Trabajo de Fin de Grado

Herramienta para reorganizar vuelos ante imprevistos

Tool to reorder flights with incidents

Sergio Díaz González

La Laguna, 8 de julio de 2015

D. **SALAZAR GONZÁLEZ, JUAN JOSÉ** , con N.I.F. 43.356.435-D profesor Titular de Universidad adscrito al Departamento de Nombre del Departamento de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

““Herramienta para reorganizar vuelos ante imprevistos”

ha sido realizada bajo su dirección por D. **Sergio Díaz González**, con N.I.F. 79.061.612-D.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 8 de julio de 2015

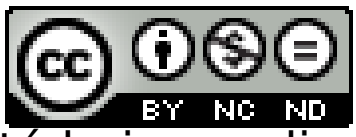
Agradecimientos:

Primeramente quiero agradecerle el apoyo a mi padre, hermana y novia durante estos tres meses haciendo el TFG y por supuesto toda la carrera, ya que siempre han estado ahí cuando he tenido dificultades.

Por otro lado agradecer a todos los profesores que me han dado clase durante estos años, por todo lo enseñado, y también a mi tutor del TFG, Juan José Salazar por el apoyo mostrado en todo momento.

Por último y no menos importante agradecerle a todos y cada uno de mis compañeros de carrera el hecho de ayudarme cuando lo he necesitado y haber compartido tanto los momentos malos como momentos los buenos durante estos cuatro años de carrera.

Licencia



Resumen

El fin de este trabajo ha sido el desarrollo de un programa capaz de recalcular las rutas de los vuelos, de una compañía en concreto, si uno de los vuelos tuviera algún tipo de percance y tuviera que verse retrasada su hora de salida. Programado en C++, he usado Qt Creator para crear una interfaz gráfica sencilla con la que manejar el programa.

El programa hace uso de algoritmos heurísticos para asegurarse la optimización de los cambios introducidos en el horario.

Palabras clave: Aplicación, C++, Qt Creator, Vuelos, Aviones, Tráfico Aéreo, Heurístico.

Abstract

The objective of this work was to develop a program to able to recaculcate flights routes from a company when a flight has had some kind of problem and it has had to be delayed. It was programmed in C++ and I used Qt Creator to create a simple graphical interface that manage the program.

The programm uses heuristic algorithims to ensure the optimization of the changes to the schedule.

Keywords: Application, C++, Qt Creator, Flights, Arcraft, Air Traffic, Heuristic.

Índice general

Capítulo 1
Introducción

Este capítulo contiene la introducción de la memoria.

1.1 Motivación

La motivación del proyecto se basa en tres razones:

1. **Poner en práctica y mejorar mis conocimientos de programación aprendidos durante la carrera.** Durante toda la carrera he aprendido y realizado prácticas en distintos lenguajes de programación. Una vez enfocado mis conocimientos al itinerario de programación web, me parecía un buen momento para retomar y mejorar mi conocimientos en C++ y practicar por primera vez con Qt Creator.

2. **Practicar con algoritmos heurísticos.** Hemos cursado varias asignaturas donde hemos trabajado con diferentes algoritmos de optimización y personalmente me habían gustado mucho. Realizar mi proyecto de fin de carrera sobre un caso real usando algoritmos heurísticos me parecía una buena práctica.

3. **Dar solución a una problemática real.** Y por supuesto ponerme a prueba en un caso real, como es el reorden del organigrama de un día de una compañía aérea. Es un reto poder intentar solucionar un problema del mundo real con una solución eficaz y eficiente.

1.2 Objetivos y requisitos

- Resolver satisfactoriamente el problema creando un algoritmo eficaz y eficiente.

- Hacer uso de algoritmos heurísticos para optimizar el resultado.
- Usar Qt Creator para realizar la interfaz gráfica.

1.3 Alcance del proyecto

Este trabajo de final de grado tiene como alcance la creación de un software que cargue el organigrama de vuelos de un día en una interfaz gráfica, con la que se pueda interactuar para poder retrasar un vuelo si se desea. A continuación de ésto se espera que se genere automáticamente un organigrama nuevo donde se vean solucionados los problemas surgidos por el retraso previo.

Con el software se desea cubrir los si siguientes puntos:

- Visualizar el organigrama de un día de una compañía cualquiera.
- Permitir la manipulación de los vuelos, pudiéndose desplazar el que se desee de su hora original.
- Reorganizar el organigrama después de un cambio y mostrar nuevo organigrama con éstos.

1.4 Destinatarios

El proyecto está destinado a solucionar un caso real de una compañía aérea, especialmente de aquella que no use ningún tipo de software en la actualidad para solucionar este tipo de problemas. Una vez se hallaran compañías específicas interesadas se podrían aplicar características propias de éstas a la hora de mejorar los resultados.

1.5 Tecnologías utilizadas

- C++
- Qt Creator
- Gitbhub

1.6 Github

Para hacer más fácil el acceso al código de la aplicación y a la vez facilitar el trabajo, evitando que posibles problemas con el repositorio local repercutan al proyecto, se ha ido subiendo el trabajo a un repositorio Github:

https://github.com/alu0100696615/TFG

Capítulo 2
Estado del arte

Existen muchos estudios técnicos y herramientas comerciales para la planificación de vuelos con varios meses de antelación. Sin embargo hay poco sobre cómo reestructurar una planificación cuando en algún momento de su desarrollo surge algún imprevisto que obliga a modificar el plan previsto. En la literatura se conoce como “disruption management” y es un problema muy complejo debido a la enorme casuística que pueden suceder, a la alta incerteza de los datos, y al poco tiempo para tomar decisiones. Generalmente las compañías toman decisiones manualmente, especialmente cuando son empresas pequeñas o medianas que no pueden afrontar los costes económicos de herramientas complejas.

Capítulo 3
Funcionalidad

Este capítulo trata sobre el funcionamiento del programa.

Lo primero que un usuario se encuentra al ejecutar el programa es la siguiente pantalla.

Figura 3.1: Primera vista

En ella se puede observar que está creado un organigrama de vuelos, aunque sin nada en su interior. Se ven las opciones de “cerrar” a la derecha debajo y “Archivo” en la barra de menú.

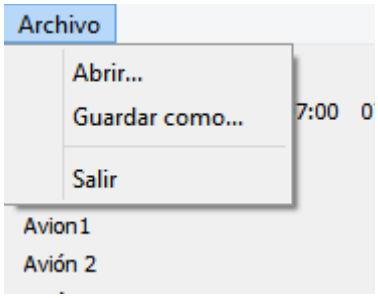


Figura 3.2: Archivo

Si se presiona sobre “Archivo” saldrá un menú vertical con las diferentes opciones que pueden ver en la imagen. Para cargar el fichero de un día cualquiera se presiona sobre “Abrir”.

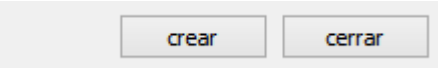


Figura 3.3: Botón crear

Una vez se cargue un fichero que contenga el organigrama en formato .txt de un día, puede observarse, abajo a la derecha junto a cerrar, que la opción “crear” ha sido añadida. Si se presiona en ella se verá el organigrama en el centro de la interfaz tal como se muestra en la siguiente imagen:

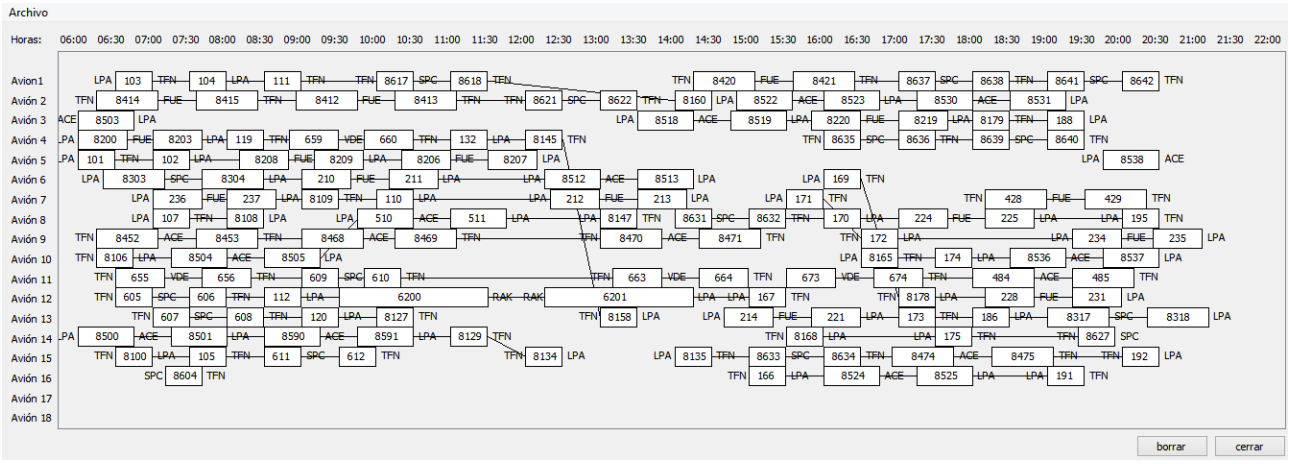


Figura 3.4: Crear organigrama

Los aviones tienen asignados los vuelos que se encuentren en su misma línea. Por ejemplo, el avión 1 tiene asignado los vuelos, desde el 103 hasta el 8642, para ese día en concreto. Las líneas representan las rutas seguidas por los pilotos y como se observa no tienen porque coincidir con la ruta del avión. La duración del vuelo viene marcada por el tamaño del rectángulo. Las siglas a la izquierda de un vuelo se corresponden con el aeropuerto origen de donde sale el vuelo y la de la derecha en el aeropuerto destino donde aterriza.

Además, podemos observar que se ha cambiado la opción de “crear” abajo a la izquierda por la de “borrar”. Si se desea cargar otro organigrama primero habrá que borrar el anterior presionando “borrar”. Una vez pulsado nos volverá aparecer la opción “crear”. Sólo se necesitará abrir el menú vertical y “Abrir” otro fichero. Si no se carga por fichero otro organigrama la opción “crear” cargará el mismo organigrama que se acaba de borrar.

Para retrasar basta con presionar encima del vuelo que se desee y arrastrar a la derecha.

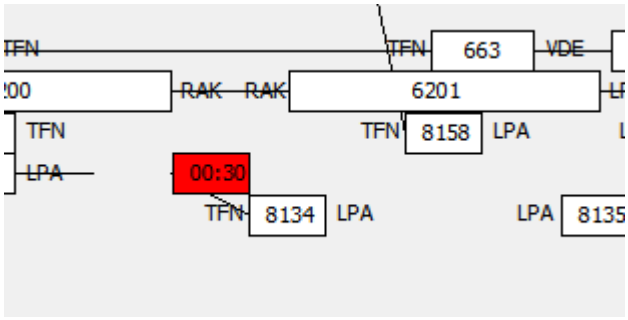


Figura 3.5: Retrasar vuelo

Como se puede observar, una vez se pinche encima de un vuelo, aparte de colorearse de rojo, se colocarla en su interior el tiempo de retraso que conlleva ese movimiento.

Una vez se suelte el elemento el programa, automáticamente, realizará el algoritmo y recargará el organigrama que se ve en el interfaz con los cambios óptimos realizados, resolviendo el problema.

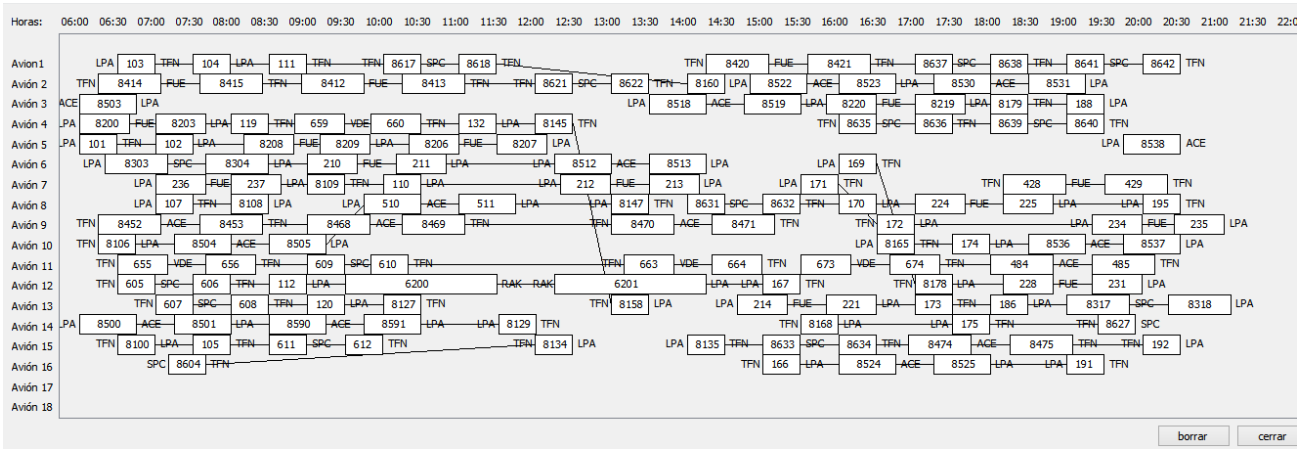


Figura 3.6: Después del retraso

Si se desea guardar este organigrama nuevo generado sólo se necesita ir a “Archivo/Guardar como”.

Capítulo 4
Implementación

En este capítulo se especifica el desarrollo de la aplicación y las distintas funciones y algoritmos usados para llevarla a cabo.

4.1 Elementos de la interfaz

Para llevar a cabo la aplicación lo primero que se planteó fue realizar una interfaz gráfica que pudiera aportar la consistencia necesaria para la aplicación y con la que el usuario pudiera interactuar fácilmente.

Para ello, mediante las distintas clases de Qt Creator, se fueron incorporando elementos con tal de lograrlo. Se puede resumir los elementos usados y su distribución en la siguiente imagen:

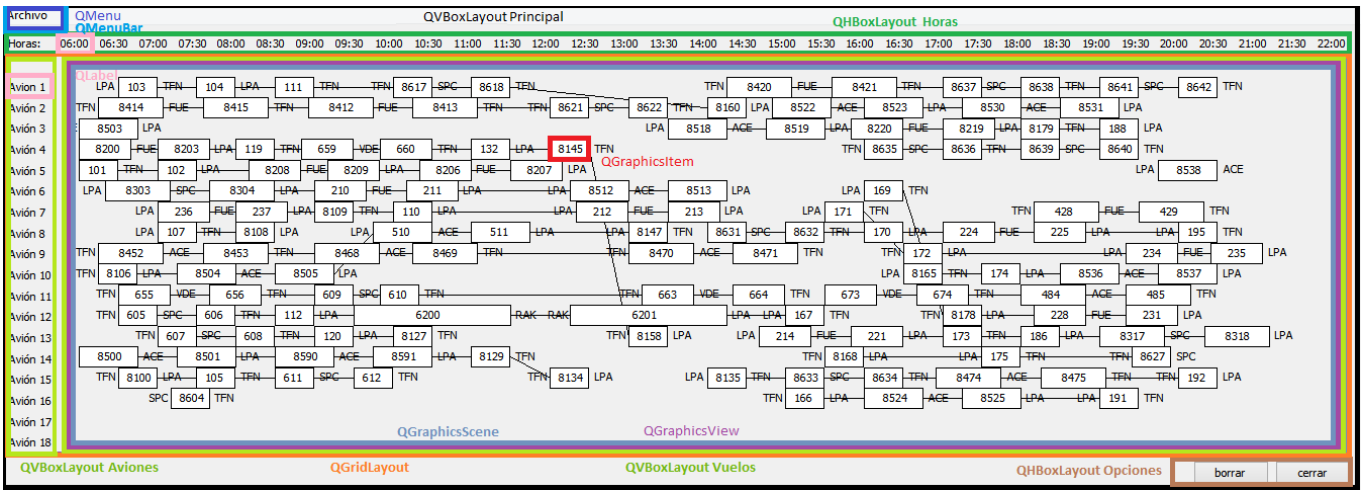


Figura 4.1: Diagrama elementos Qt Creator de la interfaz

QVBoxLayout Principal: Incorpora todo lo demás.

QMenuBar: La barra de menu contiene el menú Archivo(QMenu) donde se encuentran cargar y guardar fichero. Ambos son elementos **QAction**.

QHBoxLayout Horas: Contiene un layout con diferentes **QLabel** con las horas que se ven en pantalla.

