ARTÍCULO DE INVESTIGACIÓN

TÍTULO

Algorithm creation and optimization for the Crew Pairing Problem

AUTOR

Jesús Manuel Puentes Gutiérrez.

OTROS

Ángel A. Juan, Alexandre Viejo Galicia y Antonio Rodil Garrido Respectivamente autor, director, consultor y tutor de la Universitat Oberta de Catalunya.

ABSTRACT

En las líneas aéreas, el ahorro de costes es primordial para competir con otras compañías e incluso poder subsistir en un mercado tan agresivo como es el del sector aéreo. El factor más influyente, después del coste de combustible, es el del coste debido a las tripulaciones. Por ello tratamos de conseguir, por un lado, el ahorro de costes, y por otro, la mejora de las condiciones laborales de las tripulaciones. En concreto se trata de conseguir tramos de vuelo eficientes cuya obtención requiera el mínimo de tiempo de ejecución posible, siempre cumpliendo todas las restricciones impuestas por las compañías o convenios, y también asentar las bases para, posteriormente, seguir mejorando y optimizado las programaciones de las tripulaciones, como son la asignación de tripulantes a vuelos o la optimización del sistema para reducir retrasos o evitar cancelaciones.

KEYWORDS

Crew Pairing Problem, algoritmo de optimización, Crew Scheduling, ahorro de costes, tramos de vuelos, leg, pairing.

INTRODUCCIÓN

Las compañías aéreas buscan optimizar el funcionamiento de las programaciones de las tripulaciones (The Crew Scheduling Problem) para reducir los costes, sobre todo los asociados al personal como pueden ser salarios, dietas y alojamientos y así aumentar sus beneficios. Éstos costes suponen el segundo mayor gasto de las empresas después del combustible, tal y como podemos observar en la siguiente gráfica:

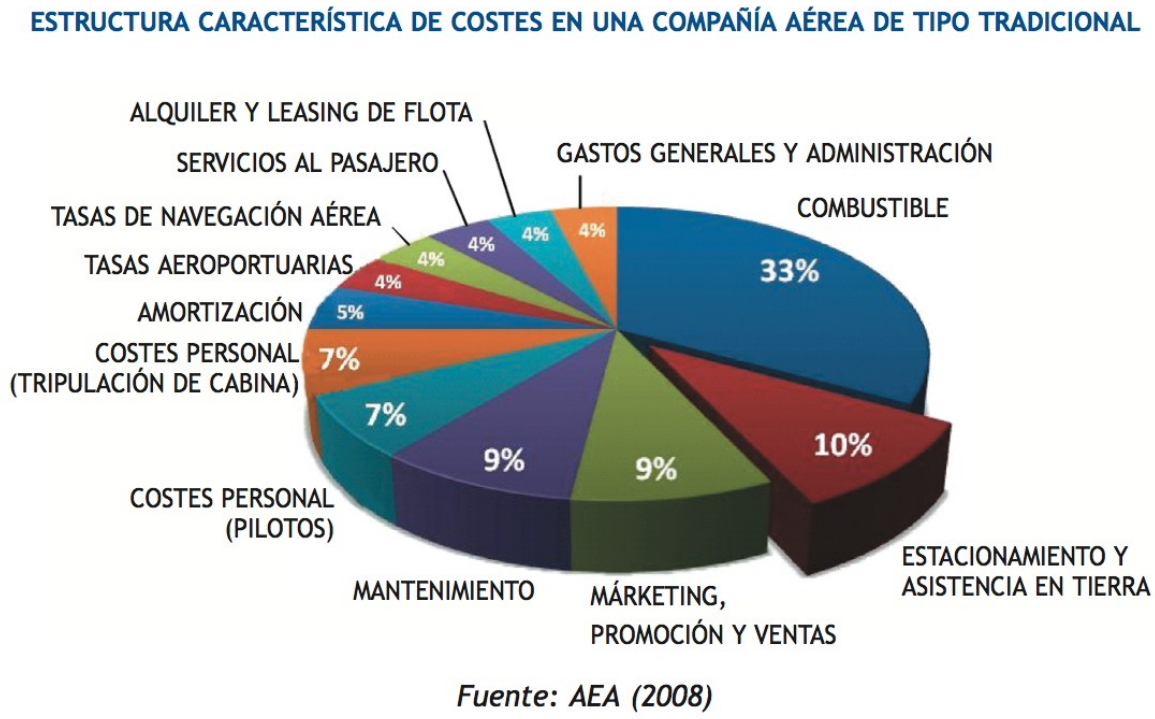

Ilustración 1: Principales gastos de una compañía aérea según el Ministerio de Fomento de España

Como podemos observar, los gastos de personal (pilotos y tripulación de cabina) suponen el 14 % de gastos en una compañía aérea, que son el segundo gasto más importante tras el gasto de combustible (33 %). Estos datos son facilitados por el Ministerio de Fomento de España y pueden ser confirmados en http://www.fomento.gob.es/NR/rdonlyres/77B84967-D3E5-4EB9-87B8-5CC1E5855641/71842/ANALISISCOMPARATIVO1.pdf
Por todo ello, un ahorro en los costes que suponen las programaciones de las tripulaciones, pueden suponer un cuantioso ahorro, suponiendo incluso la diferencia para la compañía entre sobrevivir o no.

Por consiguiente, debido a la importancia que tiene encontrar soluciones para este problema, esta investigación lo abordará dividiéndolo en dos partes: el problema de la asignación de líneas y el problema de construir horarios para las tripulaciones, estudiando en profundidad el primero de ellos, pues ambos se complementan y sin resolver la primera parte no se puede solucionar la segunda.

DESCRIPCIÓN DETALLADA DEL PROBLEMA

En el siguiente gráfico podemos observar el proceso de elaboración del problema de las programaciones de las tripulaciones:

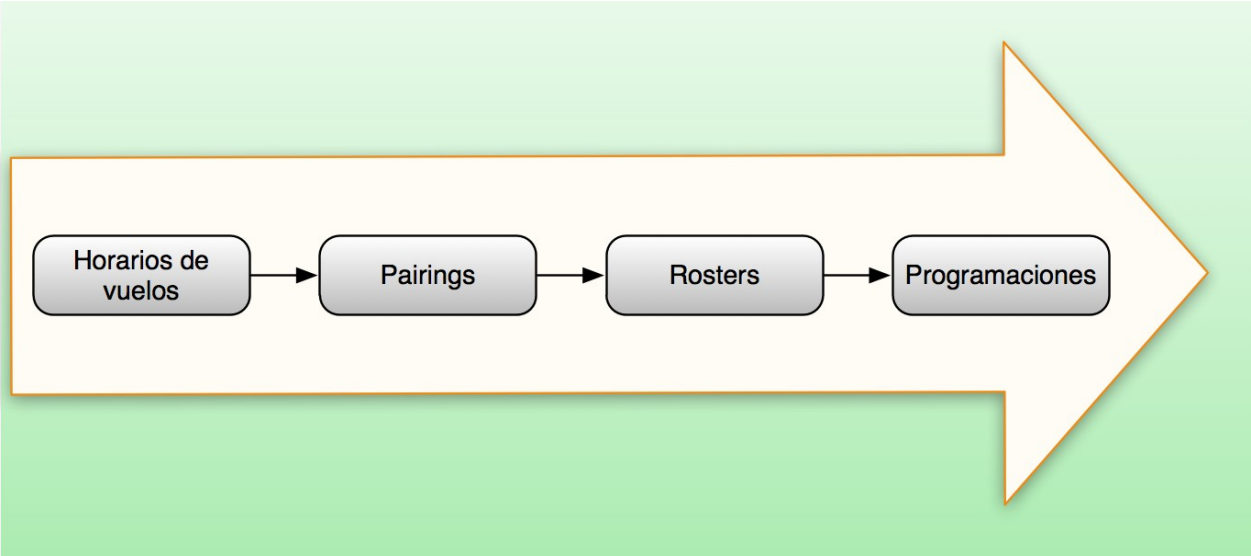

Ilustración 2: Detalle del proceso de elaboración de la programación de las tripulaciones de vuelo

Este proceso es uno de los problemas combinatorios más complejos que existen, debido a la dificultad que supone asignar personal (incluso con distintas categorías profesionales o condiciones laborales personales) a distintas jornadas laborales en función del lugar donde se encuentren y además cumpliendo la normativa vigente, como son por ejemplo, el descanso mínimo previo a la jornada laboral, las horas máximas de vuelo diarias y mensuales, el máximo número de vuelos diarios, etc. Por todo ello, las diferentes variables existentes nos proporcionan un número limitado de combinaciones posibles, resultando éste un problema difícil de resolver por lo que es mejor dividirlo en dos fases: The Crew Pairing Problem (la asignación de líneas a tripulaciones) y The Crew Rostering Problem (el proceso de construir cuadrantes u horarios para las tripulaciones).

The Crew Pairing Problem es una secuencia de tramos de vuelos (llamados líneas) que comienzan y terminan en la misma base, siempre que sea posible y satisfagan las normas y limitaciones que se tengan que cumplir. Su resolución la podemos representar gráficamente de la siguiente forma:

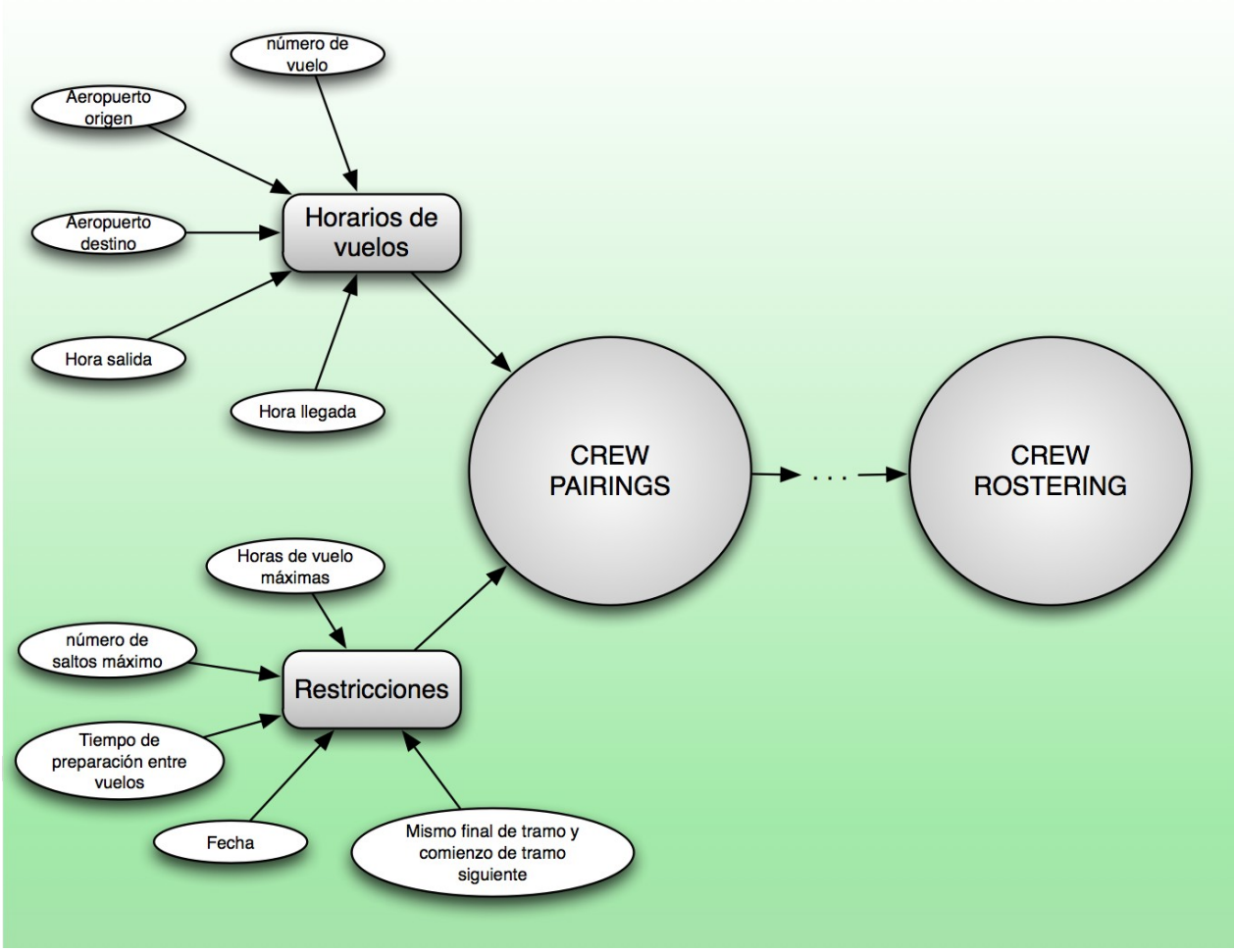

Ilustración 3: Proceso de obtención de los Crew Pairings (líneas de vuelos)

Una línea (o crew paring) puede durar entre uno y ocho días, aunque esto dependerá de la compañía, del tipo de vuelo (doméstico o transatlántico) y de los convenios laborales que existan. El objetivo de la línea es encontrar un grupo de tramos que cubran todos los destinos posibles y minimicen el coste total de las tripulaciones, es decir, generar un grupo de tramos reduciendo el coste y optimizando el máximo de tramos. La línea final resultante incluye fechas y horas por semanas, pues el problema del Crew Pairing es diario, aunque para los fines de semana se hace una aproximación por existir una reducción en el número de vuelos.

