Trabajo de Fin de Grado

Procedimiento informático para la planificación óptima de tripulaciones y aviones que cubran unos vuelos dados

Computer Science Approach for the optimal planning of crews and aircrafts to serve a given set of flights

Eduardo Pérez Hernández

La Laguna, 11 de septiembre de 2020

D. **Juan José Salazar González**, NIF 43.356.435-D, profesor Catedrático de Universidad adscrito al Departamento de Matemáticas, Estadística e Investigación Operativa de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

”*Procedimiento informático para la planificacion óptima de tripulaciones y aviones que cubran unos vuelos dados*”

ha sido realizada bajo su dirección por D. **Eduardo Pérez Hernández**, NIF 54.114.094-Q.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 11 de septiembre de 2020

Agradecimientos

A mis padres, por su incondicional cariño y apoyo
Al profesor Juan José Salazar, por darme la oportunidad de entrar en el apasionante mundo de la Logística

Licencia

Resumen

En este trabajo se explora la resolución de un problema de diseño de rutas que nace en el contexto de una aerolínea en Canarias. En este sentido se busca dar solución a los problemas concretos de asignación de flotas, enrutamiento de aviones y emparejamiento de tripulaciones cubriendo los vuelos de la aerolínea en un único día. Se propone una solución al problema mediante un algoritmo heurístico basado en un modelo de programación lineal entera (MILP), codificado en el lenguaje Python, así como de la librería de otimización `Pulp`*. Se aportan comparativas de rendimiento del algoritmo al someterlo a diferentes configuraciones iniciales.*

Palabras clave: Optimización, MILP, Algoritmia, Heurística, Python, Pulp

Abstract

This project explores the resolution of a routing problem applied to the context of a airline company in Canary Islands. For that matter, we seek to solve the concrete fleet assignment, aircraft routing and crew pairing problems covering the airline flights of a single day. We propound a solution to the problem using an heuristic algorithm bases on a integer programming model (MILP), coded in Python, and the Pulp optimization library written for that language. We also bring a performance analysis of the algorithm with different sets of initial configurations.

Keywords: Optimization, MILP, Algorithmics, Heuristics, Python, Pulp

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Breve introducción

Las compañías aéreas necesitan formas de resolver los complejos problemas logísticos inherentes al ejercicio de su profesión. Los recursos con los que trabajan, es decir, aviones y tripulaciones, tienen un costo muy alto, por lo que necesitan hacer uso de ellos de la manera más óptima posible. Debido a motivos técnicos o a requerimientos de la compañía y los sindicatos de trabajadores se introduce una gran complejidad al problema logístico. Es por ello que en la mayoría de ocasiones la planificación de los dos recursos se realiza por separado. No obstante, con este enfoque surge el problema de que al resolver cada problema por separado podría desembocar en una solución subóptima, pero por otro lado integrar los dos problemas en un solo modelo podría llevarnos a un problema cuya complejidad impida que lo podamos resolver en un tiempo razonable. Por suerte, dependiendo de las reglas del problema concreto al que nos estemos enfrentando la integración de los diferentes problema es posible. En este sentido el presente trabajo pretende describir un problema real donde los problemas de *aircraft routing*, *crew pairing* y *fleet assignment* pueden ser integrados eficientemente en un único modelo.

1.2. Antecedentes

Un trabajo pionero en resolver problemas integrados de *aircraft routing* y *crew pairing* fue Cordeau et Al. [1] En su trabajo ellos describieron una formulación basado en *path-variables* en donde todas las rutas de los aviones y también las de las tripulaciones representan columnas en un modelo de programación lineal entera (MILP). Del mismo modo, también describen un enfoque heurístico usando la descomposición de Bender para resolver la relajación lineal del problema. En dicha descomposición el problema se divide en dos partes, un problema maestro definido por las variables de los aviones y un subproblema definido por las variables de las tripulaciones, siendo ambos resueltos mediante técnica de generación de columnas.

Más adelante se demostró en un trabajo de Mercier et al. [2] que si

invertimos el orden de los problemas, es decir, que el problema maestro esté definido por las variables de las tripulaciones y el subproblema por las variables de los aviones obtenemos mejores resultados. En la misma línea, otros trabajos que tratan problemas integrados también modelan cada ruta factible de avión como una variable binaria y hacen uso de técnicas de generación de columnas para resolver los problemas. Más tarde Weide et Al describió un procedimiento que resolvía los dos problemas originales de manera iterativa, usando la solución de uno para resolver el otro. Este último procedimiento trataba de crear soluciones robustas minimizando el número de cambios de tripulaciones y aviones. El presente trabajo pretende utilizar un enfoque distinto para resolver el problema integrado de *aircraft routing*, *crew pairing* y *fleet assignment*. Dicho enfoque se puede encontrar, explicado al detalle, en el siguiente artículo.