QGridLayout: Contiene el los demás elementos. En la posición “0” se encuentra un **QVBoxLayout** con los aviones. Y en la posición “1” otro **QVBoxLayout** pero éste con los vuelos.

QGraphicsView/QGraphicsScene: El elemento en el que se dibujarán los vuelos. Permite la interacción a tiempo real mediante el “press”, “realese” o “move” del ratón. En el se contendrán los rectángulos que definirán los vuelos y las líneas que dictaminan la ruta de los pilotos(**QLine**) y el origen y el destino de los vuelos(**QString**).

QGraphicsItem: En el están dibujados los rectángulos. Éstos se pueden desplazar(move) para retrasar un vuelo pulsándolos(press) , soltándolos(release) donde se desee finalizar el movimiento. Contienen en su interior el nombre del

vuelo(**QString**).

QHBoxLayout Opciones: Contiene las opciones, crear, borrar y cerrar(todas **QPushButton**) que conectan con la función que las maneja.

4.2 Leer datos de fichero

En este apartado hablaremos de la función que permite cargar los datos que usan en las demás funciones.

Antes de nada se crean una serie de variables necesarias para almacenar la información que se va a leer. Éstas son:

- Nombre de los vuelos(QString) array. Guarda los números y las letras identificador de cada vuelo. Tamaño array = número de vuelos.
- Origen del vuelo(QString) (array). Aeropuerto origen del vuelo. Tamaño array = número de vuelos.
- Destino del vuelo(QString) (array). Aeropuerto destino del vuelo. Tamaño array = número de vuelos.
- Hora de partida del vuelo(float) (array). Hora en el que el avión despegará del aeropuerto. Tamaño array = número de vuelos.
- Hora llegada del vuelo(float) (array). Hora en la que avión llegará a su destino. Tamaño array = número de vuelos.
- Pilotos(QString) (array). Ruta de vuelos asignada para los pilotos en una jornada. Tamaño array = número de pilotos.
- Aviones(QString) (array). Ruta de vuelos asignada para los aviones en una jornada. Tamaño array = número de aviones.
- Número de vuelos para ese día(int). El número de vuelos que van a llevarse a cabo durante una jornada.
- Número de vuelos en el archivo(int). El número de vuelos que contiene el fichero. Ésta se usará cuando vaya a sobreescribir el fichero con los nuevos datos.
- Número de pilotos(int). Número de pilotos que intervienen en una jornada.
- Número de aviones(int). Número de aviones que intervienen en una jornada.

Los datos están recogidos en dos ficheros .txt. En el primero se encuentra la información de los vuelos de una semana. En concreto nombre, fecha, origen, destino, hora de partida y hora de llegada. En el segundo se adjunta la ruta llevada por los pilotos y por los aviones durante un día.

A continuación de abrirse el fichero de un día en concreto, se crea un **QFile** donde se guarda todo el contenido de los ficheros .txt (**QtextStream**). Primero se trabaja con los datos del .txt que contiene la información de los vuelos y luego lo propio con el de la información de pilotos y aviones. En el primer caso, lo primero que se realiza es un bucle para ver el número de vuelos que hay programados para el día en concreto que se esté tratando(y de paso también calcula ya el número de vuelos que hay en total en el archivo).

Una vez se sepa el número de vuelos en total, se inicializan los diferentes arrays, asociados con este fichero. Después de inicializar los arrays se recorre el fichero de nuevo, guardando los datos por separado en cada array correspondiente. Los datos de un vuelo tendrán la misma posición en todos los arrays.

Concluido el almacenamiento de esos datos se realiza una tratamiento similar al fichero de los datos del piloto y del vuelo. Primero se recorre el fichero para averiguar el número de pilotos y el número de aviones para a continuación inicializar el array de pilotos y el array de aviones con sus correspondientes datos.

Después de inicializar los mismos se recorre de nuevo para guardar los datos.

4.3 Algoritmos empleados

En este apartado se hablará sobre el funcionamiento del programa. Una vez se interactúe con el programa y se retrase un vuelo lo primero que va a hacer el algoritmo, antes de nada, es comprobar si el retraso afecta o no a otros vuelos. Hay que tener cuenta que el siguiente vuelo del piloto puede ser distinto al del avión, por lo que hay que controlar todos los posibles casos.

Si los vuelos siguientes no son afectados el algoritmos se termina satisfactoriamente. En caso de no ser así entraría en otra función, donde, mediante un algoritmo heurístico, realiza una serie de cálculos para encontrar la forma más óptima(de menor coste) de la que se pueden reorganizar los siguientes vuelos afectados.

Las premisas que se cumplen en todo el alg-oritmo son:

- El avión y el piloto deben estar mínimo 30 min en el aeropuerto de donde saldrá su siguiente vuelo.
- Si el aeropuerto está cerrado no se puede seguir por esa trayectoria

4.3.1 Algoritmo de reordenamiento

Este subapartado tratará sobre como funcionan los distintos algoritmos que han sido programados para que sea posible el reorden de los vuelos afectados por un retraso retraso.

Lo primero que se trató fueron las distintas acciones que se podrían dar para solucionar el problema planteado. Éstas son:

Figura 4.2: Soluciones

1) Retrasar vuelo: La solución que primero se nos vendría a la cabeza sería la de retrasar el vuelo afectado para que no se vea colisionado con el anterior. Esto quiere decir que se calcularía cuanto habría que retrasar el vuelo para que la diferencia entre el predecesor y éste fuera de 30 min. Aparte se tiene en cuenta que el vuelo no va más allá que el cierre del aeropuerto.

Con fin de cumplir esta tarea lo primero que hace el programa es comprobar que el retraso del vuelo no conlleve a que la llegada a su destino fuera más allá del cierre del aeropuerto, ya que siendo así no sería posible retrasar el vuelo. Luego comprueba que el avión y el piloto que se van a retrasar no son los últimos. Si es así y, tanto el vuelo siguiente del avión como el del piloto no colisionan con el actual más el retraso, se llegaría a una ruta optima y si no se comprueba:

- Si el vuelo del avión y del piloto siguiente es el mismo habría

que incluir en la lista abierta las diferentes opciones para solucionar el problema de colisión con el mismo.

- Si piloto y el avión tienen vuelos distintos a continuación del actual y la colisión se produce por culpa del piloto, nada más, se inserta en lista abierta.
- Si piloto y el avión tienen vuelos distintos a continuación del actual y la colisión se produce por culpa del avión, nada más, se inserta en la lista abierta.
- Si piloto y avión tienen vuelos distintos a continuación del actual y se produce una colisión en ambos aviones se introduce en la lista abierta ambos. Al introducirse ambos en la lista abierta se tiene en cuenta que, cuando uno de los dos alcance una trayectoria óptima la otra ya la haya encontrado también, si no se incorporará la trayectoria a la lista cerrada pero el algoritmo continuará. Si las dos han encontrado una trayectoria óptima se suman ambos costes y juntan ambas trayectorias en una que se incluirá en la lista abierta con opciones de ser el camino óptimo final.

Si ninguno de los dos es el último pero no se da ninguna de estas cuatro opciones esta trayectoria muere ahí. Ahora, si alguno de los dos es el último se comprueba:

- Si para los dos es el último vuelo, se incluye en la lista cerrada esta trayectoria porque es óptima y puede retrasarse.
- Si para el piloto es el último vuelo y para el avión no, pero se puede retrasar el vuelo actual sin colisionar con ningún otro, la trayectoria es óptima y se incluye en cerrada.
- Si para el avión es el último vuelo y para el piloto no, pero se puede retrasar el vuelo actual sin colisionar con ningún otro, la trayectoria es óptima y se incluye en cerrada.
- Si para el piloto es último vuelo, pero colisionan los horarios con los del vuelo siguiente del avión, se incluye en lista abierta la ruta.
- Si para el avión es último vuelo, pero colisionan los horarios con los del vuelo siguiente del piloto, se incluye en lista abierta la ruta.

Si no entrara en ninguno de esos casos se daría por imposible seguir por esa trayectoria.