Tras la obtención de las líneas finales, nos enfrentamos a la siguiente fase del problema de la programación de las tripulaciones, the Crew Rostering Problem, que es el proceso para asignar tripulantes individuales a las líneas de vuelos (obtenidas en la fase anterior) en periodos mensuales. El objetivo de esta fase es, por tanto, asignar líneas de vuelos (crew pairings que pueden durar varios días) a los tripulantes individualmente para maximizar los recursos humanos. Gráficamente, podemos observar el problema del Crew Rostering en el siguiente esquema:

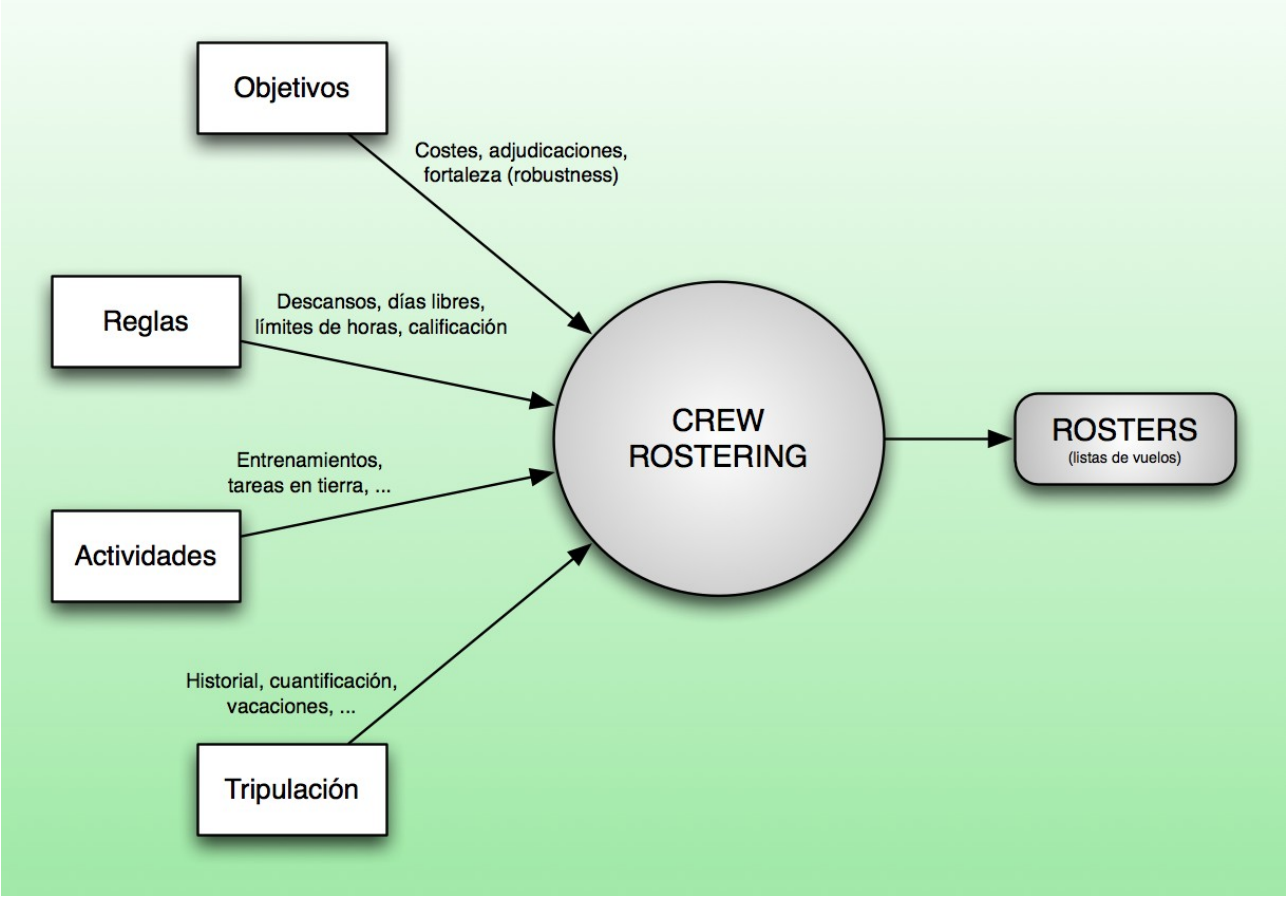

Ilustración 4: Proceso de obtención del Crew Rostering (asignación de vuelos a tripulantes)

Para la asignación de las tripulaciones además de tener en cuenta factores, como son los descansos, cursos de formación de la empresa, periodos de trabajo, etc., debemos considerar también otras variables (con carácter semanal):

- Al menos un día libre entre líneas
- Dos líneas por semana (dependiendo del número de días libres que se deben asignar para cubrir el mes)
- Unas 20 horas de vuelo por semana (dependiendo del total mensual asignado)

Estas variables son aproximaciones que dependerán siempre de los convenios y legislación vigente.

LITERATURE REVIEW

El problema de las programaciones de las tripulaciones (The Crew Scheduling Problem) se puede aplicar en multitud de aspectos de la vida cotidiana con problemas similares. Se trata de un problema en el que una compañía aérea debe cubrir cada tramo de vuelo con sus correspondientes tripulantes de una forma consistente y cumpliendo todas las normas legales y de seguridad que existen a modo de restricciones, tal y como afirma Preston (1992) en su artículo. En este artículo se investigan métodos de particionado para cubrir y resolver el problema, además usa técnicas heurísticas y un ejemplo para reflejarlo. También propone una aplicación de ordenador (The Air Crew Scheduler) para desarrollar y modificar tramos de vuelos (Crew Pairings) y para optimizar y generar programaciones de tripulaciones.

Este problema lleva ya siendo tratado y estudiado varias décadas y, aunque se trata de un problema de difícil solución, se han empleado múltiples técnicas para resolverlo, aunque no siempre con la efectividad deseada. También debemos tener en cuenta que con el paso del tiempo, las compañías aéreas han crecido enormemente y con ellas el problema que estamos tratando, por lo que se precisan métodos prácticos y procedimientos matemáticos más complejos para resolver el problema de forma efectiva, tal y como proponen y afirman Gopalakrishnan y Johnson (2005) en su artículo en el que se hace estudio de las diferentes técnicas utilizadas para tratar el problema en el momento de la creación del artículo, así como una propuesta de futuras investigaciones sobre el tema. En su estudio se centran en concreto en el problema de la asignación de las secuencias de tramos de vuelos a tripulaciones (The Crew Pairing Problem) para que produzca un coste mínimo por tripulante y en el que considera de igual forma a tripulantes técnicos y tripulantes de cabina por influir de igual manera en el problema, aunque se centra en los tripulantes técnicos por suponer un gasto mayor.

Por otro lado, (Graves et al., 1993) enfocan el problema aplicando diferentes reglas según cada categoría de tripulación y según cada tipo de avión, así pueden dividir el problema, haciéndolo más fácil de solucionar, descomponiéndolo en diferentes problemas de asignación de tramos de vuelos. A pesar de tratarse de un problema que puede enfocarse diaria, semanal o mensualmente, Gopalakrishnan y Johnson (2005) lo enfocan diariamente por ser el problema más común, y lo resuelven como un problema de particionado en función de las restricciones o costes a cubrir y mediante el uso de matrices para comprobar si un vuelo (tramo) puede ser cubierto o no.

El problema que tratamos ya era estudiado por las compañías aéreas, que debían solventarlo y optimizarlo de forma adecuada para ahorrar costes y optimizar recursos. Las propias compañías demandaban estos estudios, como el realizado por Arabeyre et al., (1969), en el que el objetivo es optimizar la adecuación de los tripulantes a los vuelos. En este estudio se hace un acercamiento mediante la programación de enteros con variables binarias (0-1) y matrices de coeficientes (binarios), cubriendo la generación, coste, reducción y optimización de estas matrices.

Pero según crecían las compañías aéreas y sus tripulantes, el problema se multiplicaba con matrices del orden de 10^4 columnas y entre 500 – 1000 filas, convirtiéndose en un problema irresoluble. De este modo apareció el estudio de (Rubin 1973) en el que posteriormente se han basado múltiples investigaciones. En este estudio se usa un algoritmo que cubre el problema de forma repetitiva con matrices mucho más pequeñas, extraídas del problema principal, generando las columnas que sean necesarias, tratando el problema principal como múltiples subproblemas.

La división de un problema complejo o NP-hard en subproblemas de menor dificultad es una técnica que se usa mucho, tal y como lo realizan Moudani et al. (2000), que resuelven imprevistos que llevan a la modificación de las programaciones mensuales personales de las tripulaciones. En concreto, tratan el problema de asignar tripulaciones a las líneas de vuelos (Crew Rostering Problem) para minimizar los cambios necesarios en la asignación mensual prevista de las tripulaciones. Proponen una nueva aproximación sistemática basada en la resolución del problema de las asignaciones de forma repetitiva, dividiendo el problema original en subproblemas de menor dificultad más asequibles. A pesar de no conseguir una solución exacta en términos matemáticos puros, si consiguen una aproximación bastante cercana a la solución óptima, adaptada al contexto operacional de las líneas aéreas.

Este tipo de problema NP-hard sobre las programaciones de tripulaciones en las líneas aéreas también es tratado por Ozdemir y Mohan (2001) pero basado en algoritmos genéticos. En concreto su objetivo es obtener soluciones que respeten las distintas necesidades de los tripulantes, minimizando el número de tripulantes necesarios, maximizando el uso del tiempo de las tripulaciones y equilibrando la carga de trabajo entre los tripulantes. Para conseguir este objetivo utilizan un algoritmo genético aplicado a un gráfico dirigido en el que se representan los diferentes vuelos a cubrir, y en el que se tienen en cuenta varias restricciones específicas del problema.

El problema que tratamos es importante, pero previamente debe tratarse otra cuestión, como es la disponibilidad de los aviones y las rutas que deben seguir, al igual que las tripulaciones. Esto debe tenerse en cuenta, ya que sin aviones no van a existir las rutas. Este tratamiento es el que dan Mercier et al. (2005), que definen esta cuestión (rutas de aviones y programación de tripulaciones) como un problema que consiste en determinar un conjunto de coste mínimo para las rutas de los aviones y las rotaciones de los tripulantes, de modo que cada tramo de vuelo sea cubierto por un avión y una tripulación, satisfaciendo todas las necesidades.

La mayoría de las líneas aéreas, después de crear una programación que define las ciudades de origen y de destino, así como los horarios de partida y llegada para cada tramo de vuelo, usan un procedimiento secuencial para planear sus operaciones. Este procedimiento asigna un tipo de avión a cada tramo de vuelo para maximizar los beneficios de antemano, de modo que se cubre exactamente cada tramo una sola vez, por cada avión individualmente. Tras fijar las rutas de los aviones y las distintas limitaciones impuestas por los convenios colectivos, la compañía aérea construye las rotaciones de las tripulaciones, resolviendo el problema de la programación de las tripulaciones. Así, la metodología usada por Mercier et al. (2005) combina la descomposición de Benders y la generación de columnas. La descomposición de Benders, explicada y desarrollada en este artículo y que se usa para reducir el número de variables a costa de aumentar el número de restricciones, resuelve la problemática de la programación de tripulaciones, siempre que la cuestión de las rutas de los aviones haya sido tratada previamente como un subproblema de Benders.

