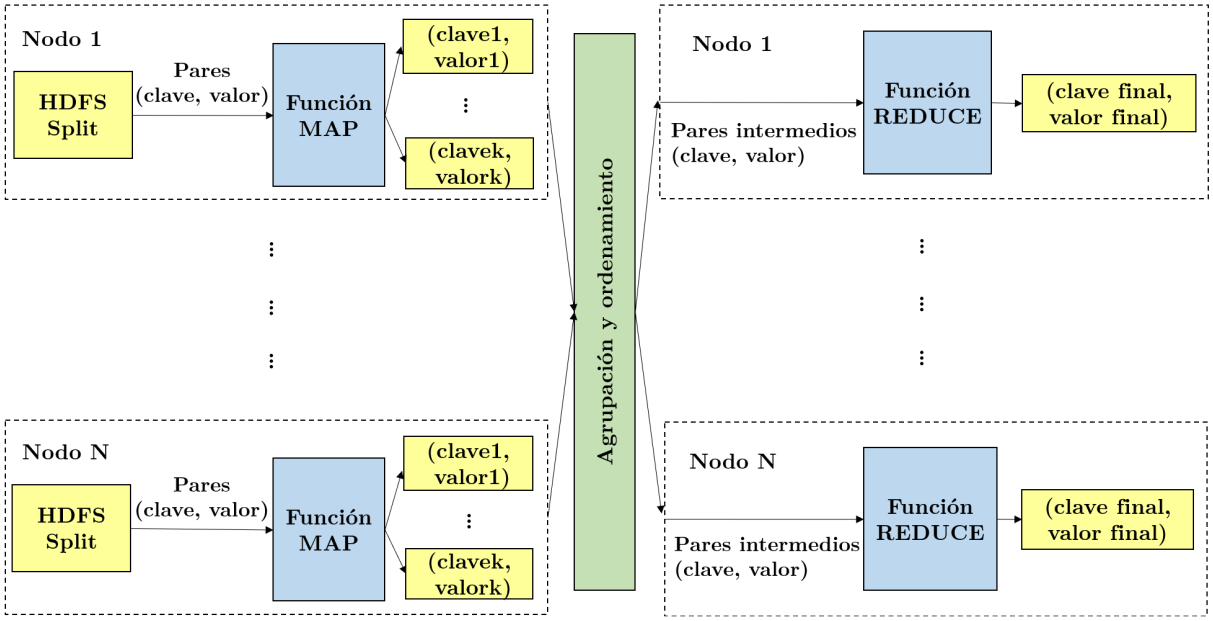


Figura 7.4: Esquema de funcionamiento de *MapReduce* [17]

A continuación, se muestra un ejemplo concreto que consiste en obtener el número de veces que aparece cada una de las palabra de un texto dado como entrada. El texto considerado consta de tres líneas diferentes que serán procesadas en paralelo. El resultado obtenido en cada una de las distintas fases puede verse en la Figura 7.5.

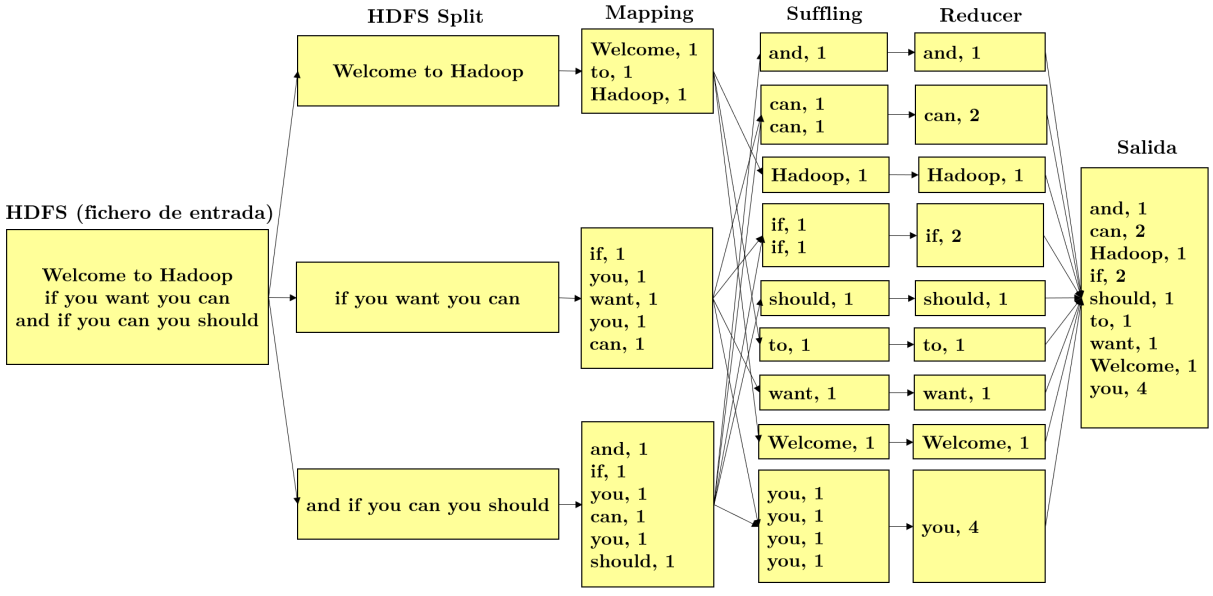


Figura 7.5: Ejemplo WordCount usando *MapReduce*

7.2. Implementación del algoritmo

Durante esta sección se pretende dar una breve explicación de la implementación realizada del algoritmo *Simulated Annealing* usando el modelo de programación *MapReduce* comentado con anterioridad y el lenguaje Java.