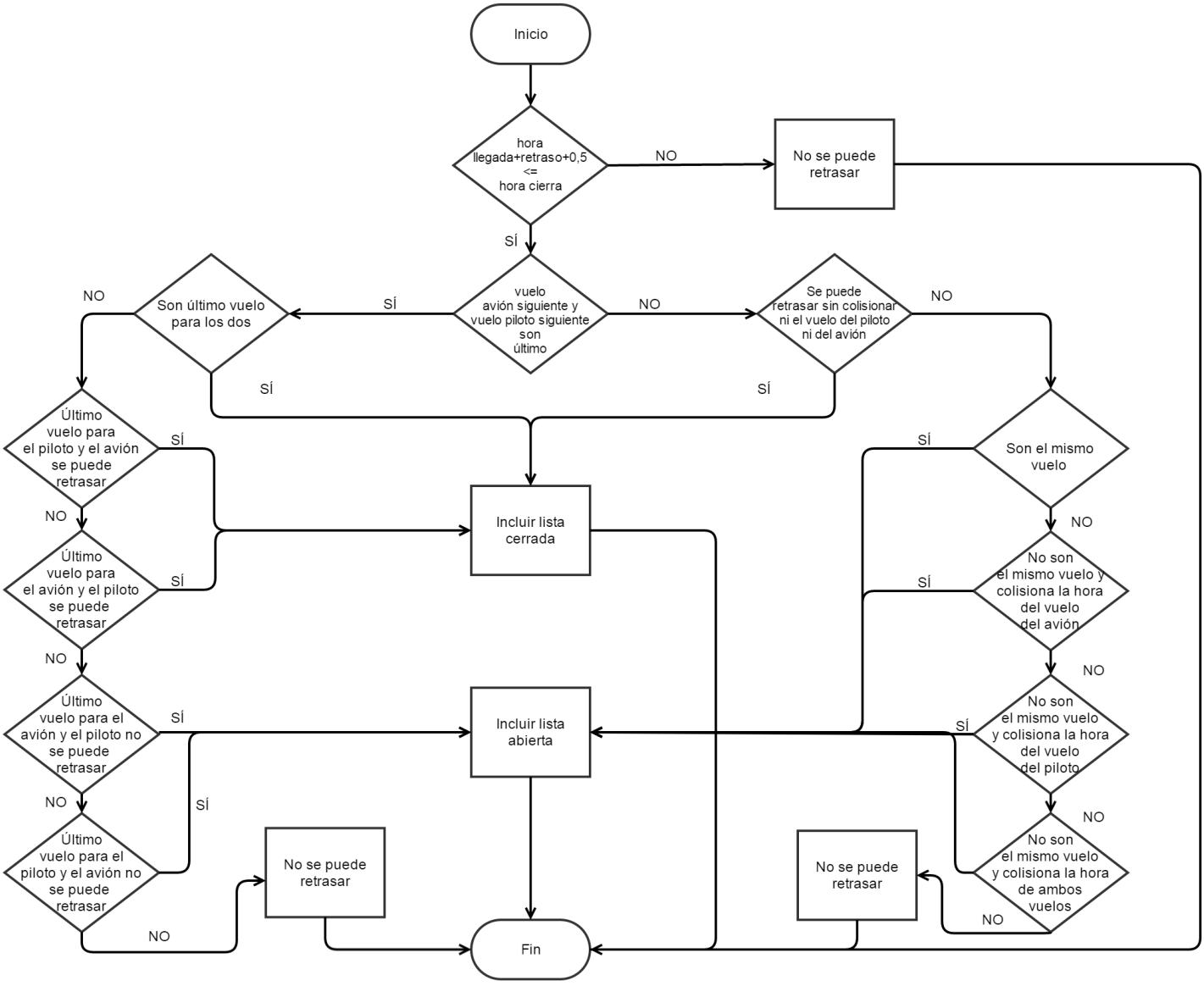


Figura 4.3: Retrasar Vuelo

2) Cambiar piloto: Otra solución es encontrar un piloto que tenga que trabajar ese día y en ese momento no estuviera involucrado en ningún vuelo. Además, si se diera ésto, se comprueba que, si tiene trabajo posterior, pueda regresar al aeropuerto de donde debe salir 30 min antes de su próximo vuelo. Todo esto teniendo en cuenta que cambiando el piloto no es seguro que no sigan colisionando dos vuelos. Ya que cambiando el piloto no se asegura que el avión no vaya a seguir teniendo el mismo problema, cosa que se comprobará antes de incluir esta opción en la lista abierta.

Con este fin se recorre el array de todos los vuelos comprobando solo los vuelos donde sus pilotos no están volando en un avión en ese momento en concreto y se encuentren en el mismo aeropuerto en el que sale el vuelo. Si el piloto en cuestión no tiene más vuelos a continuación se dará por finalizado el reorden, incluyendo este vuelo en la lista cerrada. Si no es así, se comprueba si el piloto puede regresar al aeropuerto donde sale su próximo vuelo. Si no llega a tiempo se incluye en la lista abierta y si llega en la cerrada y se da por concluido el reorden.

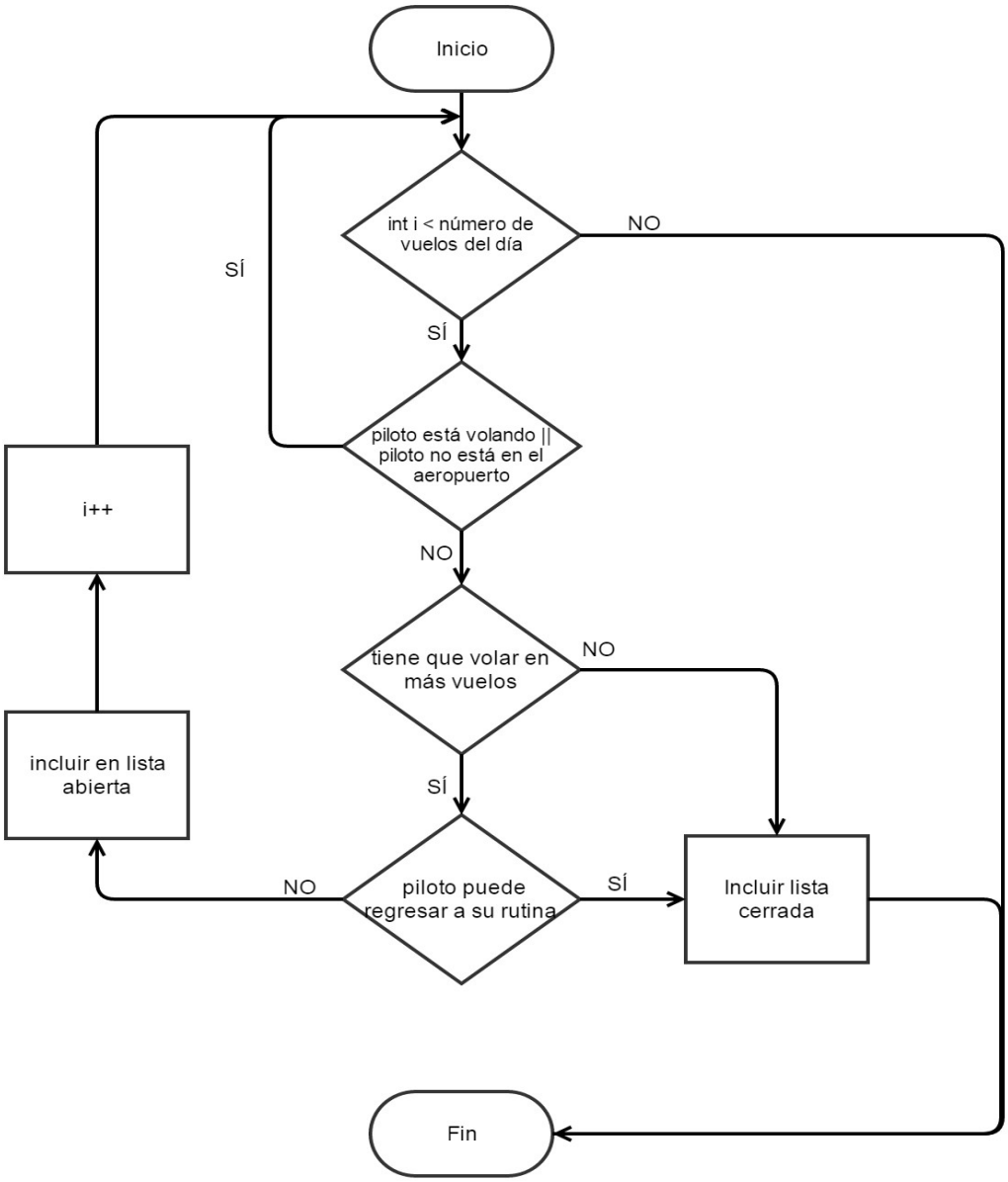


Figura 4.4: Cambiar Piloto

3) Cambiar avión: Consiste en encontrar un avión en uso ese día y que en el momento que se requiriera no estuviera involucrado en ningún vuelo. Además se comprueba que si tiene trabajo posterior pueda regresar al aeropuerto de donde debe salir 30 min antes de su próximo vuelo. Todo esto teniendo en cuenta que cambiando el avión no sigan colisionando dos vuelos todavía. Ya que cambiando el avión no se asegura que el piloto no vaya a seguir teniendo el mismo problema, cosa que se comprobará antes de incluir esta opción en la lista abierta.