1.3. Objetivos

El problema integrado que se utilizará en el presente trabajo tiene como objetivo principal identificar itinerarios para rutas de aviones y tripulaciones en un periodo de tiempo de un día. Es importante mencionar que los itinerarios, llamados rutas, son anónimos en tanto a que no han sido asignados a aviones o tripulaciones concretas. Complementariamente a este objetivo se plantean los siguientes objetivos secundarios:

- Proporcionar una descripción de un problema de *aircraft routing*, *crew pairing* y *fleet assignment* aplicado a un contexto real
- Proponer una formulación para resolver el problema integrado de *aircraft routing*, *crew pairing* y *fleet assignment*
- Dar un enfoque heurístico para encontrar las soluciones al problema integrado
- Mostrar los resultados computacionales obtenido de la aplicación de los enfoques escogidos al problema integrado

1.4. Herramientas

1.4.1. Python

Python [3] es un lenguaje de programación interpretado creado en el año 1991, por Guidovan Rossum y que en los últimos tiempos ha ganado una gran popularidad entre desarrolladores y académicos. Este éxito se debe a las virtudes que posee el lenguaje, que paso a nombrar a continuación

- Es muy sencillo de usar, especialmente en comparación a otros lenguajes, como por ejemplo C
- Posee una comunidad madura, el lenguaje posee en su haber un rodaje de 30 años, y muy activa.
- Es un lenguaje considerablemente versátil con el que puedes crear aplicaciones de toda índole ya que cuenta con un amplio abanico de librerías y frameworks.
- Big data, Machine Learning y Cloud Computing. No es una sorpresa para nadie el que la Ciencia de datos y el Procesamiento de Datos Masivos son sectores de la informática que han protagonizado un auge en los últimos años. Pues bien, Python cuenta con muchas delas librerías más punteras dentro de este ámbito, como pueden ser TensorFlow y PyTorch, de Google y Facebook, así como una buena cantidad de librerías de alta calidad para procesar y representar datos, como pueden ser pandas o numpy.

En mi caso particular me decanté por él debido a que:

En mi caso particular me decanté por él debido a que:

- Ya tenía experiencia con eél en aplicaciones de ingeniería logística
- Poseía librerías que me iban a facilitar considerablemente la labor de realizar el proyecto, como Pulp, Numpy y Pandas
- Por lo destacable que es en el campo de la usabilidad
- Big data, Machine Learning y Cloud Computing. No es una sorpresa para nadie el que la Ciencia de datos y el Procesamiento de Datos Masivos son sectores de la informática que han protagonizado un auge en los últimos años. Pues bien, Python cuenta con muchas delas librerías más punteras dentro de este ámbito, como pueden ser TensorFlow

y PyTorch, de Google y Facebook, así como una buena cantidad de librerías de alta calidad para procesar y respresentar datos, como pueden ser pandas o numpy.

A pesar de sus grandes virtudes Python también tiene numerosos defectos. Por ejemplo, hay que tener en cuenta que al ser un lenguaje interpretado no puede competir de por si solo en rendimiento con lenguajes como C. Dicho esto, este problema puede ser solventado en gran medida mediante el uso de librerías como Numpy, que te ofrecen un abanico de funciones matemáticas muy eficientes, ya que están compiladas en C.

1.4.2. Jupyter Notebook

Jupyter Notebook [4] es una aplicación cliente-servidor lanzada en 2015 por la organización sin ánimo de lucro Proyecto Jupyter. Permite crear y compartir documentos weben formato JSON que siguen un esquema versionado y una lista ordenada de celdas de entrada y de salida. Estas celdas albergan, entre otras cosas, código, texto (en formato Markdown), fórmulas matemáticas y ecuaciones, o también contenido multimedia (Rich Media).El programa se ejecuta desde la aplicación web cliente que funciona en cualquier navegador estándar

Jupyter Notebook tiene la virtud de que en una sola interfaz, los usuarios pueden escribir, documentar y ejecutar código, visualizar datos, realizar cálculos y ver los resultados. El código se organiza en celdas independientes, es decir, es posible probar bloques concretos de código de forma individual. Es debido a estas razones por lo que me decidí a usarlo para este proyecto.