En el caso concreto de la implementación del *Simulated Annealing*, el proceso seguido en la primera fase (la fase del *Mapper*) consiste en lo siguiente: dado un archivo con un conjunto de líneas (cada una de ella está asociada a la información de un cierto mensaje ADS-B), se encarga de procesarlas y agruparlas según el identificador o leg del vuelo. Luego, en la fase del *Reducer*, una vez obtenidos los datos de cada vuelo, se aplica dicho algoritmo de resolución a cada conjunto de datos obtenido. Tras ello, la salida resultante es uno o más ficheros que contienen los mensajes ordenados tras ejecutar el algoritmo para los distintos vuelos considerados, junto con cuatro columnas adicionales: *distancia*, *time_nuevo*, *tag_orig* y *tag_ord* que se detallarán a lo largo de las explicaciones posteriores.

Para realizar la ejecución del proyecto es necesario introducir una serie de parámetros de entrada de la forma: *fichero_entrada fichero_salida num_iteraciones temperatura ratio_enfriamiento tam_entorno modo tam_vuelo solap_vuelo tam_aerop solap_aerop altura*, donde los 7 primeros son obligatorios y los 5 últimos dependen del modo de ejecución elegido.

El proyecto creado consta de 5 clases que son: *Main.java*, *Algoritmo.java*, *Interpreter.java*, *Mensaje.java* y *SimulatedAnnealing.java*. A continuación se describe cada una de ellas.

Main.java: Es la clase principal del proyecto y en ella se implementan los métodos *map* y *reduce*. Por un lado, en el método *map* se lee cada una de las líneas del fichero de entrada y se toquenizan considerando como separador la coma (,). Tras ello, para cada línea se crea una instancia de la clase mensaje con la línea de la cabecera y la línea de datos leída. Finalmente, se envían al collector como datos de salida del mapper diversos pares (clave, valor) donde la clave es el identificador o leg del vuelo y el valor cada instancia de mensaje creada. Por otro lado, en el método *reduce()*, primero se recogen los mensajes correspondientes a un determinado valor de leg y se instancia un objeto de la clase mensaje con cada uno de ellos. Todos esos mensajes se guardan en un ArrayList de mensajes con el que se invoca al constructor de la clase *Interpreter.java*, y tras ello, al método *reordenarMensajes()* que se encarga de ordenarlos por *timestamp*. Después, se aplica el método *Simulated Annealing* a dichos mensajes distinguiendo los tres modos de ejecución (sin ventanas, con ventanas del mismo tamaño, o con ventanas de distinto tamaño en la zona del aeropuerto y el vuelo). Una vez que se tienen los mensajes reordenados, el método *getArrayDistance()* devuelve un vector de distancias cuyas componentes vienen dadas por la distancia existente entre cada punto con respecto al primero. Esto se usa en el método *corregirTimestamps()* que recibe la ruta ordenada y el vector de distancias para realizar la corrección de los tiempos de aquellos puntos que han sufrido cambios tras la reordenación. Finalmente, se añade al fichero de salida la cabecera dada más los campos *distancia*, *time_nuevo*, *tag_orig* y *tag_ord* que se corresponden respectivamente con la distancia entre cada punto con respecto al primero tras la reordenación, el nuevo tiempo

interpolado, la posición original y la obtenida al aplicar el algoritmo.

En el método estático *main* se crea el trabajo o *job* y se consideran las distintas opciones de parámetros de entrada. Tras ello, se define el formato de los ficheros, se llama al método *map* y *reduce*, etc. y se ejecuta el *job*.

En la Figura 7.6 se muestra el esquema de funcionamiento descrito con anterioridad. Como se ve, tras partir de un fichero con mensajes ADS-B no ordenados de distintos vuelos, se obtiene un fichero de mensajes ADS-B ordenados para cada uno de los vuelos considerados (se ordenan según el valor de leg que es el identificador del vuelo).