Con este fin se recorre el array de todos los vuelos comprobando solo los aviones que no estén volando en involucrados en un vuelo en ese momento concreto y se encuentren en el mismo aeropuerto en donde sale el vuelo. Si el avión en cuestión no tiene más vuelos a continuación, se dará por finalizado el reorden, incluyendo esta trayectoria en la lista cerrada. Si no es así, se comprueba si el avión puede regresar al aeropuerto donde sale su próximo vuelo. Si no llega a tiempo se incluye en la lista abierta y si llega en la cerrada y se da por concluido el reorden.

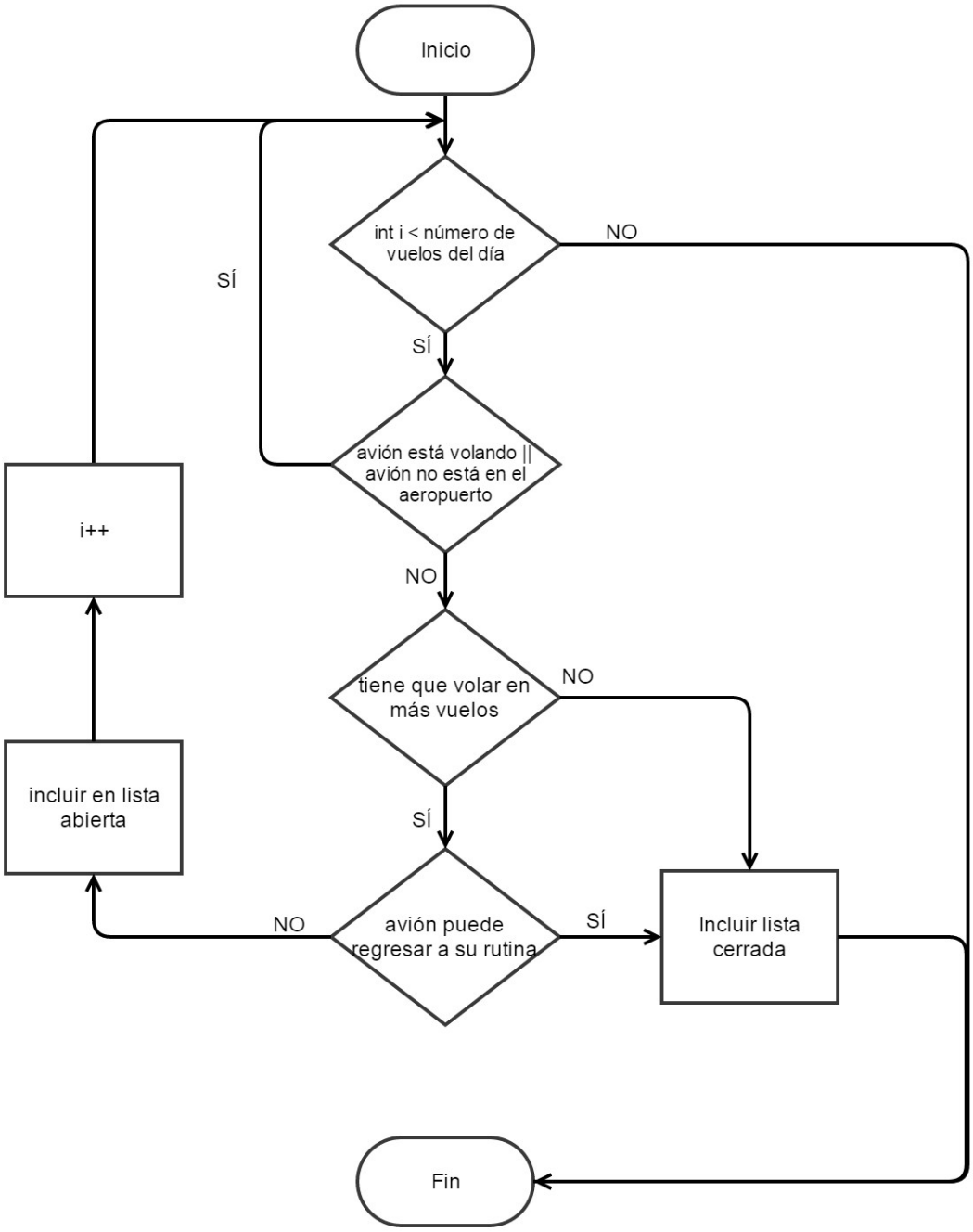


Figura 4.5: Cambiar Avión

4) Cambio de piloto & avión: Cambiar el piloto y el avión a la vez. Se busca un piloto que esté trabajando con un avión los cuales se puedan hacer cargo de la ruta siguiente afectada, siempre y cuando pueda volver luego, con 30 min de antelación, a su siguiente vuelo.

Con este fin se recorre el array de todos los vuelos comprobando

si los pilotos y aviones están en el mismo aeropuerto de donde sale el vuelo a sustituir y no tengan trabajo durante la duración del viaje. Si el avión y el piloto en cuestión no tienen más vuelos a continuación se dará por finalizado el reorden, incluyendo este vuelo en la lista cerrada. Si no es así se pueden dar múltiples combinaciones:

- El piloto ya no tiene más vuelos a continuación pero el avión sí, pero no colisionan.
- El avión ya no tiene más vuelos a continuación pero el piloto sí, pero no colisionan.
- Piloto y avión vuelan juntos en su siguiente trabajo y no colisionan.
- Piloto y avión no vuelan juntos en su siguiente trabajo pero no colisionan.

En todos estos casos se comprueba que el piloto y/o avión afectados puedan regresar a su destino a continuación. Siendo así se encontrará una trayectoria óptima. Si no se da ningún caso se dará por finalizada esta trayectoria.