1.4.3. Numpy

Numpy [5] es una biblioteca de funciones matemáticas de alto nivel creada por Travis Oliphant en el año 1995. Hacer uso de ella es fundamental si el rendimiento es un factor importante en el programa que se esté desarrollando. Esto es porque las funciones de Numpy están compiladas en C e importadas a Python, lo que las hace mucho más rápidas de lo que Python puede alcanzar debido a sus limitaciones como lenguaje de programación interpretado

1.4.4. Pandas

Pandas [6] es una biblioteca escrita por Wes McKinney en el año 2008 como extensión de NumPy para manipulación y análisis de datos para el lenguaje de programación Python. En particular, ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales.

En el caso particular que atañe a este trabajo, la librería ha sido de gran utilidad a la hora de importar y procesar los datos iniciales que el programa requiere para funcionar.

1.4.5. Pulp y Cbc

Cbc (Coin-or branch and cut) [7] es una herramienta de código abierto creada por Google y escrita en C++ que permite resolver problemas de programación lineal entera. Pulp [8] es una librería de modelado de problemas de programación linea que nos aporta funcionalidades para traducir un modelo matemático a las entradas que necesita una herramienta como Cbc para resolver un problema de programación lineal.

Estas herramientas son de suma importancia para este trabajo, ya que son las que, en última instancia, nos permiten obtener una solución al problema que intentamos resolver.

Capítulo 2

El problema

En el capítulo anterior se ha realizado una introducción al problema objeto de este trabajo, a continuación se procederá a aportar una descripción detallada del problema así como la definición del modelo matemático que resuelve el mismo.

2.1. Descripción

Nos enfrentamos a un problema en el cual una compañía de vuelos regional oferta para un día determinado una cierta cantidad de vuelos y desea encontrar el itinerario de rutas de coste mínimo que satisfaga todos los vuelos. Es importante señalar que la compañía dispone de dos operadoras de vuelo (ATB, AC6), cada una con sus propias tripulaciones y aviones, y que las tripulaciones de una operadora no pueden viajar en los aviones de otra operadora.