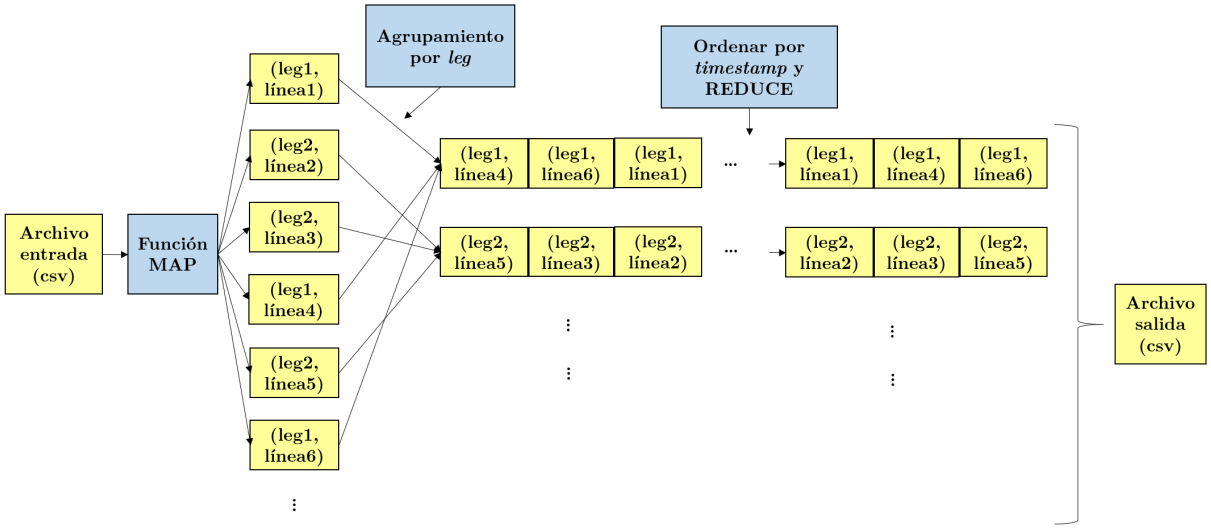


Figura 7.6: Lógica de funcionamiento

Algoritmo.java: Se trata de una clase abstracta que representa un determinado algoritmo de reordenación. En este caso sólo tenemos uno y no es estrictamente necesaria, pero se ha considerado así porque es más general y permite añadir otros algoritmos de forma rápida y eficaz. En dicha clase se definen un conjunto de variables, unos constructores (para el caso sin ventanas o ventanas del mismo tamaño, pues basta considerar para el caso sin ventanas un tamaño de ventana igual al número de datos y solapamiento cero; y para el caso de ventanas con distinto tamaño en la zona del aeropuerto y en el vuelo), así como una serie de métodos: *get* y *set*, *calcularParametros()* que calcula el número de ventanas a considerar así como el límite inferior, el límite superior, el tamaño de ventana y el solapamiento para cada una de las tres zonas posibles a considerar, *initDistanceTable()* que calcula la matriz de distancias entre cada par de puntos, *createInitialTour()* para crear la ruta inicial, *getDistance()* que calcula la distancia de la ruta actual y si se le pasa como parámetro una subruta, calcula la distancia de la misma, *getArrayDistance()* devuelve un array de distancias cuyas coordenadas vienen dadas por la distancia existente entre cada punto con respecto al primero y *runAlgorithm* para la ejecución del algoritmo.

Interpreter.java: Dicha clase posee una serie de variables y un constructor que recibe la posición de la columna asociada al campo longitud, latitud, altura, velocidad y *timestamp* en el fichero, así como un ArrayList de mensajes. Tras ello, hay tres métodos *getCoordinates()*, *getIds()* y *getVelocidades()*, que se encargan de obtener las coordenadas (longitud, latitud y altitud), los *timestamp* y las velocidades de cada punto respectivamente. Luego hay otros métodos como: *reordenarMensajes()* que se encarga de ordenar todos los mensajes del ArrayList por *timestamp*, *estaEnArray()* que comprueba si un booleano está en un array de booleanos, *buscarPuntoBueno()* que se encarga de obtener la posición del punto bueno anterior o posterior al punto que va a ser interpolado, *mean()* que calcula la media aritmética de un ArrayList dado el índice de inicio y de fin excluyendo los valores nulos y el método *buscarCoincidencias()* que recibe como parámetro de entrada un array con el orden obtenido al aplicar el algoritmo y devuelve un array donde el valor true significa que el dato no ha sido alterado y el valor false que si lo ha sido. Por último, el método *corregirTimestamps()* se encarga de corregir los tiempos para los puntos alterados tras la reordenación siguiendo el mismo modelo de programación lineal descrito en el Capítulo 5 usando el entorno de *R-Studio*.

Mensaje.java: Dicha clase es bastante sencilla. Tiene un constructor que recibe el nombre de los campos de la cabecera de un archivo y una línea de datos. Luego se definen una serie de métodos *get* y *set*, el método *getCoordenadas()* que devuelve el valor de las coordenadas de un punto en términos de longitud, latitud y altitud dado el índice de posición y el método *toString()* que va construyendo las líneas de datos concatenando los distintos valores y usando como separador la coma.

SimulatedAnnealing.java: Es la clase donde se implementa el algoritmo *Simulated Annealing*. Para su ejecución se necesita conocer cuatro parámetros que son proporcionados por el usuario: temperatura inicial, ratio de enfriamiento, número de iteraciones y tamaño del entorno de estados vecinos considerado para los distintos estados. Dicha clase, consta de dos constructores que se usarán en función del modo de ejecución considerado, así como de varios métodos.