Otros autores, en cambio, como Medard y Sawhney (2007), proponen realizar la construcción y asignación de las rotaciones de las tripulaciones en un sólo paso, en lugar de resolverlo en varios. Estos autores proporcionan técnicas con soluciones basadas en un simple árbol de búsqueda y en algoritmos más sofisticados de generación de columnas y del camino más corto. De este modo, cuando se realizan modificaciones, se tiene que deshacer parte del trabajo hecho durante la fase de elaboración de rotaciones. Estas modificaciones son debidas a cancelaciones de vuelos o retrasos, dejando las rotaciones originales imposibles de realizar para las tripulaciones, pero existe un módulo (programa) de recuperación de rotaciones de Yu et al. (2003) y Wei et al. (1997), en el cual las rotaciones interrumpidas o rotas son reparadas y otras nuevas son construidas para cubrir vuelos imprevistos.

En (Medard y Sawhney 2007) se integran las rotaciones y los modelos convencionales usados en la planificación, y aplican una técnica de generación y optimización que incorpora la construcción y asignación de rotaciones en un sólo paso. Este enfoque directo es posible gracias a una nueva perspectiva de los modelos matemáticos y gracias a la limitación de los periodos de recuperación que reducen el tamaño del problema original.

De forma similar Weide et al. (2010) consideran la problemática de las rotaciones de los aviones junto a la problemática de la rotación de tripulaciones, dividiendo ésta en dos, una para la tripulación técnica y otra para la tripulación de cabina de pasajeros. Proponen que en vez de resolver el modelo total integrado, se resuelvan dos modelos originales de forma repetitiva. Se comienza con un coste mínimo por tripulación en una rotación, sin tener en cuenta las rutas de los aviones. Entonces, en cada repetición, primero resolvemos la problemática de las rotaciones de los aviones, teniendo en cuenta la solución de la actual rotación de la tripulación, por lo que una vez terminadas las repeticiones los dos problemas quedan solucionados.

Normalmente, para la resolución de problemas de programaciones de tripulaciones (Crew Scheduling problems), se utilizan metodologías exactas, como son la programación lineal, la programación dinámica o algoritmos "branch and bound", tal y como se define en (Ángel A. Juan et al. 2010). Aunque en algunos casos estos métodos alcanzan buenos niveles de resolución, en problemas de más de 100 elementos, el éxito en la resolución es variable. Por ello, partiendo de la base de utilizar uno de estos algoritmos heurísticos, como puede ser el algoritmo CWS (Clark and Wright Savings heuristics), aplicado al problema a tratar (Crew Pairing Problem), lo podemos complementar con un proceso de randomización con distribuciones probabilísticas para mejorar así la eficiencia y eficacia de los algoritmos clásicos utilizados.

También, podemos utilizar el algoritmo CWS combinado con el uso de la simulación Monte Carlo (MCS), tal y como hacen en (Ángel A. Juan et al. 2011), que puede ser definido como un grupo de técnicas que hacen uso de números aleatorios y distribuciones probabilísticas para resolver ciertos problemas deterministas o estocásticos. Podemos entonces mejorar los resultados con soluciones numéricas, sobre todo para problemas complejos que no pueden ser eficazmente resueltos usando aproximaciones analíticas. Gracias a este método se introduce un comportamiento aleatorio para mejorar la búsqueda dentro del espacio de las soluciones factibles, que son las que recorren todos los destinos (nodos) tan sólo una vez. En lugar de escoger el destino que más ahorro produce (el más óptimo, que es el método del algoritmo CWS), se escoge uno entre los que más probabilidad tienen de ser escogidos, de entre la lista de los que más ahorro producen. Esta selección se realiza sin introducir demasiados parámetros en el algoritmo, para evitar complicarlo. Se usan distribuciones probabilísticas geométricas durante el proceso de construcción del CWS randomizado, para conseguir todos los objetivos fijados.

Otro método utilizado para resolver la problemática de la programación de tripulaciones de líneas aéreas es el propuesto por Deng y Lin (2011), el cual se basa en la optimización de una colonia de hormigas (Ant Colony Optimization Algorithm). Se basa en intentar buscar el camino más corto desde el gráfico de vuelos al igual que hace el problema del cartero (o del viajante). Los resultados indican que la optimización de la colonia de hormigas es un algoritmo heurístico eficiente y efectivo para minimizar los costes totales de las tripulaciones por medio de programar vuelos efectivos para reducir las estancias nocturnas en hoteles, horas muertas y horas de escalas.

Una vez tratada la cuestión de la asignación de líneas (The Crew Pairing Problem) como el principal problema de las programaciones (The Crew Scheduling Problem), podemos utilizarlo como base para tratar la problemática de la construcción de los cuadrantes para tripulaciones (The Crew Rostering Problem). Este problema se estudia en multitud de investigaciones como en (Ernst et al. 2004), donde realizan una revisión de aplicaciones, métodos y modelos sobre este problema, o en (Maenhout y Vanhoucke 2010), donde utilizan un algoritmo de búsqueda dispersa para el problema de los cuadrantes para las tripulaciones, en el cual se asigna un cuadrante personalizado a cada miembro de la tripulación, minimizando los costes operacionales totales.

Finalmente, esta investigación también puede utilizarse para fortalecer las programaciones generadas. Por ejemplo, se puede añadir una ventana de tiempo adicional a modo de seguro, a la programación resultante, para evitar que se propaguen retrasos, tal y como sugieren Dunbar et al. (2010) en su estudio para minimizar los retrasos y su propagación, en el que calculan de forma precisa, y minimizan el coste del retraso propagado en un marco que integran las rutas de los aviones y los tramos de los vuelos de las tripulaciones. Igualmente lo realizan AhmadBeygi et al. (2008), en el que hacen uso de un árbol de propagación para minimizar que se extiendan los retrasos debido a los vuelos y tripulantes en una ruta existente, por medio de cambios en los horarios de los vuelos, para así utilizar los periodos de inactividad presentes en toda la red de vuelos, al igual que Chiraphadhanakul y Barnhart (2011), que buscan reutilizar periodos de inactividad de forma más simple incluyendo retrasos por propagación de otros retrasos o de pasajeros.

METODOLOGÍA DE RESOLUCIÓN PROPUESTA

Para resolver el problema que tratamos (the Crew Pairing Problem), creamos un algoritmo que realiza y calcula los pairings de las tripulaciones a partir de los tramos de vuelos existentes y que deben ser cubiertos, de modo que dichos tramos se van enlazando y formando los pairings según las restricciones existentes.

Una vez conseguido dicho algoritmo, realizamos una comparativa con los existentes en la actualidad y comparamos los tiempos de procesado para, posteriormente y si es posible, mejorar dichos tiempos en el cálculo y obtención de los pairings. También realizaremos un valoración de los resultados obtenidos.

El algoritmo utilizado para obtener los pairings de los vuelos (nuestras soluciones) que cumplen las restricciones impuestas es:

```
1     procedure generateRoutesOfLegs ()
2           legs = getInputs();
3           sortListOfLegs(legs);
4           while (sol is not null) →
5                 clearRoutes(route);
6                 if we need a leg (anotherLeg is true) →
7                       leg = selectLeg(legs);
8                       route = addLeg(leg);
9                       removeFromList(leg);
10                end if
11                while (legs is not empty) →
12                      sol = searchSolution(legs);
13                      if sol is not null →
14                            addRoute(sol, route);
15                            removeFromList(sol);
16                      else
17                            saveRouteSolution(route);
18                            showRouteSolution(route);
19                            anotherLeg = true;
20                      end if
21                end while  // legs is not empty
22          end while  // sol is not null
23    end
```

Listado 1: Algoritmo principal del artículo

Como podemos observar en la línea 2, vamos a leer los datos de partida, que son los legs que vamos a crear en nuestro algoritmo. Estos datos que generan nuestros legs son leídos de un archivo de datos (test.txt) disponible en modo texto. Dichos datos, en concreto, son los siguientes:

- Número de línea o tramo (número único)
- Aeropuerto de origen del tramo de vuelo concreto
- Aeropuerto de destino del tramo de vuelo en cuestión
- Hora de inicio o de despegue de dicho tramo
- Hora de aterrizaje o de fin del tramo de vuelo tratado
- Fecha en la que se realiza dicho tramo de vuelo

Tras la lectura de estos datos, se modela un objeto mediante una clase de java que contiene nuestro tramo de vuelo de la forma BCN – MAD (por ejemplo), con las horas de despegue y aterrizaje (10:00 – 11:00, por ejemplo), que corresponden al número de vuelo "X" en la fecha dada "Y". En concreto, esta clase modelada sería la clase *Leg* que está contenida en una lista dentro de la clase *Input*, tal y como se puede apreciar en el apéndice donde se encuentra el desarrollo de la aplicación en Java. La correspondiente instrucción en Java que realiza esta acción sería:

```
// Read inputs files and construct the inputs object
Inputs inputs = InputsManager.getInputs(testsFilePath);
```

Listado 2: Código Java que lee los datos de entrada y los modela en una clase

Después de disponer de los datos de entrada, los ordenamos (línea 3 del algoritmo) por la hora de despegue del tramo para ir utilizándolos de una forma más coherente.

Acto seguido, mediante un bucle en el que vamos a ir seleccionando (línea 6) y utilizando todos los tramos de vuelo en el que tenemos como entrada (línea 4), iremos aplicando todas las restricciones impuestas para obtener nuestras soluciones finales. En cada iteración de este bucle, seleccionaremos un tramo de vuelo (el primero de los tramos restantes no utilizados y ordenados) que será el comienzo de nuestro pairing solución parcial. Esta selección (realizada en la línea 7) se añade a la solución parcial (línea 8) y se elimina de la lista de legs que tenemos, debido a que ya se ha utilizado.

Después, mediante otro bucle (línea 11), procedemos a buscar un tramo de vuelo de los tramos restantes que tenemos disponibles para enlazar, que cumple todas las restricciones impuestas (línea 12). En concreto, estas restricciones que vamos a imponer y que son las básicas que exigen las leyes vigentes y los convenios, son las siguientes:

- La primera y más obvia es que el aeropuerto de destino de un leg debe ser el aeropuerto de inicio del siguiente leg enlazado en el pairing solución. Esto es debido a que el avión y la tripulación deben estar en el comienzo del siguiente tramo para poder proseguir con el pairing.
- El tiempo mínimo para poder preparar el avión, embarcar y desembarcar a los pasajeros, prepararse la tripulación, preparar la documentación, etc., será de 1 hora de margen entre tramo y tramo.
- Los pairings serán realizados para la misma fecha, es decir, diarios, aunque luego puedan unirse para formar pairings de varios días.
- El número de tramos máximo que compondrá un pairing será de 5.
- El número máximo de horas de vuelo que puede realizar una tripulación en un día será de 8 horas.

El motivo de elegir estas limitaciones en la creación de los pairings está basado en la experiencia personal del autor de esta investigación trabajando durante 22 años en una compañía aérea como tripulante, además de ser los parámetros habituales que se usan en el desarrollo de las programaciones de las compañías aéreas. En concreto se utiliza 1 hora de escala entre tramos por ser la duración habitual y por comodidad en el cálculo, aunque en vuelos domésticos se suelen utilizar 45 minutos de escala. El valor de 5 tramos por día también es el máximo habitual para este tipo de vuelos.

Estas restricciones serán las que en principio se utilizarán, pudiendo añadirse o modificarse las que sean necesarias, lo que conllevará un aumento del tiempo de procesado del algoritmo y su complejidad. Para facilitar la modificación de las restricciones impuestas en un principio, en el código Java, se crean unas constantes al principio de la clase *Inicio* principal, suficientemente indicativas de su función, y en el que se pueden modificar sus valores, tal y como podemos apreciar en un detalle del código Java de la aplicación:

```
    final static int TIEMPO_ESCALA_MIN = 60;     // Tiempo mínimo de
                                                       escala en minutos
    final static int NUM_MAX_SALTOS = 5;
    final static int HORAS_MAX_VUELO = 800;      // Equivale a 8 horas de
                                                       vuelo máximas
```

Listado 3: Constantes con restricciones que podemos modificar fácilmente

La comprobación de restricciones se realiza en la línea 12 del algoritmo principal, mediante la llamada a otro procedimiento que tiene el siguiente algoritmo:

```
1     procedure searchSolution (legs)
2           solution = null;
3           while list contain legs →
4                 if leg meets all constraints →
5                       solution = actualLeg();
6                 end if
7           end while
8           return solution;
9     end
```

Listado 4: Algoritmo que aplica las restricciones necesarias en cada solución buscada

En este procedimiento se produce un recorrido (línea 3) por todos los tramos disponibles restantes que se le pasan como dato (línea 1: legs). Se comprueba si existe un tramo que cumpla todas las restricciones que le hemos impuesto (línea 4) y si es así, la devuelve como solución para añadirla al pairing que estamos creando como parte de una de las soluciones finales. Si no, devuelve el valor "null" (línea 2) que indica que no encontró una solución.