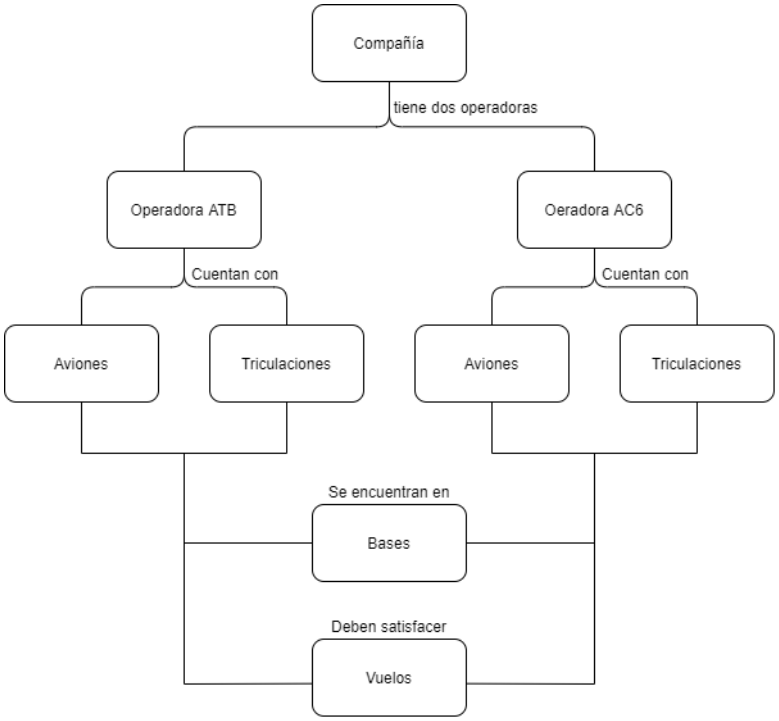

Figura 2.1: Esquema con la diferentes entidades del problema y sus relaciones

Nuestra aerolínea opera vuelos entre islas, conectando pequeños aeropuertos localizados cerca de zonas residenciales y que están cerrados por la noche, por lo que los vuelos deberán ser planificado dentro del intervalo de tiempo comprendido entre las 7 a.m y las 11 p.m . En nuestro problema hay implicados 7 aeropuertos (LPA, TFN, TFS, SPC, VDE, ACE, FUE). De todos esos aeropuertos hay dos que son *bases*, en este caso LPA y TFN. Lo que diferencia a las bases del resto de aeropuertos en que todas las rutas han de empezar en una base, teniendo que terminar la ruta obligatoriamente en la misma base de la que se partió. Esto es así porque los profesionales que forman las tripulaciones viven en una de las bases y si se permitiera que una tripulación no terminara su ruta en su base de partida se incurriría en un sobre coste excesivo debido a los gastos derivados de tener que encontrar y pagar una estancia para todos los miembros de la tripulación.

En cuanto a las tripulaciones, las rutas de éstas tienen unas restricciones específicas que dependen de a operadora, como la cantidad máxima de vuelos que pueden contener la ruta así como la duración máxima de la ruta. También el número de rutas activas en un determinado instante de tiempo ha de ser menor que la cantidad de tripulaciones disponibles que había en la base al comienzo del día

Existen, de igual forma, un conjunto de reglas de la secuencia de vuelos. Para este problema el tiempo mínimo que puede transcurrir entre dos vuelos de una misma ruta es 20 minutos, ya que es el tiempo que se necesita para preparar un vuelo. Del mismo modo, el tiempo de conexión máximo entre dos vuelos de una misma ruta es de 3 horas. Se considera la conexión entre dos vuelos de una misma ruta es corta si el tiempo de conexión es mayor a 20 minutos y menor a 30 minutos. La única diferencia entre una conexión corta y una larga es que en las conexiones largas se permite que la tripulación efectúa un cambio de avión antes de continuar con el siguiente vuelo de la ruta, penalizando el cambio de avión en la ruta en el coste de la misma, y en las conexiones cortas esto no está permitido. Esta última restricción se debe al tiempo añadido que supone para una tripulación el cambiar de avión.

A cada ruta se le asigna un coste de acuerdo a la operadora que pertenezca y al sumatorio de los tiempos de conexión de la ruta. Los tiempos de conexión muy cortos o muy largos se penalizan considerablemente. De hecho, los tiempos de conexión cortos, aquellos de menos de 30 minutos, no son deseables ya que pueden llevar a la propagación del tiempo de retraso del vuelo en caso de haberlo, mientras que los tiempos de conexión largos deberían evitarse ya que para este problema el coste aumenta proporcionalmente al tiempo gastado si que haya un vuelo activo. En vez de asignar un coste a cada ruta de los aviones, para la compañía está interesada en minimizar el número de aviones usados en la solución.

Resumiendo, el problema estudiado en este trabajo pide encontrar, simultáneamente, una asignación factible de vuelos a operadoras, y las rutas

de los aviones y las tripulaciones de manera que se satisfagan todos los vuelos ofertados, con el objetivo de minimizar el coste total y maximizar la robustez de la planificación.

2.2. Modelo matemático

2.2.1. Introducción al modelo

El modelo que vamos a presentar a continuación es una adaptación reducida del modelo de variables de arco para aviones y de camino para tripulaciones presentado en el trabajo Salazar-González et Al [9].

Para esta formulación usaremos dos grafos acíclicos. El primero , llamado *aircraft graph*, se usa para representar secuencias de vuelos válidas para rutas de aviones. El segundo, llamado *crew graph*, se usa para representar secuencias de vuelos válidas para rutas de tripulaciones. Una secuencia de vuelos es factible para una ruta de aeronave si el tiempo de conexión entre los dos vuelos es mayor a 20 minutos. Una secuencia de vuelos es factible para una ruta de tripulación si el tiempo de conexión es mayor a 20 minutos y menor que 3 horas. Esto implica que el *crew graph* está contenido en el *aircraft graph* ya que el primero es una versión más restrictiva del segundo. Ambos grafos cuentan con el mismo conjunto N de nodos, dados por la unión de del conjunto de nodos N^f, representando los vuelos, N^{bd} denotando las bases de salida, y N^{ba}, indicando las bases de llegada.