- *getDistance()*: Obtiene la distancia de la ruta actual en kilómetros.
- *criterioAcep()*: Se corresponde con el criterio de aceptación de estados definido en (4.1).
- *randomDouble()*: Devuelve un valor aleatorio en [0,1).
- *randomInt()*: Devuelve un número entero aleatorio en el rango [min, max).
- *runAlgorithm()*: Implementa el algoritmo *Simulated Annealing* siguiendo el esquema dado en el Capítulo 4 y distinguiendo en función de los distintos modos de ejecución posibles. Para ello, primero se calcula el valor de los parámetros invocando al método *calcularParametros()* y, para cada una de las ventanas, se ejecuta el algoritmo teniendo en cuenta cuál es el límite inferior y superior de cada una de ellas. Tras ello, mientras la temperatura sea menor que 1 se genera un estado aleatorio en el espacio de estados S y, para el número de iteraciones fijado, se genera otro estado aleatorio en un entorno próximo al estado anterior (dicho entorno es fijado por el usuario). Después, se comprueba si el estado es aceptado o no (se tiene en cuenta el criterio de aceptación y que la nueva

distancia sea igual o mejor que la anterior). En caso de ser aceptado se intercambian dichos estados, y en otro caso, no se modifica su posición. Una vez que acaban todas las iteraciones para un estado se desciende el valor de la temperatura teniendo en cuenta el ratio de enfriamiento. Esto se repite hasta que la temperatura toma un valor menor o igual a 1.

En la Figura 7.7 se muestra una imagen de una parte del fichero resultante tras ejecutar el proyecto creado en Java usando *MapReduce*. En ella, se observan los distintos mensajes ordenados asociados al primero de los vuelos considerados en el fichero de entrada.

```
id,time,latitude,longitude,altitude,speed,vspeed,squawk,track,ground,leg,date,distancia,time_nuevo,tag_orig,tag_ord
fr24-34250C-1517590708,1517590708,40.48718,-3.59243,0,0,0,0000,13,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.0,1517590925,2,1
fr24-34250C-1517590744,1517590744,40.48716,-3.59244,0,0,0,0000,154,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.0,1517590926,7,2
fr24-34250C-1517590794,1517590794,40.48716,-3.59243,0,0,0,0000,335,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.0,1517590926,8,3
fr24-34250C-1517590723,1517590723,40.48716,-3.59243,0,0,0,0000,335,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.0,1517590926,5,4
fr24-34250C-1517590809,1517590809,40.48716,-3.59244,0,0,0,0000,335,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.0,1517590926,9,5
fr24-34250C-1517590697,1517590697,40.48716,-3.59244,0,0,0,0000,304,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.0,1517590926,1,6
fr24-34250C-1517590733,1517590733,40.48717,-3.59243,0,0,0,0000,154,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.0,1517590926,6,7
fr24-34250C-1517590712,1517590712,40.48716,-3.59243,0,0,0,0000,194,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.0,1517590926,3,8
fr24-34250C-1517590717,1517590717,40.48717,-3.59243,0,0,0,0000,154,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.0,1517590926,4,9
fr24-34250C-1517590927,1517590927,40.48552,-3.59122,0,22,0,0000,87,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.21,1517590927,10,10
fr24-34250C-1517590941,1517590941,40.48552,-3.59027,0,25,0,0000,90,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.29,1517590941,11,11
fr24-34250C-1517590947,1517590947,40.48551,-3.59016,0,25,0,0000,90,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.3,1517590947,12,12
fr24-34250C-1517590951,1517590951,40.48552,-3.58990,0,25,0,0000,90,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.32,1517590951,13,13
fr24-34250C-1517590962,1517590962,40.48552,-3.58919,0,27,0,0000,90,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.38,1517590962,14,14
fr24-34250C-1517590967,1517590967,40.48554,-3.58869,0,27,0,0000,87,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.43,1517590967,15,15
fr24-34250C-1517590987,1517590987,40.48635,-3.58818,0,25,0,0000,24,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.53,1517590987,16,16
fr24-34250C-1517590992,1517590992,40.48656,-3.58819,0,25,0,0000,24,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.55,1517590992,17,17
fr24-34250C-1517590997,1517590997,40.48682,-3.58819,0,25,0,0000,2,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.58,1517590997,18,18
fr24-34250C-1517591012,1517591012,40.48754,-3.58821,0,26,0,0000,2,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.66,1517591012,19,19
fr24-34250C-1517591023,1517591023,40.48794,-3.58820,0,25,0,0000,2,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.7,1517591023,20,20
fr24-34250C-1517591038,1517591038,40.48877,-3.58821,0,27,0,0000,2,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.8,1517591038,21,21
fr24-34250C-1517591044,1517591044,40.48906,-3.58821,0,27,0,0000,359,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.83,1517591044,22,22
fr24-34250C-1517591054,1517591054,40.48967,-3.58821,0,29,0,0000,359,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,0.9,1517591054,23,23
fr24-34250C-1517591115,1517591115,40.49245,-3.58661,0,16,0,0000,90,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,1.23,1517591115,24,24
fr24-34250C-1517591120,1517591120,40.49244,-3.58606,0,17,0,0000,90,True,IBE04HT_34250C_1517590697_1517594767,2018-02-02,1.28,1517591120,25,25
```

Figura 7.7: Fichero de salida resultante

Capítulo 8

Conclusiones y trabajo futuro

En este capítulo del trabajo se busca describir cuáles han sido las conclusiones obtenidas tras su realización así como los objetivos conseguidos y el conocimiento adquirido. Además, se presentarán un conjunto de líneas de trabajo futuro y posibles mejoras del proyecto de investigación desarrollado.