La parte de la aplicación Java que corresponde a este procedimiento en el que se busca un Leg que cumpla las restricciones impuestas, se puede apreciar en el siguiente detalle del código Java de la aplicación:

```
/**
 * Función que busca una solución factible con los condicionantes
 * requeridos
 *
 * @param legs
 * @param destino
 * @param horaInicio
 * @param horaFin
 * @param fecha
 * @param numSaltos
 * @param numHorasVuelo
 * @return Devuelve un Leg que cumple las condiciones impuestas
 */
public static Leg searchSol(ArrayList<Leg> legs, String destino,
int horaInicio, int horaFin, String fecha, int numSaltos, int
numHorasVuelo){
     Leg sol = null;
     boolean found = false;
     int horas = 0;
     int suma = 0;

     for (Leg i: legs)
          if ((i.getFecha().equals(fecha)) &&
               (i.getOrigen().equals(destino)) &&
               ((calcDifMin(horaFin, i.getHoraInicio())) >=
               TIEMPO_ESCALA_MIN) && (i.getHoraInicio() >
               horaInicio) && (numSaltos < NUM_MAX_SALTOS) &&
               (numHorasVuelo <= HORAS_MAX_VUELO)) {
               horas = calcFlightHours(i.getHoraInicio(),
                         i.getHoraFin());
               suma = suma + horas;
               suma = addFlightHours(suma, numHorasVuelo);
               if ((found == false) && (suma <=
                         HORAS_MAX_VUELO)) {
                    sol = i;
                    found = true;
               }
          }

     return sol;
}
```

Listado 5: Código Java del algoritmo que busca soluciones aplicando las restricciones impuestas.

Continuando con el algoritmo principal (*procedure generateRoutesOfLegs()*), si encontramos un tramo que cumpla todas las restricciones (línea 13), la añadimos al pairing solución parcial que estamos creando y eliminamos de la lista el tramo utilizado (línea 15) que dejará de estar disponible. Si no encontramos un tramo que cumpla las restricciones (línea 16) significa que ya no podemos seguir creando o ampliando dicho pairing y pasa a ser una de las soluciones parciales finales, por lo que lo guardamos como parte de la solución (línea 17) y proseguimos con los tramos restantes (línea 19).

Una vez finalizadas todas las iteraciones y utilizados todos los tramos disponibles, ya disponemos (tanto por pantalla como grabados en un archivo) de todos los pairings que se pueden generar con las condiciones impuestas, con lo que ya hemos conseguido tratar el problema del pairing de las tripulaciones de una forma más o menos eficiente, que era lo que buscábamos. En general, el esquema lógico de funcionamiento del algoritmo, representado mediante un diagrama de flujo, el cual nos permite apreciar los bucles de iteración, las restricciones aplicadas, etc., que sería el siguiente:

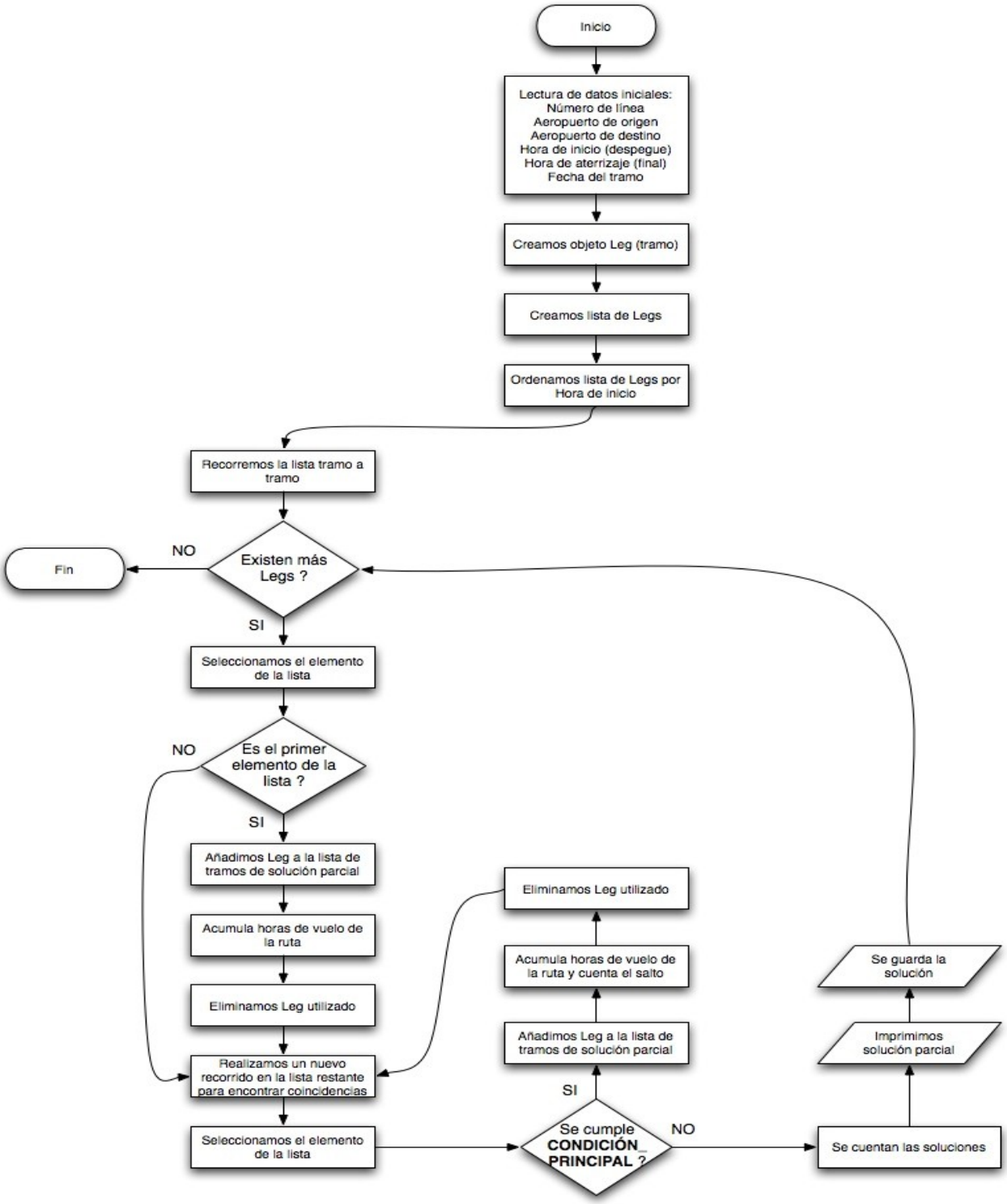

Ilustración 5: Diagrama de flujo del algoritmo

La parte en código Java que refleja el funcionamiento principal del algoritmo y es mostrada por el diagrama de flujo es la siguiente:

```
do{
      // Vaciamos la ruta para empezar otro tramo
      ruta.clear();
      // Reiniciamos el contador de tramos
      numSaltos = 1;


      if (primero == true){
            // Seleccionamos los datos necesarios del primer
            elemento de la lista
            destino = legs.get(0).getDestino();
            horaInicio = legs.get(0).getHoraInicio();
            horaFin = legs.get(0).getHoraFin();
            fecha = legs.get(0).getFecha();
            // Añadimos el primer elemento del tramo (si es el
            primero utilizado)
            ruta.add(legs.get(0));
            // Calculamos las horas de vuelo del tramo
            numHorasVuelo = calcFlightHours (horaInicio, horaFin);
            // Sumamos al total de horas de vuelo que tenemos
            totalHorasVuelo = addFlightHours(totalHorasVuelo,
                  numHorasVuelo);
            // Borramos el elemento usado (si es el primero
            utilizado)
            legs.remove(0);
            primero = false;
      }


      do {
            // Buscamos un Leg solucion
                  l = searchSol(legs, destino, horaInicio, horaFin,
                        fecha, numSaltos, totalHorasVuelo);

            if (l != null){
                  // Añadimos el Leg a la ruta
                  ruta.add(l);
                  // Contamos el salto
                  numSaltos++;

                  destino = l.getDestino();
                  horaInicio = l.getHoraInicio();
                  horaFin = l.getHoraFin();
                  fecha = l.getFecha();


                  // Calculamos las horas de vuelo del tramo
                  numHorasVuelo = calcFlightHours (horaInicio,
                        horaFin);
                  // Sumamos al total de horas de vuelo que tenemos
```

```
                    totalHorasVuelo = addFlightHours(totalHorasVuelo,
                         numHorasVuelo);

                    // Borramos el Leg utilizado
                    int indice = legs.indexOf(l);
                    legs.remove(indice);
               } else {
                    // Imprimimos solución en pantalla
                    numSols++;
                    solution = showSol(ruta, numSols, totalHorasVuelo);

                    // Guardamos la solución en un archivo de texto
                    (out.txt)
                    saveSolution(solution, outputFilePath);

                    primero=true;
                    totalHorasVuelo = 0;

                    }

          } while (l != null);       // hasta que no existan
                                     coincidencias de Legs
     } while (legs.size() > 0);      // hasta que se vacíe la lista de
                                     Legs
```

Listado 6: Código Java del algoritmo principal que refleja la creación de los Crew Pairings.

En este código podemos apreciar los dos bucles que realizan los recorridos y que van formando los tramos solución de la aplicación (representados por bucles *do...while*), haciendo la llamada a la función *searchSol*, que es la encargada de comprobar que se cumplen las restricciones.

Para obtener el tiempo de procesado del algoritmo creado, se realiza una clase de Java que utiliza librerías propias de este lenguaje. Así podemos calcular al inicio de la aplicación, la hora del sistema y al finalizar, volviendo a calcular la hora del sistema, podemos obtener la diferencia de tiempo entre ambas. Esto nos indica el tiempo de ejecución o procesado de la aplicación creada en horas, minutos y segundos.

En un principio, el algoritmo y sus restricciones serán aplicados en vuelos domésticos y en pairings diarios, que es la base de trabajo para solucionar el problema de los pairings. Los vuelos transatlánticos o de mayor duración tienen otras limitaciones en cuanto a número de horas de vuelo (que podrán ser mayores de 8 horas) y a número de tramos (normalmente 1), por lo que el algoritmo sería distinto aunque la base es la misma. Por ello habría que diseñar un nuevo algoritmo con una nueva investigación o adaptar el utilizado con los nuevos parámetros.

EXPERIMENTOS COMPUTACIONALES

Una vez que hemos conseguido desarrollar nuestro algoritmo como una aplicación Java, ponemos nuestra aplicación a prueba, mediante una serie de experimentos para probar su efectividad.