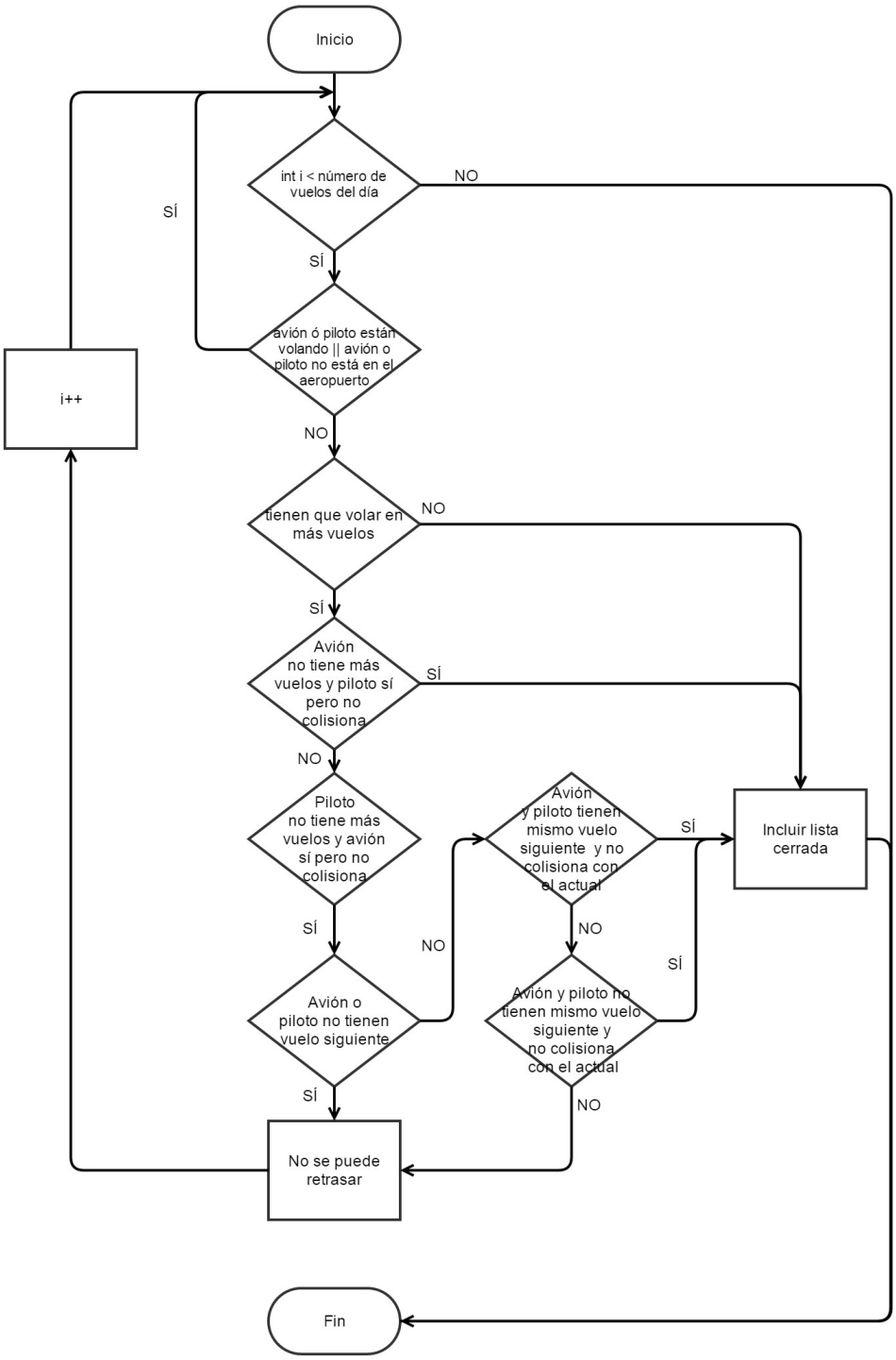


Figura 4.6: Cambiar piloto y avión

5) Cancelar el vuelo: Es la solución más drástica pero también se

puede dar. Cancelar el vuelo y así no se vería afectado el resto del organigrama. Sólo se puede dar en el primer vuelo, ya que de resto saldría más caro.

Cada una de éstas opciones tendría un coste diferente:

TIPO	**COSTE**
Retrasar vuelo	Variable*
Cambio Piloto	Coste bajo
Cambio Avión	Coste medio-alto
Cambio Piloto-Avión	Suma de los dos costes por separado
Cancelar vuelo	Coste muy alto

Tabla 4.1: Coste de cada tipo

*El coste depende del tiempo del retraso. La formula que uso es: CosteRetrasarVuelo=Coste Medio*tiempo que se tarda.

4.3.2 Algoritmo heurístico

En este subapartado se explicará el algoritmo heurístico que se ha usado para optimizar la elección de la trayectoria.

1. Se forma una lista de trayectorias parciales, ABIERTA, con una trayectoria inicial que comienza en el nodo raíz (el retraso del vuelo nodo). También se forma una lista CERRADA, de trayectorias desechadas mínimas, inicialmente vacía.
2. Hasta que la lista ABIERTA esté vacía o se encuentre el objetivo, se analiza su primera trayectoria:
    1. Si la trayectoria termina en un nodo donde ningún vuelo más se vea afectado, se finaliza el bucle.
    2. Si la trayectoria no termina en un nodo donde se vea finalizado el bucle:
        1. Se elimina la primera trayectoria de la lista ABIERTA, incluyéndola en la lista CERRADA. En el caso de que ya

exista una similar, se elimina la de mayor coste.

2. Se forman nuevas trayectorias a partir de la trayectoria eliminada de ABIERTA ramificando el último nodo de la misma. Estas cuatro nuevas trayectorias se corresponderán con: retrasar vuelo, cambio piloto, cambio avión y cambio de piloto & avión. Aunque no siempre. Se comprueba que cambio de piloto y cambio de avión se pueden aplicar o si no no se incluyen en la lista. De igual forma se comprueba que si se hace cambio de piloto & avión, el avión y el piloto anterior, suponiendo que esta trayectoria puede ser óptima, tengan más vuelos vinculados a parte del actual y sean capaces de llegar 30 min antes al aeropuerto correspondiente para acometerlos. Si no, no se incluye cambio de piloto & vuelo. Si se incluye, o cambio de piloto o cambio de avión, como es comprensible, ya no se incluye cambio de piloto & vuelo, pues si se le puede aplicar una por separado no tiene sentido que se le apliquen las dos a la vez juntas.

   Cancelar solo se tiene en cuenta en el primero de los casos, ya que cancelar cualquier vuelo posterior sería más caro siempre que cancelar solo el primero y finalizar ahí.

3. Se añaden las nuevas trayectorias a la lista ABIERTA, si existen.

Se ordena la lista ABIERTA en base al costo total estimado de cada una, colocando la de mínimo coste al inicio de la lista.

4. Si dos o más trayectorias de ABIERTA acaban en un nodo común, se borran las mismas excepto la que posee mínimo coste entre ellas. Se elimina esta última también si existe una similar con menor coste en la lista CERRADA. Al eliminar trayectorias de ABIERTA deben insertarse en CERRADA, salvo que ya exista allí una similar de menor coste.

3. Si se alcanza el nodo objetivo, es decir no se ven afectados más vuelos, el problema tiene solución y se determina la trayectoria óptima, en caso contrario no existe solución.

La función de coste total estimado f*(n) para cada nodo n, viene dada por: **f*(n) = g(n) + h*(n)**

g(n): coste acumulado desde la raíz hasta el nodo n

h*(n): estimación del coste h(n) desde el nodo n hasta el objetivo o coste restante.

4.3.3 Mejora algoritmo heurístico: pila binaria

Para mejorar el rendimiento del algoritmo heurísitico se ha usado una pila binaria. Con esto al insertar o borrar en una lista el número de comparaciones se reduce notablemente, con lo que el programa ganará mucha velocidad, 2 o 3 veces más rápido que otros tipos, como pueden ser los selection sorts o los quick sorts.

Todo ello en una matriz unidimensional. Las posiciones hijas de un elemento siempre van a ser la posición *2 o la posicion *2+1. Si se quiere insertar solo se tiene que incluir en la última posición de la matriz el nodo que se quiera añadir y realizar comparaciones de forma ascendente con los padres de esa posición (esto se consigue dividiendo la posición actual entre dos). Si el coste del elemento es menor que el del padre se intercambia el nodo hijo y el padre, así hasta que el padre sea menor que el hijo y se termina el bucle.

Ejemplo:

Supongamos que añadimos un elemento con un coste de 17 a nuestra pila. En este momento tenemos 7 elementos en ella, así que el nuevo elemento debería situarse en la posición número 8. Así queda la pila; el nuevo elemento está subrayado:

10 30 20 34 38 30 24 <u>17</u>

Luego se compara este elemento con su padre, el cual esta en la posición 8/2 = posición 4. El valor F del elemento que ahora mismo está en la posición 4 es 34. Ya que 17 es menor que 34 se intercambian sus respectivas posiciones.

10 30 20 <u>17</u> 38 30 24 34

Después se compara con su nuevo padre. Como está en la posición 4 se compara con el elemento en la posición 2 (4/2=2). Ese elemento tiene una puntuación F de 30. Ya que 17 es menor que 30, y ahora la pila sería así:

10 <u>17</u> 20 30 38 30 24 34

Para borrar de la lista abierta un elemento se lleva a cabo el mismo procedimiento pero de manera inversa. El elemento a eliminar siempre va a ser la primera posición de la matriz, por lo

que se iguala ésta con el último nodo eliminando este nodo a continuación de la última posición de la lista. Luego se va comparando, de forma descendente, hasta encontrar la posición adecuada, que no tiene porque ser de nuevo la última posición de la lista nuevamente.