8.1. Conclusiones

Una vez realizado el desarrollo del presente trabajo, se puede concluir que han sido alcanzados con éxito los objetivos planteados al comienzo del mismo. Así, gracias al estudio y desarrollo de los distintos algoritmos presentados a lo largo del mismo, en particular centrándome en el *Simulated Annealing*, se ha podido obtener una solución eficiente y escalable que permite corregir aquellas trayectorias que presentan alguna anomalía (asociada principalmente a una mala asignación de los *timestamps*) y realizar la correspondiente reasignación de *timestamps* para los puntos que han sido alterados durante la realización de la reordenación. Por lo tanto, se ha podido observar como los fundamentos teóricos de dicho algoritmo y las conclusiones del mismo que fueron descritas en el Capítulo 4 eran certeras, ya que ha sido posible su visualización práctica.

Para ello, ha sido de gran utilidad el dashboard implementado usando el entorno de programación *R-Studio*. Dicho dashboard ha sido el pilar fundamental para la realización del presente trabajo, ya que ha sido usado para poder ver el funcionamiento de los diferentes algoritmos y realizar la comparativa mostrada en el Capítulo 6. Gracias a ello, se ha podido ilustrar cómo al aplicar alguno de dichos algoritmos a los distintos vuelos proporcionados, se logra obtener trayectorias cuyas distancias reducen de manera considerable la de la trayectoria de partida.

Todo esto, permite concluir que el estudio realizado puede ser llevado a cabo para su aplicación en los sistemas actuales de gestión del tráfico aéreo. Para el caso concreto del *Simulated Annealing*, se ha realizado un proyecto en Java usando el modelo de programación *MapReduce*, que permite obtener las rutas ordenadas de un conjunto dado de vuelos de manera eficiente y escalable, así como llevar a cabo un modelo de interpolación lineal para realizar la reasignación de tiempos. El propósito de su realización es que pueda ser

incorporado al tratamiento de los datos que se hace en las plataformas Big Data que tratan con las trayectorias y que permita mejorar y avanzar en lo relativo a la gestión del tráfico aéreo.

Tras finalizar dicho trabajo, puedo decir que ha sido de gran utilidad para mí, tanto por darme la posibilidad de poner en práctica ciertos conocimientos adquiridos durante estos 5 años de carrera, como por darme la oportunidad de aprender cosas que no había visto hasta ahora y que seguramente usaré en el futuro.

Entre los conocimientos que he puesto en práctica se podrían destacar los siguientes:

Utilización y Administración de Sistemas Operativos: Se ha usado principalmente para las instalaciones de los distintos programas y entornos usados.

Fundamentos de las Tecnologías de la Información, Tecnologías Web y Grado de Matemáticas: Ha sido de utilidad los conocimientos principalmente estadísticos adquiridos en la carrera de Matemáticas para realizar el dashboard en *R*, analizar los datos y resultados obtenidos y entender en mayor profundidad el significado de los mismos. También ha sido útil el conocimiento adquirido en Fundamentos de las Tecnologías de la Información y Tecnologías Web para la aplicación de estilos en el dashboard.

Metodología de la Programación, Programación Orientada a Objetos, Programación y Estructura de Datos: Se ha usado para realizar la implementación del método *Simulated Annealing* en Java usando el modelo de programación *MapReduce*.

Proceso de Desarrollo del Software, Modelado de Software, Gestión de Proyectos Basados en las TI y Plataformas Software Empresariales: Para realizar el presente documento sobre el proyecto de investigación realizado.

Además he adquirido nuevos conocimientos como los referentes al estudio e investigación del entorno aeronáutico, al desarrollo del dashboard en *Shiny* usando *R-Studio*, y en particular, al uso del modelo de programación *MapReduce*, el cual, ha sido de gran importancia en la realización del trabajo ya que ha permitido dar una solución escalable al problema planteado. Por lo tanto, con todo ello se ha conseguido lograr los objetivos o metas fijadas al comienzo de su realización y poder concluir con éxito el trabajo realizado.

Además, estas tecnologías están en pleno auge en el campo de la Informática, ya que en todos los aspectos del mundo actual que nos rodea es necesario tratar con grandes cantidades de datos. Por ello, el desarrollo de este trabajo de investigación me ha permitido adquirir un mayor conocimiento sobre este tipo de tecnologías, ampliando así mi aprendizaje de cara al futuro profesional, lo cual, es de gran utilidad.

8.2. Líneas de trabajo futuro

Como ya he comentado el trabajo llevado a cabo es un proyecto de investigación que se enmarca dentro de la colaboración que se realiza con el proyecto de "AIRPORTS" (CIEN, 2015)" liderado por Boeing Research and Technology Europe.