Tras someterlo a una batería de pruebas y comprobar el correcto funcionamiento con 50, 100, 150 y 500 vuelos diarios, seleccionamos un grupo de vuelos reales de la compañía aérea Iberia L. A. E. como muestra, acorde a nuestro algoritmo, para comparar los resultados obtenidos. Los resultados de Iberia son:

IBERIA
LINEAS AEREAS DE ESPAÑA

A-320 TRIPULANTES PAG. 8

		H.V	ALR	ALP	ALT	F.B.	V.S	P.V	ETPS	DSCNSO	DIF
* I1446 2 0 030100 240100 43											
0548 0551 3244											
L MAD . SCQ . MAD . /AMS/											
(1520) 1505-1615 1705-1810 1910-2130											
		4.58	7.67	12.50	3.08	9.67	.00	.00	3		
(3249)											
M AMS . MAD M											
(1855) 1840-2105											
		2.42	3.67	12.50	1.25	21.58	.00	.58	1	19.92	9.42
	RESUMEN 0 2	7.00	11.34	25.00	4.33	31.25	.00	0 .58	4	19.92	
* I1447 3 0 010100 010100 44		H.V	ALR	ALP	ALT	F.B.	V.S	P.V	ETPS	DSCNSO	DIF
0554											
S MAD . SCQ											
(1800) 1745-1855											
		1.17	2.42	12.50	1.25	7.00	.00	1.83	1		
VS0541 2600 4219 1569											
D SCQ . MAD . (BCN) . SCQ . BCN											
(0825) 0810-0915 1130-1230 1505-1640 1735-1905											
		4.08	12.17	13.00	7.00	24.00	.53	.00	4	12.00	1.50
(4424) 4427 VS1945											
L BCN . ORY . BCN . (MAD) L											
(1330) 1315-1455 1545-1720 1845-1945											
		3.25	7.75	14.50	3.50	20.25	.50	.00	3	16.92	2.75
	RESUMEN 0 3	8.50	22.34	40.00	11.75	51.25	1.03	0 1.83	8	28.92	
* I1448 3 0 020100 020100 45		H.V	ALR	ALP	ALT	F.B.	V.S	P.V	ETPS	DSCNSO	DIF
0800 4626 4627											
D MAD . BCN . FCO . (BCN)											
(0715) 0700-0800 0845-1025 1140-1320											
		4.33	7.58	13.50	3.25	17.75	.00	.00	3		
4626 4627 4662 4659											
L BCN . FCO . (BCN) . MXP . BCN											
(0900) 0845-1025 1140-1320 1510-1640 1735-1920											
		6.58	11.83	13.00	5.25	24.00	.00	.00	4	18.17	6.58
4570 3531											
M BCN . MUC . (MAD) M											
(1530) 1515-1715 1800-2035											
		4.58	6.58	13.00	2.00	21.08	.00	.00	2	18.67	4.83
	RESUMEN 0 3	15.49	25.99	39.50	10.50	62.83	.00	0 .00	9	36.84	
* I1449 3 0 020100 020100 46		H.V	ALR	ALP	ALT	F.B.	V.S	P.V	ETPS	DSCNSO	DIF
VS0800 6181 (3456) 3457 VS1700											
D MAD . BCN . MAD . NCE . MAD . BCN											
(0715) 0700-0800 0855-0955 1050-1230 1320-1505 1600-1700											
		4.42	11.25	12.75	4.83	17.75	1.00	.00	5		
6181 (3456) 3457 6970											
L BCN . MAD . NCE . MAD . LPA											
(0910) 0855-0955 1050-1230 1320-1505 1635-1925											
		7.25	11.75	13.00	4.50	24.00	.00	.00	4	14.67	1.42
VS0807 (3502) (3503)											
M LPA . MAD . FRA . MAD M											
(1035) 1120-1350 1505-1735 1825-2055											
		5.00	10.83	13.50	3.33	21.42	1.25	.00	3	14.67	.92
	RESUMEN 0 3	16.67	33.83	39.25	12.66	63.17	2.25	0 .00	12	29.34	
* I1450 2 0 090100 300100 47		H.V	ALR	ALP	ALT	F.B.	V.S	P.V	ETPS	DSCNSO	DIF
0800 4216											
D MAD . BCN . BRU											
(0715) 0700-0800 0900-1105											
		3.08	5.33	14.00	2.25	17.75	.00	.00	2		
3207											
L BRU . MAD L											
(0730) 0715-0935											
		2.33	3.58	14.00	1.25	10.08	.00	.67	1	18.92	8.42
	RESUMEN 2	5.41	8.91	28.00	3.50	27.83	.00	.67	3	18.92	
* I1451 3 0 140100 280100 48		H.V	ALR	ALP	ALT	F.B.	V.S	P.V	ETPS	DSCNSO	DIF
0800 4626 4627											
V MAD . BCN . FCO . /BCN/											
(0715) 0700-0800 0845-1025 1140-1320											
		4.33	7.58	13.50	3.25	17.75	.00	.00	3		
4422 4423											
S BCN . ORY . BCN											
(0640) 0625-0805 0855-1030											
		3.25	5.33	14.00	2.08	24.00	.00	.00	2	15.83	4.25
1702 1707 6181											
D BCN . PMI . BCN . MAD D											
(0605) 0550-0630 0715-0800 0855-0955											
		2.42	5.33	12.50	2.92	10.42	.00	.58	3	18.08	7.58
	RESUMEN 0 3	10.00	18.24	40.00	8.25	52.17	.00	0 .58	8	33.91	

Ilustración 6: Tabla real de los Pairings de una compañía aérea

Los datos de entrada que se utilizarán son, a modo de ejemplo y para las dos primeras líneas:

```
0548   MAD    SCQ    1505   1615   03/09/2012
0551   SCQ    MAD    1705   1810   03/09/2012
3244   MAD    AMS    1910   2130   03/09/2012
3249   AMS    MAD    1840   2105   04/09/2012
…
```

Listado 7: Ejemplo de datos de entrada utilizados en la comparativa con aplicación real.

Seguiríamos añadiendo datos hasta completar la muestra comparativa. También debemos tener en cuenta que la compañía aérea Iberia L. A. E. utiliza como escala mínima 45 minutos para vuelos de corto radio o domésticos, que introduciremos como parámetro en la constante TIEMPO_ESCALA_MIN. También utiliza 5 tramos de vuelo como máximo para una jornada laboral (parámetro NUM_MAX_SALTOS). Supondremos también que el número máximo de horas de vuelo para una jornada será de 8 (parámetro HORAS_MAX_VUELO).

En primer lugar, vamos a realizar una batería de pruebas tal y como viene presentada en el modelo, para así comprobar su efectividad; es decir, una fecha para cada pairing, tal y como viene en la muestra. El resultado obtenido por nuestro algoritmo es:

```
**************************************************************************
**********************   S O L U C I Ó N   1  ****************************
**************************************************************************

 fecha =                              18/09/2012
 horas de vuelo realizadas =          2 horas y 25 minutos (2.41)

 numLineas =         1702       1707       6181
               BCN ------ PMI ------ BCN ------ MAD
                 0550-0630  0715-0800  0855-0955

**************************************************************************
**********************   S O L U C I Ó N   2  ****************************
**************************************************************************

 fecha =                              17/09/2012
 horas de vuelo realizadas =          3 horas y 15 minutos (3.25)

 numLineas =         4422       4423
               BCN ------ ORY ------ BCN
                 0625-0805  0855-1030

**************************************************************************
**********************   S O L U C I Ó N   3  ****************************
**************************************************************************

 fecha =                              08/09/2012
 horas de vuelo realizadas =          4 horas y 20 minutos (4.33)

 numLineas =         800       4626       4627
               MAD ------ BCN ------ FCO ------ BCN
                 0700-0800  0845-1025  1140-1320
```

```
*************************************************************************
**********************   S O L U C I Ó N   4  ***************************
*************************************************************************

 fecha =                              11/09/2012
 horas de vuelo realizadas =          6 horas y 25 minutos (6.41)

 numLineas =        800       6181       3456       3457       1700
               MAD ------ BCN ------ MAD ------ NCE ------ MAD ------ BCN
                 0700-0800  0855-0955  1050-1230  1320-1505  1600-1700

*************************************************************************
**********************   S O L U C I Ó N   5  ***************************
*************************************************************************

 fecha =                              14/09/2012
 horas de vuelo realizadas =          3 horas y 5 minutos (3.08)

 numLineas =        800       4216
               MAD ------ BCN ------ BRU
                 0700-0800  0900-1105

*************************************************************************
**********************   S O L U C I Ó N   6  ***************************
*************************************************************************

 fecha =                              16/09/2012
 horas de vuelo realizadas =          4 horas y 20 minutos (4.33)

 numLineas =        800       4626       4627
               MAD ------ BCN ------ FCO ------ BCN
                 0700-0800  0845-1025  1140-1320

*************************************************************************
**********************   S O L U C I Ó N   7  ***************************
*************************************************************************

 fecha =                              15/09/2012
 horas de vuelo realizadas =          2 horas y 20 minutos (2.33)

 numLineas =        3207
               BRU ------ MAD
                 0715-0935

*************************************************************************
**********************   S O L U C I Ó N   8  ***************************
*************************************************************************

 fecha =                              06/09/2012
 horas de vuelo realizadas =          5 horas y 10 minutos (5.16)

 numLineas =        541       2600       4219       1569
               SCQ ------ MAD ------ BCN ------ SCQ ------ BCN
                 0810-0915  1130-1230  1505-1640  1735-1905

*************************************************************************
**********************   S O L U C I Ó N   9  ***************************
*************************************************************************

 fecha =                              09/09/2012
 horas de vuelo realizadas =          6 horas y 35 minutos (6.58)

 numLineas =        4626       4627       4662       4659
               BCN ------ FCO ------ BCN ------ MXP ------ BCN
                 0845-1025  1140-1320  1510-1640  1735-1920
```

```
*************************************************************************
**********************   S O L U C I Ó N   10  **************************
*************************************************************************

 fecha =                              12/09/2012
 horas de vuelo realizadas =          7 horas y 15 minutos (7.25)

 numLineas =         6181       3456       3457       6970
               BCN ------ MAD ------ NCE ------ MAD ------ LPA
                 0855-0955  1050-1230  1320-1505  1635-1925

*************************************************************************
**********************   S O L U C I Ó N   11  **************************
*************************************************************************

 fecha =                              13/09/2012
 horas de vuelo realizadas =          7 horas y 30 minutos (7.50)

 numLineas =         807       3502       3503
               LPA ------ MAD ------ FRA ------ MAD
                 1120-1350  1505-1735  1825-2055

*************************************************************************
**********************   S O L U C I Ó N   12  **************************
*************************************************************************

 fecha =                              07/09/2012
 horas de vuelo realizadas =          4 horas y 15 minutos (4.25)

 numLineas =         4424       4427       1945
               BCN ------ ORY ------ BCN ------ MAD
                 1315-1455  1545-1720  1845-1945

*************************************************************************
**********************   S O L U C I Ó N   13  **************************
*************************************************************************

 fecha =                              03/09/2012
 horas de vuelo realizadas =          4 horas y 35 minutos (4.58)

 numLineas =         548       551       3244
               MAD ------ SCQ ------ MAD ------ AMS
                 1505-1615  1705-1810  1910-2130

*************************************************************************
**********************   S O L U C I Ó N   14  **************************
*************************************************************************

 fecha =                              10/09/2012
 horas de vuelo realizadas =          4 horas y 35 minutos (4.58)

 numLineas =         4570       3531
               BCN ------ MUC ------ MAD
                 1515-1715  1800-2035

*************************************************************************
**********************   S O L U C I Ó N   15  **************************
*************************************************************************

 fecha =                              05/09/2012
 horas de vuelo realizadas =          1 horas y 10 minutos (1.16)

 numLineas =         554
               MAD ------ SCQ
                 1745-1855
```

```
*************************************************************************
**********************   S O L U C I Ó N   16  **************************
*************************************************************************

 fecha =                              04/09/2012
 horas de vuelo realizadas =          2 horas y 25 minutos (2.41)

 numLineas =         3249
               AMS ------ MAD
                 1840-2105
```

Listado 8: Salida obtenida por la aplicación al introducir los mismos vuelos con los mismos parámetros utilizados por la compañía Iberia.

Como podemos comprobar nuestras soluciones coinciden exactamente con la muestra, aunque ordenado por las horas de los vuelos. En las "horas de vuelo realizadas =" de nuestros resultados, hemos incluido entre paréntesis las horas de vuelo realizadas en tanto por ciento, para poder compararlas con las de la muestra utilizada presentes en la columna "H. V", las cuales vienen en este mismo formato. Como podemos observar, coinciden exactamente salvo algún redondeo en el último decimal y en los vuelos cuyo número comienza por VS (vuelo en situación; es decir, sin trabajar). En ese caso la duración se considera, a efectos de salario, como si fuese la mitad. Esta posibilidad no ha sido considerada en nuestro algoritmo por ser muy particular y depender de los convenios de cada compañía aérea.