Tomando como ejemplo la pila anterior, sacando la posición 10 de la pila, ahora se tiene que el que antes fue el último elemento de la pila ahora está subrayado en primera posición:

34 17 20 30 38 30 24

Lo siguiente es comparar el elemento con cada uno de sus hijos, los cuales están en las posiciones (posición actual x 2) y (posición actual x 2+1). Si tiene una puntuación menor que sus 2 hijos, se queda donde está. Si no, se intercambia su posición con el menor de sus hijos. Así que, en este caso, los dos hijos del elemento en la posición 1, están en la posición 1x2=2 y 1x2+1=3. Ya que 34 no es menor que sus hijos se intercambia por el menor de ellos, en este caso el 17. La cosa queda así:

17 34 20 30 38 30 24

Ahora se compara el elemento con sus 2 hijos, que están en la posición 2x2=4, y 2x2+1=5. Como 34 no es menor que sus hijos se intercambia por el menor (el 30 en la posición 4). Ahora sale esto:

17 30 20 34 38 30 24

Sin embargo para encontrar un elemento determinado en la pila no hay otra forma que hacerlo comparando uno a uno los elementos de la pila.

4.3.4 Otras funciones que uso

Otras funciones que se usan para poder llevar a cabo todo lo los distintos algoritmos son:

- int PilotoPuedeVolver(int posicionActual, int posicion Siguiente). Comprueba que un piloto es capaz de llegar de un aeropuerto a otro antes de que empiece su próximo trabajo. Para ello necesita viajar en un vuelo que vaya a su destino. Recorre todos los vuelos buscando un que pueda coger a determinada hora y le permita llegar a tiempo.
- int AviónPuedeVolver(int posicionASustituir, int posicion Siguiente). Comprueba que un avión es capaz de llegar de un aeropuerto a otro antes de que empiece su próximo trabajo. Para ello, calculando la duración del vuelo, se estima si llega a tiempo o no a su trabajo.

- QString horaAString(float hora). Convierte una hora float en QString para, por ejemplo, sobrescribir un fichero.
- float horaAFloat(QString hora). Convierte una hora QString en float para poder tratar con los datos en las distintas funciones
- int busquedaPilotoArray(QString nombreVueloLista). Busca la posición del vuelo de un piloto en la lista de todos los vuelos del día.
- int busquedaAvionArray(QString nombreVueloLista). Busca la posición del vuelo de un avión en la lista de todos los vuelos del día.
- int masVuelosAfectadosRetraso(int poicionAcual) Comprueba , antes de cambiar piloto y/o avión, que el retraso que se lleve no sea lo suficientemente largo como para que haya afectado a más de uno de los vuelos previsto. Si fuera así, el algoritmo no podría darse por finalizado aunque se encontrara un avión y/o piloto disponible para sustituir este vuelo.
- QstringList buscarOtraRutaSeguida(). Se usa en el caso de que un problema haya afectado a dos vuelos distintos a la vez, se hayan generado dos trayectorias distintas(pero que al final tendrán que sumar su coste) y las dos trayectorias hayan encontrado su nodo objetivo. Cuando se encuentre el camino óptimo para la segunda trayectoria, se sabrá la ruta seguida por ésta pero en cambio no la de la primera. Esta función devuelve la ruta seguida por la primera trayectoria óptima.

4.3.5 Más variables a usar en estos algoritmos

Además de los variables de tipo array donde antes se almacenaron los datos, los distintos algoritmos hacen también uso de las siguientes variables.

- CaminosComprobados(int) (array). Cada vez que se inserta un elemento a una lista realizo un “caminosComprobados++”. Ese va a ser el identificador de la ruta que he metido en la lista.
- NúmeroElementosListaAbierta(int) (array). Número de elementos que en un momento concreto están en la lista abierta.
- NúmeroElementosListaCerrada(int) (array). Número de

elementos que en un momento concreto están en la lista cerrada.

- ListaAbierta (int) (array). Es el array donde se incluye las rutas que aún no se han comprobado.
- ListaCerrada(int) (array). Es el array donde se incluye las rutas que ya se han comprobado.
- NombresVueloLista(QString) (array). Para facilitar y no estar buscando en todo momento el nombre de los vuelos en el array en el que se guardan todos los vuelos del día, se guarda el nombre del vuelo que va a entrar en la lista aquí.
- CostF(int) (array). El coste acumulado de las rutas que están en listas.
- TipoOperacion(int) array. El único con un tamaño distinto(tamaño = 6) y unas variables definidas antes de empezar. Es el coste numérico para cada una de las operaciones:
  - [0]=Retrasar=6
  - [1]=Cambio piloto & avión=17
  - [2]=Cambio piloto=2
  - [3]=Cambio avión=15
  - [4]=Cancelar vuelo=100
  - [5]=Ruta que no se puede seguir=100000
- VuelosOperación(int) (array). La operación que tiene realizar o que realiza este elemento de la lista.
- RutaSeguida(QStringList) (array). Contiene la ruta seguida para llegar hasta el punto en el que está. Cada fila del QStringList tiene el nombre del vuelo por el que paso y la operación que realizo.
- Retraso(int). El retraso que tiene el primer vuelo en entrar a la lista
- RetrasoIndividual(int) (array). El retraso recalculado para cada caso que lleva, respecto a su posición original, el elemento de la lista.
- ListaDondeEsta(bool) (array). Devuelve si un elemento concreto está en un momento determinado en la Lista Abierta(1) o a Lista Cerrada(0).

- DependeDeOtraTrayectoria(bool) (array). Devuelve si en algún nodo la trayectoria nueva afecta a dos vuelos a la vez. 0 para no y 1 para sí.
- PrimeraVueloAfectadoPorDosTrayectorias(int) (array). El primer vuelo cuya trayectoria aplicada afectaba a dos vuelos a la vez.
- OtraTrayectoriaAfectada(int) (array). El primer vuelo de la otra rama, a la que está ligada esta trayectoria, que fue afectada.

El tamaño de los arrays(menos en el caso del TipoOperacion) será siempre el mismo:

Tamaño=número de vuelos en el día * 3 +1

*3 ya que es el máximo número tipos distintos de reorden que va a sufrir un vuelo: a todos se les va a poder aplicar “retraso” pero si se les aplica “cambio de avión & piloto” ya no se les va a aplicar “cambio avión” y “cambio piloto” por separado y viceversa.

+1 porque la operación “cancelar” solo se va a poder aplicar en el primer elemento retrasado.

Cada vez que se inserte un elemento nuevo se inicializará, según su identificador (caminosComprobados) el nombreVueloLista, vueloOperación, rutaSeguida, listaDondeEstá y costF.

4.4 Reescritura

En este apartado hablaremos de las acciones llevadas a cabo después de reordenar el organigrama.

Ahora, con la ruta seguida por la trayectoria que ha llegado a un nodo objetivo en la que no colisionen más vuelos, es momento de reescribir los datos que han cambiado.

Se va recorriendo la lista de las rutas seguidas y según el tipo que sea se acomete una operación u otra:

1)Retrasar vuelo: Se busca el vuelo y se retrasa la hora sumando la hora actual más el retraso estimado para ese nodo. Con estos datos cambiamos a continuación el archivo de vuelos, fechas y horas.

2) Borrar piloto & avión: Se busca en la lista de aviones y pilotos y se borra el vuelo en el avión afectado y, a continuación, en el piloto afectado.

3) Borrar piloto: Se busca en la lista de aviones y pilotos y se borra el vuelo en el piloto afectado.

4) Borrar avión: Se busca en la lista de aviones y pilotos y se borra el vuelo en el avión afectado.

5) Cancelar vuelo: Se busca y se borra en el archivo de aviones y pilotos el vuelo y también se busca y se borra de la lista de vuelos, fechas y horas

5) Piloto & avión a introducir: Se introduce en el archivo de aviones y pilotos que el piloto en cuestión se va a hacer cargo del vuelo e igual con el avión.

6) Piloto a introducir: Se introduce en el archivo de aviones y pilotos que el piloto en cuestión se va a hacer cargo del vuelo.

7) Avión a introducir: Se introduce en el archivo de aviones y pilotos que el avión en cuestión se va a hacer cargo del vuelo.

Una vez se cambian los datos se recarga la interfaz automáticamente y se ve el nuevo organigrama generado.

4.5 Almacenamiento de datos en un fichero

Después de introducir el nombre del archivo en el que se va a guardar el fichero, el proceso de almacenamiento es el mismo que la carga. Se van leyendo los datos guardados en las distintas variables(Nombre del vuelo, Origen del vuelo, Destino del vuelo, Hora de despegue, Hora de llegada, Pilotos, Aviones) y se va incluyendo en el fichero a guardar.