Su principal propósito es estudiar la eficiencia de los vuelos de aeronaves comerciales en función de la trayectoria que estos describen al realizar dicho vuelo. Como ya se ha podido observar, gracias al desarrollo del presente trabajo, se ha logrado mejorar las trayectorias dadas aplicando diferentes algoritmos de resolución e implementando una solución escalable para ello. Además, usando un modelo de interpolación lineal, se ha realizado la reasignación de los nuevos *timestamps* para aquellos puntos que han sufrido un cambio de orden tras la reordenación.

Sin embargo, aunque el trabajo realizado es un buen comienzo, no es más que un principio de todo lo que queda por hacer, pues es necesario seguir estudiando, analizando e investigando para lograr una gestión del tráfico aéreo completa y eficiente.

Algunas de las posibles mejoras de dicho proyecto podrían ser las siguientes:

- **Conexión con una base de datos**: Para el dashboard realizado en *R-Studio*, en lugar de tener todos los vuelos guardados en una carpeta dentro del mismo directorio que el script creado, una posible mejora sería tener una base de datos donde tanto Juan Manuel Velasco Heras como yo pudiéramos acceder para la lectura de los ficheros *csv* con los datos de los vuelos. Así, de una manera más dinámica, rápida y eficiente podríamos disponer de los datos asociados a los distintos vuelos.
- **Integración del programa creado en *MapReduce***: Además de realizar la implementación del algoritmo *Simulated Annealing* usando el modelo de programación *MapReduce* para el procesamiento escalable y en paralelo de un conjunto grande de datos de diferentes vuelos, se podría haber realizado su integración en la plataforma de Boeing para así poder poner en práctica su funcionamiento.

Dado que el tiempo para realizar el trabajo es limitado no ha sido posible incluir dichas mejoras, pero en el caso de disponer de más tiempo, habrían sido las siguientes tareas a realizar.

Bibliografía

[1] William J. Cook, *In Pursuit of the Traveling Salesman*, Princeton University Press, 1957.

[2] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan y D.B. Shmoys, *The Traveling Salesman*, Princeton University y AT & T Bell Laboratories, New Jersey 1985.

[3] P.J.M. van Laarhoven y E.H.L. Aarts, *Simulated Annealing: Theory and Applications*, Springer Science+Business Media Dordrecht, 1987.

[4] Emile Aarts y Jan Korst, *Simulated Annealing and Boltzmann Machines*, John Wiley & Sons Lid, 1989.

[5] Wikipedia, *Teoría de grafos*, URL: https://es.wikipedia.org/wiki/Teoria_de_grafos, 2019. Visitado por última vez el 18 de Enero de 2019.

[6] Alfredo Caicedo, Graciela Wagner de García y Rosa María Méndez, *Introducción a la teoría de grafos*, Ediciones Elizcom, 2010.

[7] María Luisa Pérez Delgado, *Los mapas de rasgos autoorganizativos y el problema del viajante de comercio*, Ediciones Universidad de Salamanca, 2003.

[8] Automatic Dependant Surveillance, *Automatic Dependant Surveillance*, URL: https://www.skybrary.aero/index.php/Automatic_Dependent_Surveillance_(ADS), 2017. Visitado por última vez el 18 de Enero de 2019.

[9] SESAR, *SESAR*, URL: https://ec.europa.eu/transport/modes/air/sesar, 2019. Visitado por última vez el 18 de Enero de 2019.

[10] Martínez Prieto, Miguel A., Bregón Bregón, A., García Miranda, I., Álvarez Esteban, Pedro C., Díaz, F. y Scarlatti, D. (2017), *Integrating Flight-related Information into a (Big) Data Lake*, *IEEE*.

[11] Martínez Prieto, Miguel A., Bregón Bregón, A., García Miranda, I., Álvarez Esteban, Pedro C. y Díaz, F. (2017), *Towards a Scalable Architecture for Flight Data Management*, *SCITEPRESS (Science and Technology Publications, Lda.)*, 263-268. DOI: 10.5220/0006473402630268.

[12] Alonso Isla, A., Martínez Prieto, Miguel A., Bregón Bregón, A., García Miranda, I., Álvarez Esteban, Pedro C., Díaz, F. y Gordaliza, P. (2017), *Airports: Análisis de Eficiencia Operacional basado en Trayectorias de Vuelo*, URL: https://biblioteca.sistedes.es/submissions/descargas/2018/JISBD/2018-JISBD-063.pdf.

[13] Gestión del tráfico aéreo (ATM), *Gestión del tráfico aéreo (ATM)*, URL: https://greatbustardsflight.blogspot.com/2018/06/gestion-del-trafico-aereo-atm-de-forma.html, 2018. Visitado por última vez el 8 de Febrero de 2019.

[14] Wikipedia, *OpenSky Network*, URL: https://en.wikipedia.org/wiki/OpenSky_Network, 2018. Visitado por última vez el 16 de Febrero de 2019.

[15] Lenguaje R, *Lenguaje R*, URL: https://blog.datatons.com/2016/04/08/que-es-lenguaje-programacion-r/, 2018. Visitado por última vez el 22 de Febrero de 2019.