La segunda batería de pruebas la realizamos teniendo en cuenta el objetivo con el que fue diseñado el algoritmo; es decir, conseguir la máxima eficiencia para un número de vuelos dado en la misma fecha y con los mismos parámetros. La salida que obtenemos, suponiendo que dichos vuelos deben realizarse el mismo día, es:

```
*************************************************************************
**********************   S O L U C I Ó N   1  ***************************
*************************************************************************

 fecha =                              03/09/2012
 horas de vuelo realizadas =          6 horas y 20 minutos (6.33)

 numLineas =         1702       1707       4626       4627       4219
               BCN ------ PMI ------ BCN ------ FCO ------ BCN ------ SCQ
                 0550-0630  0715-0800  0845-1025  1140-1320  1505-1640

*************************************************************************
**********************   S O L U C I Ó N   2  ***************************
*************************************************************************

 fecha =                              03/09/2012
 horas de vuelo realizadas =          7 horas y 30 minutos (7.50)

 numLineas =         4422       4423       4424       4427       1945
               BCN ------ ORY ------ BCN ------ ORY ------ BCN ------ MAD
                 0625-0805  0855-1030  1315-1455  1545-1720  1845-1945
```

```
*************************************************************************
**********************   S O L U C I Ó N   3  ***************************
*************************************************************************

 fecha =                              03/09/2012
 horas de vuelo realizadas =          7 horas y 35 minutos (7.58)

 numLineas =        800       4626       4627       4662       4659
               MAD ------ BCN ------ FCO ------ BCN ------ MXP ------ BCN
                 0700-0800  0845-1025  1140-1320  1510-1640  1735-1920

*************************************************************************
**********************   S O L U C I Ó N   4  ***************************
*************************************************************************

 fecha =                              03/09/2012
 horas de vuelo realizadas =          6 horas y 20 minutos (6.33)

 numLineas =        800       4626       4627       4570
               MAD ------ BCN ------ FCO ------ BCN ------ MUC
                 0700-0800  0845-1025  1140-1320  1515-1715

*************************************************************************
**********************   S O L U C I Ó N   5  ***************************
*************************************************************************

 fecha =                              03/09/2012
 horas de vuelo realizadas =          6 horas y 25 minutos (6.41)

 numLineas =        800       6181       3456       3457       1700
               MAD ------ BCN ------ MAD ------ NCE ------ MAD ------ BCN
                 0700-0800  0855-0955  1050-1230  1320-1505  1600-1700

*************************************************************************
**********************   S O L U C I Ó N   6  ***************************
*************************************************************************

 fecha =                              03/09/2012
 horas de vuelo realizadas =          5 horas y 25 minutos (5.41)

 numLineas =        800       6181       3456       3457
               MAD ------ BCN ------ MAD ------ NCE ------ MAD
                 0700-0800  0855-0955  1050-1230  1320-1505

*************************************************************************
**********************   S O L U C I Ó N   7  ***************************
*************************************************************************

 fecha =                              03/09/2012
 horas de vuelo realizadas =          3 horas y 20 minutos (3.33)

 numLineas =        3207       2600
               BRU ------ MAD ------ BCN
                 0715-0935  1130-1230

*************************************************************************
**********************   S O L U C I Ó N   8  ***************************
*************************************************************************

 fecha =                              03/09/2012
 horas de vuelo realizadas =          5 horas y 40 minutos (5.66)

 numLineas =        541       548       551       3244
               SCQ ------ MAD ------ SCQ ------ MAD ------ AMS
                 0810-0915  1505-1615  1705-1810  1910-2130
```

```
************************************************************************
**********************   S O L U C I Ó N   9  **************************
************************************************************************

 fecha =                              03/09/2012
 horas de vuelo realizadas =          6 horas y 0 minutos (6.00)

 numLineas =        6181       3502       3503
               BCN ------ MAD ------ FRA ------ MAD
                 0855-0955  1505-1735  1825-2055

************************************************************************
**********************   S O L U C I Ó N   10  *************************
************************************************************************

 fecha =                              03/09/2012
 horas de vuelo realizadas =          2 horas y 5 minutos (2.08)

 numLineas =        4216
               BCN ------ BRU
                 0900-1105

************************************************************************
**********************   S O L U C I Ó N   11  *************************
************************************************************************

 fecha =                              03/09/2012
 horas de vuelo realizadas =          5 horas y 20 minutos (5.33)

 numLineas =        807       6970
               LPA ------ MAD ------ LPA
                 1120-1350  1635-1925

************************************************************************
**********************   S O L U C I Ó N   12  *************************
************************************************************************

 fecha =                              03/09/2012
 horas de vuelo realizadas =          1 horas y 30 minutos (1.50)

 numLineas =        1569
               SCQ ------ BCN
                 1735-1905

************************************************************************
**********************   S O L U C I Ó N   13  *************************
************************************************************************

 fecha =                              03/09/2012
 horas de vuelo realizadas =          1 horas y 10 minutos (1.16)

 numLineas =        554
               MAD ------ SCQ
                 1745-1855

************************************************************************
**********************   S O L U C I Ó N   14  *************************
************************************************************************

 fecha =                              03/09/2012
 horas de vuelo realizadas =          2 horas y 35 minutos (2.58)

 numLineas =        3531
               MUC ------ MAD
                 1800-2035
```

```
*************************************************************************
**********************   S O L U C I Ó N   15  **************************
*************************************************************************

 fecha =                              03/09/2012
 horas de vuelo realizadas =          2 horas y 25 minutos (2.41)

 numLineas =         3249
               AMS ------ MAD
                 1840-2105
```

Listado 9: Soluciones obtenidas por nuestra aplicación al utilizar los mismos vuelos como datos de entrada, pero aplicando la misma fecha.

Con nuestro algoritmo se consiguen ajustar más las tripulaciones, reordenando los tramos de vuelo.

DISCUSIÓN DE RESULTADOS

Tal y como podemos observar en los resultados obtenidos, en igualdad de condiciones, se obtienen exactamente los mismos resultados (asignando una fecha para cada grupo de tramos). Este grupo de pruebas no tiene interés práctico, pero si nos permite comprobar la efectividad de nuestro algoritmo, ya que nos confirma que realiza exactamente el cometido para el que fue diseñado.

El resto de columnas distintas de la "H. V" de la muestra, no presenta ningún interés, ya que son datos que se calculan en función de las horas de vuelo y acuerdos legales, por lo que coinciden haciendo los cálculos correspondientes, con los que podamos obtener nosotros. Estos son, por ejemplo, ALR (actividad laboral realizada), que simplemente suma algunas horas fijas por las escalas, ALP (actividad laboral permitida), que corresponde al límite legal permitido para es número de saltos y según horario, etc.

En cambio, en la segunda batería de pruebas realizada, sí obtenemos diferencias, comprobando que logramos ajustar más los resultados obtenidos, tal y como reflejamos en la siguiente tabla comparativa:

	Legs	**H. V. R**	**H. V. O**	**H. V. S**	**N. P. T**	**N. M. S**	**N. P. M**	**N. H. V. P**	**H. V. M**
Iberia	44	69.61	63.07	3.29	16	3	1	4.33	4.35
1ª batería de pruebas	44	69.61	69.61	__	16	3	1	4.33	4.35
2ª batería de pruebas	44	69.61	69.61	__	15	3	4	5.41	4.64
Total de horas de vuelo contabilizadas = 69 horas 36 minutos 36 segundos Número de tramos contabilizados = 44									

Tabla 1: Tabla comparativa de resultados entre Iberia y nuestro algoritmo

Leyenda:

- H. V. R = Horas de vuelo reales
- H. V. O = Horas de vuelo obtenidas
- H. V. S = Horas de vuelo en situación (sin trabajar)
- N. P. T = Número de pairings totales
- N. M. S = Número medio de saltos por pairing
- N. P. M = Número de pairings ajustados al máximo (5 saltos)
- N. H. V. P = Número medio de horas de vuelo por pairing
- H. V. M = Número de horas de vuelo realizadas de media

A pesar de no ser una muestra de prueba muy amplia (44 vuelos), conseguimos optimizar y mejorar los resultados obtenidos por la compañía aérea utilizada como ejemplo, logrando ahorrar una tripulación (1 pairing menos) con el mismo número medio de saltos por pairing, aumentando en unas décimas las horas de vuelo de media. También se aumenta el número medio de horas que se realiza por pairing consiguiendo un mayor ajuste, haciendo que tengan un mayor número de saltos aunque con el mismo número medio de saltos por pairing, con lo que el reparto sigue siendo equilibrado.

También podemos observar que existe una diferencia de horas con respecto a las horas de vuelo obtenidas. Esto es debido a que Iberia calcula las horas de vuelo en situación como la mitad de horas con respecto a las reales, por lo que si multiplicamos H. V. S por 2 y se las sumamos a H. V. O, obtenemos 69.65 horas de vuelo obtenidas, que difiere de 69.61 por los redondeos. Como ya comentamos con anterioridad, nuestro algoritmo no tiene en cuenta esta situación.

CONCLUSIONES

Mediante la programación en Java y por medio de la utilización del Software Libre (entorno Eclipse, Java), hemos conseguido solucionar el problema del Crew Pairing mediante el desarrollo de un algoritmo propio. Las compañías aéreas logran este objetivo (enlazar diferentes Legs para obtener Pairings cumpliendo todas las restricciones necesarias) gracias a la elaboración manual de dichos Pairings, con el consiguiente gasto en personal y recursos, o mediante costosas aplicaciones desarrolladas por terceros (otras compañías de software), como puede ser el famoso programa de Software llamado CARMEN, adquirido por la compañía Jeppesen (http://ww1.jeppesen.com/index.jsp) en 2006, perteneciente a la compañía Boeing. En concreto, trabajan todos los aspectos sobre las programaciones, así como el Jeppesen Crew Pairing (http://ww1.jeppesen.com/industry-solutions/aviation/commercial/carmen-crew-pairing.jsp). Con el desarrollo de nuestro algoritmo conseguimos nuestro objetivo con costes mucho menores y con un tiempo computacional realmente bajo (apenas 1 segundo en todas las pruebas), incluso en la generación de Pairings para grandes compañías con muchos vuelos diarios.

También podemos ajustar los resultados que queremos obtener con tan sólo modificar los parámetros. Por ejemplo, si quisiéramos tener pairings con mayor número de vuelos (legs) por tripulación, bastaría con aumentar el número de horas de vuelo permitidas. También, si quisiéramos equilibrar el número de saltos que hace cada tripulante, sería suficiente con disminuir el número de legs permitido, obteniendo así pairings más equilibrados. Se debe tener en cuenta que estas modificaciones deben estar dentro de los límites legales.

De todas formas no descartamos utilizar otras técnicas matemáticas para mejorar este algoritmo, como puede ser la aplicación de randomización sobre los datos de entrada o la aplicación de técnicas estadísticas para mejorar la eficiencia del algoritmo que, aunque en este momento no sea necesario, en un futuro puede serlo si utilizamos esta investigación como base para otras, como puede ser para el estudio del problema del Crew Rostering, contenido también en la problemática del Crew Scheduling.