Capítulo 5
Estructuración archivos

Para realizar este proyecto se han organizado las funciones en ficheros de la siguiente forma:

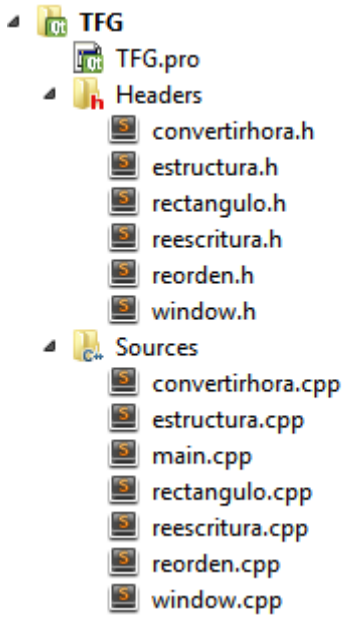


Figura 5.1: Ficheros

Main. Es donde se ejecuta el programa. Invoca a **Window**.

Window. Es la principal del programa. En ella se leen los datos, invoca a Estructura e invoca a Reorden. Hereda de la clase **ConvertirTiempo** funciones para convertir la hora en float. Se encuentra la barra de menú y las opciones de la esquina derecha.

Estructura. En ella se crea la estructura de la interfaz. Se crea un QGraphicsScene dentro de un QGraphicsView dónde se invocará a un objeto de la clase **Rectángulo**.

Rectangulo. Hereda de la clase QGraphicsItem y es donde se crear los recuadros de los vuelos que tienen capacidad de movimiento según interaccionemos con el ratón con ellos. Una vez se cambia la posición retrasando un elemento invoca a la clase **Window** para que esta invoque al **Reorden**.

Reorden. En ella están los algoritmos de reordenamiento y los heurísticos en los que se apoya. También se encuentran todas las funciones necesarias para llevar a cabo la reorganización. En ella se invoca a **Reescritura** una vez se tenga solucionado el problema.

Reescritura. En ella se encuentran los algoritmos que reescriben las variables y los ficheros con los nuevos datos aportados después del reorden. Hereda de la clase **ConvertirTiempo** funciones para convertir la hora float en Qstring.

Capítulo 6
Conclusiones y líneas futuras

En este capítulo se expondrán las conclusiones y se hablará sobre distintas líneas futuras a seguir.

En las fases iniciales de este proyecto se definieron una serie de objetivos y requisitos necesarios que he ido cumpliendo. Ahora es momento de repasar esos objetivos e ir verificando su cumplimiento.

El primero de ellos era poner en práctica las lecciones de programación en C++ recibidas durante la carrera en un proyecto grande y complejo que me permitiera repasarlas y mejorar. Hacía tiempo que no programaba en este lenguaje en concreto.

Segundo, aprender a desenvolverme con un framework que me permitiera crear interfaces gráficas como es Qt Creator . Manejarme con sus distintas funcionalidades ha sido un reto pues nunca había usado nada parecido.

Tercero, poner a prueba lo aprendido en la carrera sobre algoritmo de búsquedas y algoritmos heurísticos. Durante la carrera hemos dado diferentes algoritmos pero el adaptar lo visto, a las funcionalidades que requería el programa, ha sido una experiencia grata a la hora de poder retarme a mí mismo y aprender a manejarme.

La aplicación por supuesto se puede mejorar y hacerla más completa. Por ejemplo se puede:

- Añadir más restricciones para hacerla más específica y más eficiente.
- Mejorar la interacción con la interfaz añadiendo los datos de los pilotos y de los aviones específicos para hacerlo más próxima y cercana.

En cuanto a la valoración personal la verdad he tenido que

dedicarle muchas horas, ya no sólo a realizar la aplicación, si no también al aprendizaje para poder exponer las ideas que tenía en mente y cumplir los requisitos satisfactóriamente. En conclusión estoy muy contento de mi labor desempeñada, los conocimientos adquiridos y, por supuesto, del resultado final.

Capítulo 7
Summary and Conclusions

In this chapter I'm going to speak about my conclussions and the posible future research

First at all, it was defined a several objetives and requirements which have been fulfilled. Now it's time to review those objetivos verify its compliance.

The most important objetive was to implement the lessons of programming received in the degree in a large and complex project which would allow me to review them and improve. In addition, I hadn't programmed in this languague, c++, in a much time.

Second, to learn to manage a framework which would allow me to create prahical interfaces such as Qt Creator. Managing different functionalities has been challenging to me becuase I had never used anything like this.

Third, testing all that I learnt in the degree about searches and heuristic algorithms. We have learnt many algorithms during the degree but aplying them in my project wasn't easy. However, challenge myself has been a great experience.

The application can be improved, of course. For example:

- Add more restrictions to make it more specific and more efficient.
- Improve interaction with the interface data adding pilots and specific aircraft to do better it.

In conclusion, my personal assessment is good, though I had to spend many hours, not only to make the application but to learn and develop my ideas to pass requeriments satisfactorily. But, in conclusion, I'm very happy with my work, the knowledge learnt and, of course, the result of the application.

Capítulo 8
Presupuesto

En este capítulo se explica de manera detallada los gastos que tendrá el desarrollo e implantación de la aplicación “Herramienta para reorganizar vuelos ante imprevistos”.

Dado que este proyecto requiere alrededor de tres meses para llevarlo a cabo, se expondrá de forma precisa el precio estimado para cada elemento y el tiempo.

A continuación, se detalla dicho presupuesto en diferentes apartados:

8.1 Presupuesto de mobiliario

Para la creación de la aplicación se necesita de cierto mobiliario imprescindible. Lógicamente un ordenador en donde poder trabajar. Además de un ratón, una silla y una mesa, utensilios básicos sin los que no se podría llevar a cabo el trabajo..

Concepto	Unidad	Precio/Unidad	Total
Mesa	1	50 €	50 €
Silla	1	20 €	20 €
Ordenador	1	1000 €	1000 €
Ratón	1	50 €	50 €

Tabla 8.1: Presupuesto mobiliario

8.2 Presupuesto en lugar de trabajo y derivados a éste

En este apartado se hablará de los gastos necesarios para proporcionar a alguien cualquier espacio de trabajo: gastos de alquiler, electricidad, internet y agua.

Por una parte, el proyecto es llevado a cabo por una persona que carece de medios económicos suficientes para poder pagar un alquiler normal, así se ha recurrido a un coworking que minimice los costes de alquiler, así los gastos de electricidad o agua.

Por otro parte, para que el desarrollo de dicha aplicación se lleve a cabo de manera eficaz y eficiente es obvio que se necesita conexión a intrnet, para poder consultar cualquier tipo de duda y poder dar solución a errores y problemas que vayan surgiendo.

Concepto	Meses	Precio/Mes	Total
Alquiler	3	200 €	600 €
Electricidad	3	20 €	60 €
Internet	3	20 €	60 €

Tabla 8.2: Presupuesto en suministros y similares

8.3 Presupuesto en salario

Esta parte del presupuesto corresponderá al salario asignado al trabajador. En este caso un titulado en Ingeniería Informática por la Universidad de La Laguna que cobrará según de las horas dedicadas al proyecto entorno 320 horas.

Empleado	Categoría	Horas	Euros/Hora	Total
Sergio Díaz González	Ingeniero Informático	320	15	4800 €

Tabla 8.3: Presupuesto en suministros y similares

8.4 Presupuesto final

En este último apartado se detallara el gasto total que va a suponer la puesta en marcha y finalización de este proyecto. Se obtendrá por la suma del “presupuesto en mobiliario” con el “presupuesto en el lugar de trabajo y derivados a éste” y el “presupuesto en salario”. El resultante es el coste que va a tener la aplicación en su conjunto.

Descripción	**Total**
Presupuesto en mobiliario	1070 €
Presupuesto en el local de trabajo y derivados	720 €
Presupuesto en salarios	4800 €
Total	6590 €

Tabla 8.4: Presupuesto Final

Por lo tanto, el proyecto tendrá un gasto total de 6590 €

Bibliografía

- Apuntes asignatura Inteligencia Artificial, Optimización, Estructura de Datos, Estructura de Datos Avanzados, etc.
- http://www.policyalmanac.org/games/articulo4.htm