[16] Introducción a Shiny, *Introducción a Shiny*, URL: http://rstudio-pubs-static.s3.amazonaws.com/21373_8af3d3634b97461089c8a76659982915.html, 2014. Visitado por última vez el 25 de Febrero de 2019.

[17] MapReduce, *MapReduce*, URL: http://hadoopontheroad.blogspot.com/2013/02/mapreduce.html, 2013. Visitado por última vez el 19 de Abril de 2019.

[18] Hadoop HDFS, *Hadoop HDFS*, URL: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Introduction, 2018. Visitado por última vez el 19 de Abril de 2019.

[19] Kai Lai Chung, *Markov Chains*, Springer, NewYork, 1960.

[20] Isaacson, D. and R. Madsen [1976], *Markov Chains*, Wiley, New York.

[21] Arquitectura de Hadoop, *Arquitectura de Hadoop*, URL: http://www.diegocalvo.es/hadoop/, 2016. Visitado por última vez el 1 de Junio de 2019.

Apéndice A

Contenido del CD-ROM

En esta parte se detalla el contenido del CD-ROM aportado junto con el manual de instalación de cada uno de los programas implementados.

El CD-ROM adjunto en la entrega contiene el código en Java del proyecto realizado usando el modelo de programación *MapReduce*, el código fuente del dashboard realizado usando *R-Studio*, así como una copia en pdf de esta memoria. Para que sea más claro, se detalla la estructura de directorios del CD-ROM entregado y el manual de instalación cuando sea necesario:

- **MapReduceTrayectoriasSA** → Dicha carpeta contiene el proyecto realizado en la máquina virtual de *Cloudera* para implementar el algoritmo de resolución *Simulated Annealing* usando el modelo de programación *MapReduce* y el lenguaje Java. Todas las clases *.java* se encuentran dentro de la carpeta src. En la Figura A.1 se muestra la estructura de directorios del proyecto escalable creado en Java.

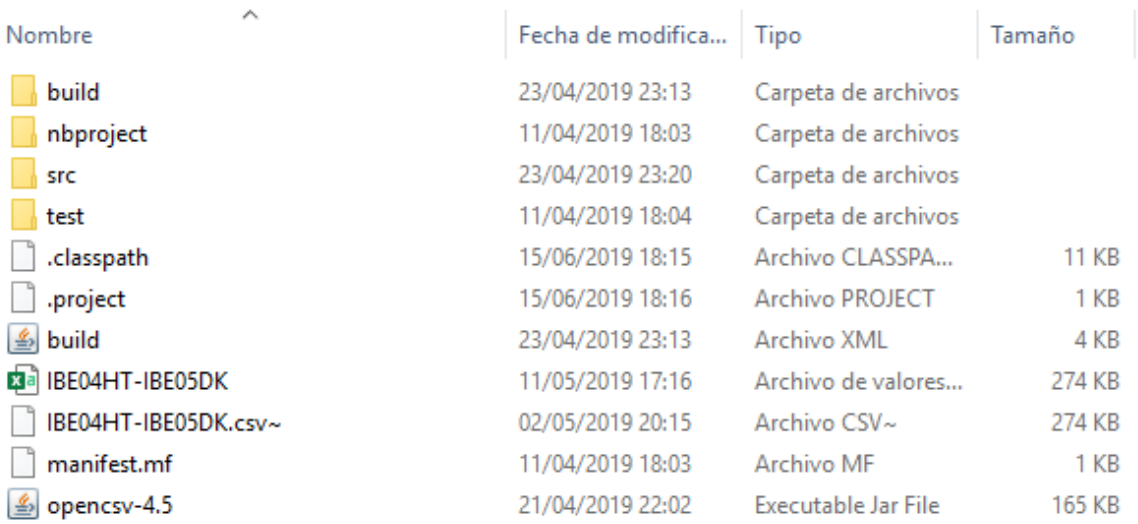


Figura A.1: Estructura de directorios de la aplicación en Java

Para poder ejecutar la implementación escalable realizada se deben cumplir los siguientes requisitos:

1. Disponer de un sistema operativo Cloudera, que integra las librerías de Apache Hadoop.
2. Java JDK 1.7 o superior.
3. IDE Eclipse Luna.

Para descargar Cloudera se puede hacer desde la siguiente página web `https://es.cloudera.com/`. Dicha página web se visualiza en la Figura A.2.