REFERENCIAS

- *AhmadBeygi, Shervin; Cohn, Amy and Lapp, Marcial, "Decreasing airline delay propagation by re-allocating scheduled slack". IEE Transactions. Vol. 42, no. 7, pages 478-489. 2010*

- *Angel A. Juan, Javier Faulin, Rubén Ruiz, Barry Barrios and Santi Caballé, "The SR-GCWS hybrid algorithm for solving the capacited vehicle routing problem". Applied Soft Computing 10, pages 215-224. 2010*

- *Angel A. Juan, Javier Faulin, J. Jorba, D. Riera, D. Masip and Barry Barrios, "On the use of Monte Carlo simulation, cache and splitting techniques to improve the Clarke and Wright savings heuristics". Journal of the Operational Research Society 62, pages 1085-1097. 2011*

- *Bazargan Massoud, "Airline Operations and Scheduling 2nd Edition". Ashgate Publishing Limited, Embry-Riddle Aeronautical University, USA. 2010*

- *Chiraphadhanakul, Virot and Barnhart, Cynthia, "Robust flight schedules through slack re-allocation". Operations Research Center, Massachusetts Institute of Technology. 2011*

- *Deng, Guang Fen and Lin, Woo-Tsong, "Ant Colony optimization-based algorithm for airline crew scheduling problem". Expert Systems with Applications. Vol. 38, no. 5, pages 5787-5793. 2011*

- *Dunbar, Michelle and Froyland, Gary, "Robust Airline schedule planning: Minimizing propagated delay in an integrated routing and crewing framework". School of Mathematics and Statistics, University of New South Wales. 2010*

- *El Moudani, W; Brochado MR; Handou, M and Mora-Camino, F, "A fuzzy reactive approach for the crew rostering problem". Current Advances in Mechanical Design and Production VII. Vol. 7, pages 611-619. 2000*

- *Gopalakrishnan, Balaji and Johnson, Ellis J. "Airline Crew Scheduling: state-of-the-Art", Annals of Operations Research 140, pages 305-337. 2005*

- *Graves, G. W., R. D. McBride and Gershkoff. "Flight Crew Scheduling". Management Science 39(6), pages 736-745. 1993*

- *Medard, Claude P and Sawhney, Nidhi, "Airline Crew Scheduling from planning to operations". European Journal of Operational Research. Vol. 38, no. 3, pages 1013-1027. 2007*

- *Mercier, A; Cordeau, JF and Soumis, F, "A computacional study of Benders decomposition for the integrated aircraft routing and crew scheduling problem". Computers & Operations Research. Vol. 32, no. 6, pages 1451-1476. 2005*

- *Ozdemir, HT and Mohan, CK, "Flight graph based genetic algorithm for crew scheduling in airlines". Information Sciencies. Vol. 133, no. 3-4, pages 165-173. 2001*

- *Preston. "Airline Crew Scheduling". Preston Group Pty Ltd. 1992*

- *Wei, Guo; Yu, Gang and Song, Mark. "Optimization model and algorithm for crew management during irregular operations". Journal of Combinatorial Optimization. Vol. 1, pages 80-97. 1997*

- *Weide, Oliver; Ryan, David and Ehrgott, Matthias, "An iterative approach to robust and integrated aircraft routing and crew scheduling". Computers & Operations Research. Vol. 37, no. 5, pages 833-844. 2010*

- *Yen, JW and Birge, JR, "A Stochastic programming approach to the airline crew scheduling problem". Transportation Science. Vol. 40, no. 1, pages 3-14. 2006*

- *Yu, Gang; Arguello, Michael; Song, Gao; McCowan, Sandra and White, Anna. "A new era for crew recovery at Continental Airlines". Interfaces 33. Vol. 1, pages 5-22. 2003*

APÉNDICE

- Código Java de la aplicación desarrollada:

```
package CrewPairingP;

/**************************************************************************
* Project SimORouting -  ElapsedTime.java
* Given an elapsed time in seconds, the method printHMS() of this class returns
*  an string with the equivalent time in hours, minutes and seconds.
* Given an elapsed time in seconds, the method doPause() of this class forces
*  the code to do a pause for a time interval of that size.
* Date of last revision (YYMMDD): 101224
* (C) Angel A. Juan - ajuanp(@)gmail.com
**************************************************************************/
public class ElapsedTime
{
    public ElapsedTime(){}

    public static long systemTime()
    {   long time = System.nanoTime();
        return time;
    }

    public static double calcElapsed(long start, long end)
    {   double elapsed = (end - start) / 1.0e+9;
        return elapsed;
    }

    public static String calcElapsedHMS(long start, long end)
    {   String s = "";
        double elapsed = (end - start) / 1.0e+9;
        s = s + calcHMS((int) Math.round(elapsed));
        return s;
    }

    public static String calcHMS(int timeInSeconds)
    {   String s = "";
        int hours, minutes, seconds;
        hours = timeInSeconds / 3600;
        timeInSeconds = timeInSeconds - (hours * 3600);
        minutes = timeInSeconds / 60;
        timeInSeconds = timeInSeconds - (minutes * 60);
        seconds = timeInSeconds;
        s = s + hours + "h " + minutes + "m " + seconds + "s";
        return s;
    }

    public static void doPause(int timeInSeconds)
    {   long t0, t1;
        t0 = System.currentTimeMillis();
        t1 = System.currentTimeMillis() + (timeInSeconds * 1000);
        do
        {   t0 = System.currentTimeMillis();
        } while (t0 < t1);
    }
}
package CrewPairingP;
```

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.*;

/*****************************************************************************
 *                                                                           *
 * Clase encargada de gestionar el programa. Lee los datos del fichero       *
 * test.txt de prueba, trata las condiciones necesarias para generar         *
 * las soluciones al generar los tramos de vuelos con los Legs y muestra     *
 * una salida formateada con los tramos solución                             *
 *                                                                           *
 *                                          CREW PAIRING PROBLEM             *
 *                                      Proyecto Fin de Master - U O C        *
*                                                                            *
 * @author Jesús Manuel Puentes Gutiérrez                                    *
 *                                                                           *
 * @director Angel A. Juan                                                   *
 *                                                                           *
 *****************************************************************************/


/**
 * @author shuli
 *
 */
public class Inicio {

    final static String inputFolder = "inputs";// + File.separator + "TSP";
    final static String fileNameTest = "test500.txt";//fichero de prueba
                                                     test.txt
    final static String outputFolder = "outputs";//directorio de salida
    final static String fileNameOutput = "out500.txt";//fichero con las
                                                        soluciones

    final static int TIEMPO_ESCALA_MIN = 60;    // Tiempo mínimo de escala
                                                      en minutos
    final static int NUM_MAX_SALTOS = 5;
    final static int HORAS_MAX_VUELO = 800;     // Equivale a 8 horas de
                                                      vuelo máximas

      /**
       * Método principal de la aplicación Crew Pairing Problem
       *
       * @param args
       */
      public static void main(String[] args) {
            // TODO Auto-generated method stub
        System.out.println("****  WELCOME TO THIS PROGRAM  ****");
        long programStart = ElapsedTime.systemTime();

        String testsFilePath = inputFolder + File.separator + fileNameTest;
        String outputFilePath = outputFolder + File.separator + fileNameOutput;

        // Comprobamos si existe un archivo de solución anterior y lo borramos
        // si es así. De este modo se generan soluciones limpias cada vez
        String sFichero = outputFilePath;
```

```
        File fichero = new File(sFichero);

          if (fichero.exists()) {
            fichero.delete();
          }

        // Read inputs files and construct the inputs object
        Inputs inputs = InputsManager.getInputs(testsFilePath);

       // Mostramos un listado con todos los legs disponibles que hemos leído de
       // nuestro archivo de datos
      System.out.println("\n--- Lista con Legs disponibles (sin ordenar)
                         ---\n");
      System.out.println(inputs.showLegList());

        ArrayList<Leg> legs = new ArrayList<Leg>();
        ArrayList<Leg> ruta = new ArrayList<Leg>();

        Leg l;
        boolean primero = true;
        String solution = "";
        int numSols = 0;
        int numSaltos = 0;
        int numHorasVuelo = 0;
        int totalHorasVuelo = 0;

        // Genero nueva lista
        for (int i=0; i < inputs.getLegList().length; i++)
            legs.add(inputs.getLegList()[i]);

        // Ordenamos la lista de Legs según la hora de inicio del vuelo
            sortList(legs);

      // Seleccionamos los datos necesarios del primer elemento de la lista
      String destino = legs.get(0).getDestino();
      int horaInicio = legs.get(0).getHoraInicio();
      int horaFin = legs.get(0).getHoraFin();
      String fecha = legs.get(0).getFecha();

        do{
            // Vaciamos la ruta para empezar otro tramo
            ruta.clear();
            // Reiniciamos el contador de tramos
            numSaltos = 1;


            if (primero == true){
                  // Seleccionamos los datos necesarios del primer elemento de
                  la lista
                  destino = legs.get(0).getDestino();
                  horaInicio = legs.get(0).getHoraInicio();
                  horaFin = legs.get(0).getHoraFin();
                  fecha = legs.get(0).getFecha();
                  // Añadimos el primer elemento del tramo (si es el primero
                  utilizado)
                  ruta.add(legs.get(0));
                  // Calculamos las horas de vuelo del tramo
                  numHorasVuelo = calcFlightHours (horaInicio, horaFin);
                  // Sumamos al total de horas de vuelo que tenemos
```

```
                totalHorasVuelo = addFlightHours(totalHorasVuelo,
                            numHorasVuelo);
                // Borramos el elemento usado (si es el primero utilizado)
                legs.remove(0);
                primero = false;
          }


          do {
                // Buscamos un Leg solucion
                      l = searchSol(legs, destino, horaInicio, horaFin, fecha,
                                  numSaltos, totalHorasVuelo);

                if (l != null){
                      // Añadimos el Leg a la ruta
                      ruta.add(l);
                      // Contamos el salto
                      numSaltos++;

                      destino = l.getDestino();
                      horaInicio = l.getHoraInicio();
                      horaFin = l.getHoraFin();
                      fecha = l.getFecha();

                      // Calculamos las horas de vuelo del tramo
                      numHorasVuelo = calcFlightHours (horaInicio, horaFin);
                      // Sumamos al total de horas de vuelo que tenemos
                      totalHorasVuelo = addFlightHours(totalHorasVuelo,
                                        numHorasVuelo);

                      // Borramos el Leg utilizado
                      int indice = legs.indexOf(l);
                      legs.remove(indice);
                } else {
                      // Imprimimos solución en pantalla
                      numSols++;
                      solution = showSol(ruta, numSols, totalHorasVuelo);

                      // Guardamos la solución en un archivo de texto
                      (out.txt)
                      saveSolution(solution, outputFilePath);

                      primero=true;
                      totalHorasVuelo = 0;

                      }

          } while (l != null);  // hasta que no existan coincidencias de Legs
      } while (legs.size() > 0); // hasta que se vacíe la lista de Legs


      /**************************************************************************
      * END OF PROGRAM
      *************************************************************************/
      System.out.println("\n****  END OF PROGRAM  ****");
          long programEnd = ElapsedTime.systemTime();
          System.out.println("Total elapsed time = "
              + ElapsedTime.calcElapsedHMS(programStart, programEnd));
```

```
	}

	/**
	 * Método para ordenar la lista de Legs
	 *
	 * @param legs
	 */
	@SuppressWarnings("unchecked")
	private static void sortList(ArrayList<Leg> legs) {
		Collections.sort(legs);
	}

	/**
	 * Función que busca una solución factible con los condicionantes
	 * requeridos
	 *
	 * @param legs
	 * @param destino
	 * @param horaInicio
	 * @param horaFin
	 * @param fecha
	 * @param numSaltos
	 * @param numHorasVuelo
	 * @return Devuelve un Leg que cumple las condiciones impuestas
	 */
	public static Leg searchSol(ArrayList<Leg> legs, String destino, int
		horaInicio, int horaFin, String fecha, int numSaltos, int
		numHorasVuelo){
		Leg sol = null;
		boolean found = false;
		int horas = 0;
		int suma = 0;

		for (Leg i: legs)
			if ((i.getFecha().equals(fecha)) &&
				(i.getOrigen().equals(destino)) && ((calcDifMin(horaFin,
				i.getHoraInicio())) >= TIEMPO_ESCALA_MIN) &&
				(i.getHoraInicio() > horaInicio) && (numSaltos <
				NUM_MAX_SALTOS) && (numHorasVuelo <= HORAS_MAX_VUELO)) {
				horas = calcFlightHours(i.getHoraInicio(),
							i.getHoraFin());
				suma = suma + horas;
				suma = addFlightHours(suma, numHorasVuelo);
				if ((found == false) && (suma <= HORAS_MAX_VUELO)) {
					sol = i;
					found = true;
				}
			}

		return sol;
	}

	/**
	 * Método que nos muestra la solución con un formato específico de algunas
	 * compañías aéreas
	 *
	 * @param leg
	 * @param numSol
	 * @param hVuelo
```

```
	 * @return Devuelve cadena con la solución
	 */
	public static String showSol(ArrayList<Leg> leg, int numSol, int hVuelo) {
		String sol = "";
		String fechaL = " fecha =                                  ";
		String f = "";
		String horasVuelo = " horas de vuelo realizadas =         ";
		String nLin = " numLineas =        ";
		String tramos = "";
		String horas = "";
		String horaInicio = "";
		String horaFin = "";

		sol +=
		"**********************************************************************
		********\n";
		if (((numSol / 10) > 0) && ((numSol / 10) < 10))
			sol += "**********************   S O L U C I Ó N   " + numSol
			+ "  ***************************\n";
		else if (((numSol / 10) > 9) && ((numSol / 10) < 100))
			sol += "**********************   S O L U C I Ó N   " + numSol
			+ "  ************************\n";
		else if (((numSol / 10) > 99) && ((numSol / 10) < 1000))
			sol += "**********************   S O L U C I Ó N   " + numSol
			+ "  *************************\n";
		else
			sol += "**********************   S O L U C I Ó N   " + numSol
			+ "  ***************************\n";
		sol +=
		"**********************************************************************
		********\n";


		for (int i = 0; i < leg.size(); i++){

			if (f == "") {
				f = leg.get(i).getFecha();
				fechaL += f;
			}

			nLin += leg.get(i).getNumLinea() + "       ";

			if (i == 0){
				tramos += "                " + leg.get(i).getOrigen() + "
				------ " + leg.get(i).getDestino();
				horaInicio = String.format("%04d",
					leg.get(i).getHoraInicio());
				horaFin = String.format("%04d",
					leg.get(i).getHoraFin());
				horas += "                  " + horaInicio + "-" +
					horaFin;
			} else {
				tramos += " ------ " + leg.get(i).getDestino();
				horaInicio = String.format("%04d",
					leg.get(i).getHoraInicio());
				horaFin = String.format("%04d",
					leg.get(i).getHoraFin());
				horas += "  " + horaInicio + "-" + horaFin;
			}
```