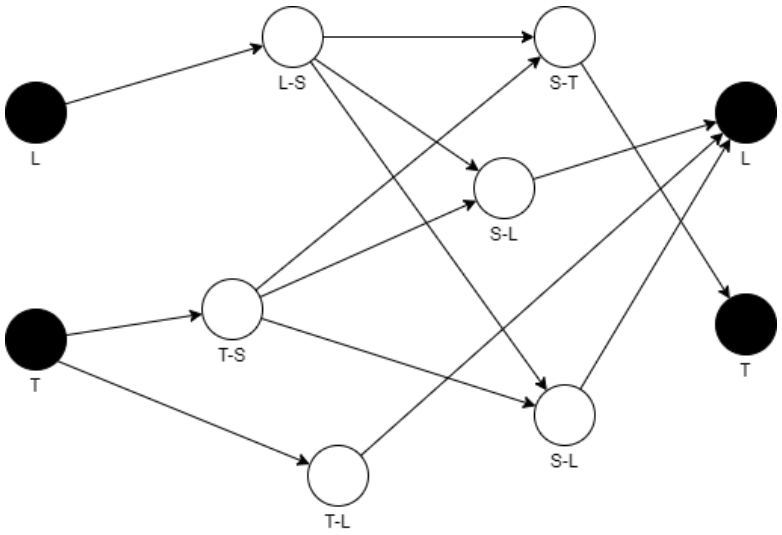

Figura 2.2: Ejemplo de un *Aircraft Graph*

El motivo por el que se han duplicado los nodos de las bases en nodos de salida y llegada es para permitirnos trabajar con grafos acíclicos. El *aircraft graph* $G^a = (N, A^a)$ (respectivamente el *crew graph* $G^c = (N, A^c)$) contiene un arco $(i, j) \in A^a$ (respectivamente $(i, j) \in A^c$) si se cumple alguna de estas condiciones:

- $i, j \in N^f$ representando dos vuelos que pueden ir en secuencia en una ruta
- Correspondiéndose $i \in N^{bd}$ con una base y $j \in N^f$ siendo el primer vuelo de la ruta que parte de esa base
- Correspondiéndose $j \in N^{ba}$ con una base y $i \in N^f$ siendo el último vuelo de la ruta que termina en esa base

La Figure 2.2 muestra un ejemplo se muestra un ejemplo de un *aircraft graph* G^a. Dicho grafo se corresponde con una instancia con 6 vuelos, indicados por los nodos blancos, que conectan 3 aeropuertos (TFN, LPA y SPC como T, L, S respectivamente), siendo dos de ellos bases, indicadas por los nodos negros. Las letras que hay bajo los nodos de las bases representan el aeropuerto correspondiente al nodo, mientras el par de letras bajo los nodos de los vuelos representan el par de aeropuertos origen-destino de cada uno de los vuelos.

En este grafo, N^{bd} contiene los dos nodos negros de la izquierda mientras que N^{ba} contiene los dos de la derecha. Desde cada base de salida hay un arco hacia cada vuelo que sale de dicha base. Del mismo modo, hay un arco hacia cada base de llegada desde cada vuelo que aterrice en dicha base. Un grafo de tripulación sería muy similar al ejemplificado anteriormente, con la única diferencia de que aquellos arcos que conectaran vuelos cuyo tiempo de conexión fuera mayor a 3 horas no estarían permitidos.

Como se ha mencionado en apartados anteriores la aerolínea se divide en dos operadoras, representadas por el conjunto K. Sabiendo esto y contando con el conjunto N^{bd} que representa al conjunto de nodos de salida tenemos que, para $k \in K$ y $l \in N^{bd}$, n_c^{kl} y n_a^{kl} son el número de tripulaciones y aviones, respectivamente, que se encuentran para la operadora k en la base l.

Las rutas de las tripulaciones y de los aviones se identifican meidante una operadora $k \in K$ y una base $l \in N^{bd}$. De hecho, es de suma importancia conocer la base de cada una de las tripulaciones ya que en las rutas de éstas se impone la restricción de que la base de salida y la de llegada sean la misma, con el fin de evitar los sobrecostes que supondría que toda la tripulación pasara la noche fuera de su base.

Las rutas de las tripulaciones y de los aviones se identifican meidante una operadora $k \in K$ y una base $l \in N^{bd}$. De hecho, es de suma importancia conocer la base de cada una de las tripulaciones ya que en las rutas de

éstas se impone la restricción de que la base de salida y la de llegada sean la misma, con el fin de evitar los sobrecostes que supondría que toda la tripulación pasara la noche fuera de su base.