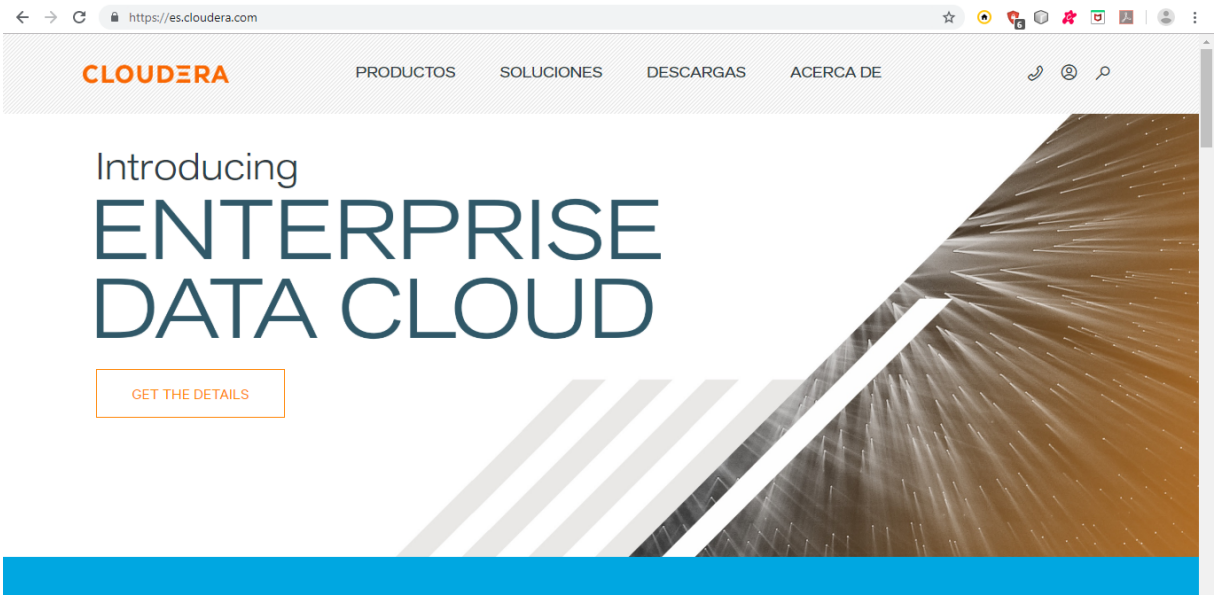


Figura A.2: Página web de Cloudera

- **ProyectoR** → Dicha carpeta contiene una carpeta denominada Shiny que tiene: el código fuente del dashboard realizado en *R-Studio* usando el lenguaje R (*app.R*), una carpeta llamada www que tiene las imágenes e iconos usados en el dashboard, dos ejecutables para los métodos de ejecución *Lin-Kernighan* y *Concorde* (*linkern.exe* y *concorde.exe*), una carpeta llamada Vuelos que tiene un conjunto de archivos en formato *csv* donde cada uno de ellos tiene el nombre de un determinado vuelo y otro ficheros requeridos. Luego, dentro de la carpeta ProyectoR, hay un fichero denominado *runShinyApp.R* que contiene el código necesario para que el dashboard creado se ejecute directamente desde el CD-ROM sin necesidad de tener que abrir el script creado en *R-Studio*, así como otros ficheros adicionales.

En las Figuras A.3 y A.4 se muestra la estructura de directorios comentada con anterioridad.

Nombre	Fecha de modifica...	Tipo	Tamaño
R-Portable	28/06/2019 10:01	Carpeta de archivos	
shiny	02/07/2019 13:31	Carpeta de archivos	
	28/06/2019 11:31	R Workspace	30 KB
.Rhistory	28/06/2019 16:32	Archivo RHISTORY	1 KB
leeme	28/06/2019 11:51	Documento de tex...	1 KB
run	28/06/2019 9:52	Archivo por lotes ...	1 KB
runShinyApp	28/06/2019 12:52	Archivo R	1 KB
ShinyApp	02/07/2019 13:34	Documento de tex...	9 KB

Figura A.3: Estructura de directorios del dashboard (ProyectoR)

Nombre	Fecha de modifica...	Tipo	Tamaño
source	28/06/2019 10:01	Carpeta de archivos	
Vuelos	02/07/2019 13:30	Carpeta de archivos	
www	28/06/2019 10:01	Carpeta de archivos	
	30/06/2019 16:46	R Workspace	115 KB
.Rhistory	30/06/2019 16:46	Archivo RHISTORY	13 KB
app	30/06/2019 16:32	Archivo R	67 KB
concorde	23/12/2003 3:10	Aplicación	2.550 KB
cygintl-2.dll	03/04/2009 7:04	Extensión de la apl...	31 KB
cygwin1.dll	29/03/2011 10:11	Extensión de la apl...	2.605 KB
linkern	23/12/2003 3:06	Aplicación	653 KB

Figura A.4: Estructura de directorios del dashboard (Shiny)

Por otro lado, el proceso de instalación para ejecutar el dashboard es el siguiente:

1. Descomprimir el zip *ProyectoR.zip*.
2. Abrir la carpeta *ProyectoR* y ejecutar el archivo *run.bat*.
3. Tras ello, se abre un terminal y hay que esperar unos segundos hasta que se abre el dashboard en el navegador que se tenga como predeterminado.
4. Una vez que se acaba de trabajar con la aplicación basta con cerrar la pestaña del navegador, lo cual, cierra la sesión del terminal abierto.

Dicho procedimiento también se encuentra detallado en el fichero *leeme.txt* que se encuentra dentro de la carpeta ProyectoR.

- **Memoria** → Dicha carpeta incluye en formato pdf una copia del presente documento denominado *TFG_MaríaZorita.pdf*.