```
		}

		minDecimal = ((hVuelo%100) * 100) / 60;
		if (minDecimal < 10) {
			resulMinDec = "0" + minDecimal;
		} else {
			resulMinDec = "" + minDecimal;
		}

		sol += "\n" + fechaL + "\n" + horasVuelo + (hVuelo/100) + " horas y
		" + (hVuelo%100) + " minutos" + " (" + (hVuelo/100) + "." +
		resulMinDec + ")" + "\n\n" + nLin + "\n" + tramos + "\n" + horas +
		"\n\n";

		System.out.println(sol);

		return sol;
	}


	/**
	 * Método que graba en un archivo de texto las soluciones encontradas,
	 * añadiéndolas al archivo cada vez
	 *
	 * @param solution
	 * @param archivo
	 */
	public static void saveSolution(String solution, String archivo) {
		try {
			FileWriter flS = new FileWriter(archivo, true);
			BufferedWriter fS = new BufferedWriter(flS);

			fS.write(solution);
			fS.close();
		}catch(IOException e) {
			System.out.println("Error E/S en fichero escritura");
		}
	}

	/**
	 * Método que calcula las horas de vuelo realizadas en un leg, pasándole
	 * las horas a la que comienza y finaliza dicho leg, devolviendo el
	 * número de horas realizadas como un entero
	 *
	 * @param horaInicio
	 * @param horaFin
	 * @return Devuelve las horas de vuelo realizadas como un entero
	 */
	public static int calcFlightHours(int horaInicio, int horaFin) {
		int minInicio = 0;
		int minFin = 0;
		int horasInicio = 0;
		int horasFin = 0;
		int minTotal = 0;
		int horasTotal = 0;
		int tiempoTotal = 0;
		int minTotalesInicio = 0;
		int minTotalesFin = 0;
```

```
            int tiempoTotalMin = 0;

            minInicio = horaInicio % 100;
            minFin = horaFin % 100;
            horasInicio = horaInicio / 100;
            horasFin = horaFin / 100;

            minTotalesInicio = (horasInicio * 60) + minInicio;
            minTotalesFin = (horasFin * 60) + minFin;

            // Si horaFin < horaInicio es que pasamos de día, por lo que
            // debemos sumar 24 horas (1440 minutos)
            if (horaFin < horaInicio)
                  minTotalesFin = minTotalesFin + 1440;

            tiempoTotalMin = minTotalesFin - minTotalesInicio;
            horasTotal = tiempoTotalMin / 60;
            minTotal = tiempoTotalMin % 60;

            tiempoTotal = (horasTotal * 100) + minTotal;

            return tiempoTotal;
      }

      /**
       * Método que calcula la diferencia de tiempo en minutos que existe entre
       * dos horas pasadas como parámetros
       * @param horaInicial
       * @param horaFinal
       * @return Tiempo existente en minutos entre dos horas dadas
       */
      public static int calcDifMin(int horaInicial, int horaFinal) {
            int minInicio = 0;
            int minFin = 0;
            int horasInicio = 0;
            int horasFin = 0;
            int minTotalesInicio = 0;
            int minTotalesFin = 0;
            int difMinTiempo = 0;

            minInicio = horaInicial % 100;
            minFin = horaFinal % 100;
            horasInicio = horaInicial / 100;
            horasFin = horaFinal / 100;

            minTotalesInicio = (horasInicio * 60) + minInicio;
            minTotalesFin = (horasFin * 60) + minFin;

            difMinTiempo = minTotalesFin - minTotalesInicio;

            return difMinTiempo;
      }

      /**
       * Método que va acumulando las horas de vuelo realizadas y
       * las devuelve como un entero
       *
       * @param totalHorasVuelo
       * @param numHorasVuelo
```

```
	 * @return Devuelve las horas de vuelo acumuladas como un entero
	 */
	public static int addFlightHours(int totalHorasVuelo, int numHorasVuelo) {
		int minTotal = 0;
		int horasTotal = 0;
		int minHV = 0;
		int horasHV = 0;
		int sumaMin = 0;
		int sumaHoras = 0;
		int totalMin = 0;
		int totalH = 0;
		int totalM = 0;
		int totalTiempo = 0;

		minTotal = totalHorasVuelo % 100;
		horasTotal = totalHorasVuelo / 100;
		minHV = numHorasVuelo % 100;
		horasHV = numHorasVuelo / 100;

		sumaMin = minTotal + minHV;
		sumaHoras = (horasTotal * 60) + (horasHV * 60);

		totalMin = sumaMin + sumaHoras;

		totalH = totalMin / 60;
		totalM = totalMin % 60;

		totalTiempo = (totalH * 100) + totalM;

		return totalTiempo;
	}
}

package CrewPairingP;

/*****************************************************************************
 *                                                                           *
 * Clase encargada de contener listado de Legs leído del archivo de datos    *
 *                                                                           *
 *                                          CREW PAIRING PROBLEM             *
 *                                    Proyecto Fin de Master - U O C        *
 *                                                                           *
 * @author Jesús Manuel Puentes Gutiérrez                                    *
 *                                                                           *
 * @director Angel A. Juan                                                   *
 *                                                                           *
 *****************************************************************************/


public class Inputs {
	private Leg[] legList;

	/* Constructor */
	public Inputs(int nLegs) {
		legList = new Leg[nLegs];
	}

	/* getters */
	public Leg[] getLegList() {
```

```
            return legList;
      }

      /* Muestra lista de Legs */
      public String showLegList() {
            String temp="";
            for (Leg l: legList)
                  temp += l.toString();
            return temp;
      }

}

package CrewPairingP;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;

/*****************************************************************************
 *                                                                           *
 * Clase encargada de leer los datos del fichero de datos para generar la    *
 * clase Legs con su correspondiente constructor. Utiliza Scanner para leer  *
 * los datos del fichero                                                     *
 *                                                                           *
 *                                                                           *
 *                                          CREW PAIRING PROBLEM             *
 *                                      Proyecto Fin de Master - U O C        *
*                                                                            *
 * @author Jesús Manuel Puentes Gutiérrez                                    *
 *                                                                           *
 * @director Angel A. Juan                                                   *
 *                                                                           *
 *****************************************************************************/


public class InputsManager {
      public static Inputs getInputs(String legsFilePath) {
            Inputs inputs = null;
        try
        {   // 1. COUNT THE # OF LEGS (# OF LINES IN legsFilePath)
           BufferedReader br = new BufferedReader(new FileReader(legsFilePath));
            String f = null;
            int nLegs = 0;
            while( (f = br.readLine()) != null )
            {   if( f.charAt(0) != '#' )
                    nLegs++;
            }

            // 2. CREATE THE INPUTS OBJECT WITH nLegs
            inputs = new Inputs(nLegs);

            FileReader reader = new FileReader(legsFilePath);
            Scanner in = new Scanner(reader);
            int k = 0;

            while( in.hasNextLine() )
            {   String s = in.next();
```

```
                if( s.charAt(0) == '#' )
                    in.nextLine();
                else
                {   int numLinea = Integer.parseInt(s);
                    String origen = in.next();
                    String destino = in.next();
                    int horaInicio = in.nextInt();
                    int horaFin = in.nextInt();
                    String fecha = in.next();
                   Leg leg = new Leg(numLinea, origen, destino, horaInicio,
                              horaFin, fecha);
                    inputs.getLegList()[k] = leg;
                    k++;
                }
            }
            in.close();

        }
        catch (IOException exception)
        {   System.out.println("Error processing inputs files: " + exception);
        }
        return inputs;
      }
}

package CrewPairingP;

/*****************************************************************************
 *                                                                           *
 * Clase que contiene los datos de los Legs, con sus correspondientes        *
 * Getters y Setters, así como un formato básico de salida toString()        *
 * y un comparador para ordenar los Legs por la hora de inicio del Leg       *
 *                                                                           *
 *                                          CREW PAIRING PROBLEM             *
 *                                     Proyecto Fin de Master - U O C         *
*                                                                            *
 * @author Jesús Manuel Puentes Gutiérrez                                    *
 *                                                                           *
 * @director Angel A. Juan                                                   *
 *                                                                           *
 *****************************************************************************/
@SuppressWarnings("rawtypes")
public class Leg implements Comparable {

      /* Instance fields */
      private int numLinea;
      private String origen;
      private String destino;
      private int horaInicio;
      private int horaFin;
      private String fecha;

      /* Constructor */
      public Leg(int numLinea, String origen, String destino, int horaInicio,
                  int horaFin, String fecha) {
            super();
            this.numLinea = numLinea;
            this.origen = origen;
            this.destino = destino;
```

```
		this.horaInicio = horaInicio;
		this.horaFin = horaFin;
		this.fecha = fecha;
	}

	/* Getters and Setters */
	public int getNumLinea() {
		return numLinea;
	}
	public void setNumLinea(int numLinea) {
		this.numLinea = numLinea;
	}
	public String getOrigen() {
		return origen;
	}
	public void setOrigen(String origen) {
		this.origen = origen;
	}
	public String getDestino() {
		return destino;
	}
	public void setDestino(String destino) {
		this.destino = destino;
	}
	public int getHoraInicio() {
		return horaInicio;
	}
	public void setHoraInicio(int horaInicio) {
		this.horaInicio = horaInicio;
	}
	public int getHoraFin() {
		return horaFin;
	}
	public void setHoraFin(int horaFin) {
		this.horaFin = horaFin;
	}
	public String getFecha() {
		return fecha;
	}
	public void setFecha(String fecha) {
		this.fecha = fecha;
	}

	/**
	 * Método que devuelve una salida formateada de Legs
	 */
	@Override
	public String toString() {
		return "\nLeg [numLinea=" + numLinea + "\n\t\t\t " + origen
				+ " --- " + destino + "\n\t horaInicio=" + horaInicio
				+ ", horaFin=" + horaFin + ", fecha=" + fecha + "]\n\n";
	}

	/**
	 * Método para ordenar objetos según la hora de inicio del Leg
	 */
	public int compareTo(Object o) {
		Leg leg = (Leg)o;
```

```
            if(this.horaInicio == leg.getHoraInicio())
                  return 0;
            else if (this.horaInicio < leg.getHoraInicio())
                  return -1;
            else
                  return 1;
      }
}

package CrewPairingP;

/*****************************************************************************
 *                                                                           *
 * Clase que gestiona un error de ejecución                                  *
 *                                                                           *
 *                                          CREW PAIRING PROBLEM             *
 *                                     Proyecto Fin de Master - U O C       *
 *                                                                           *
 * @author Jesús Manuel Puentes Gutiérrez                                    *
 *                                                                           *
 * @director Angel A. Juan                                                   *
 *                                                                           *
 *****************************************************************************/

@SuppressWarnings("serial")
public class LegException extends RuntimeException{
      LegException(String mensaje){
            super(mensaje);
      }
}
```

- **Datos de entrada de la aplicación:**

```
...
3028  MXP   BCN   1850  1950  03/09/2012
2050  BCN   MAD   2050  2140  03/09/2012
2240  MAD   BCN   2240  2310  03/09/2012
1010  BCN   VLC   1420  1440  03/09/2012
1009  VLC   BCN   1540  1610  03/09/2012
…
```

Ofrecemos una captura parcial de todos los datos utilizados. Para obtener una copia de esta información, se puede acudir a la dirección http://bscw.uoc.es donde se encuentra expuesto de forma pública, previa autorización del director del proyecto y de la universidad.