Movidos porla necesidad de modelar la naturaleza compleja de nuestro problema a través de una única función objetivo se han considerado los cuatro siguientes pesos:

- α: peso sobre la suma de los tiempos de conexión sobre las rutas de las tripulaciones.
- β_k: peso sobre el número de rutas de tripulación para cada operadora la operadora $k \in K$
- γ: peso del número de rutas de los aviones para cada operadora $k \in K$.
- δ: peso sobre el número de cambios de avión durante las rutas de las tripulaciones.

Es importante señalar que estos pesos son aportados por la aerolínea una vez realizado un análisis exhaustivo del problema por parte de la misma.

2.2.2. Modelo *Arc-Path*

En este modelo vamos a utilizar caminos para representar las rutas de las tripulaciones y arcos para representar las rutas de los aviones. Como hemos duplicado los nodos correspondientes a las bases, una ruta de tripulación se corresponde con una ruta en G^c desde una base de salida hasta una base de llegada. Por tanto, definimos como R_c^{kl} al conjunto de rutas de factibles de tripulación dentro del grafo $G^c = (N, A^c$ para una una operadora $k \in K$ partiendo de una base $l \in N^{bd}$. Es de suma importancia tener en cuenta que una ruta de tripulación será factible si y solo si satisface las restricciones en la duración de la ruta, el máximo número de vualos ejecutables y en la secuencia de vuelo.

Cada ruta $R \in R_c^{kl}$ tiene asociado un coste c_R, cuyo valor depende de la operadora a la que pertenezca la ruta así como de la suma lastrada, por medio de los pesos definidos en la sección anterior, delos tiempos de conexión entre los vuelos consecutivos de la ruta. Como ya se había mencionado anteriormente, los pesos en los tiempos de conexión se utilizan para penalizar aquellos tiempos que sean excesivamente largos o excesivamente cortos.

Para representar rutas de las tripulaciones definimos una variable binaria x_R por cada ruta $R \in R_c^{kl}$, de maner $x_R = 1$ si y solo si la ruta R es asignada a una tripulación perteneciente a la operadora $k \in K$ que tiene como base de salida a $l \in N^{bd}$

Para representar rutas de los aviones definimos una variable binaria y_{ij}^{kl} para cada arco $(i, j) \in A^a$, cada operadora $k \in K$ y cada base de salida $l \in N^{bd}$, de manera que $y_{ij}^{kl} = 1$ si y solo si el arco (i, j) es operado por un avión perteneciente a la operadora k con base de salida en l

Para representar los cambios de avión en una ruta de tripulación definimos una variable binaria z_{ij} de manera que, por cada arco $(i, j) \in A^c/A_s^c$, o lo que es lo mismo, por cada arco perteneciente al grafo que contiene exclusivamente aquellos arcos (i, j) cuyos tiempos de conexión no sean cortos, $z_{ij} = 1$ si y solo si un cambiio de avión se produce entre los vuelos i y j ($i, j \in N^f$

Una formulación matemática para el problema minimiza

$$\sum_{k\in K,\, l\in N^{bd},\, R\in\mathcal{R}_c^{kl}} (\alpha\cdot c_R+\beta_k)\cdot x_R + \sum_{k\in K,\, l\in N^{bd},\, i\in N^{bd},\, j\in N^f:(i,j)\in A^a} \gamma_k\cdot y_{ij}^{kl} + \delta\cdot \sum_{(i,j)\in A^c\setminus A_s^c} z_{ij}, \quad (1)$$

sujeta a restricciones en las tripulaciones

$$\sum_{k\in K,\, l\in N^{bd},\, R\in\mathcal{R}_c^{kl}:\, i\in R} x_R = 1 \quad \forall\, i\in N^f, \quad (2)$$

$$\sum_{R\in\mathcal{R}_c^{kl}} x_R \le n_c^{kl} \quad \forall\, k\in K,\, l\in N^{bd}, \quad (3)$$

$$x_R\in\{0,1\} \quad \forall\, k\in K,\, l\in N^{bd},\, R\in\mathcal{R}_c^{kl}, \quad (4)$$

restricciones en los aviones

$$\sum_{(i,j)\in A^a} y_{ij}^{kl} = \sum_{(j,i)\in A^a} y_{ji}^{kl} \quad \forall\, k\in K,\, l\in N^{bd},\, i\in N^f, \quad (5)$$

$$\sum_{i\in N^{bd},\, j\in N^f:(i,j)\in A^a} y_{ij}^{kl} \le n_a^{kl} \quad \forall\, k\in K,\, l\in N^{bd}, \quad (6)$$

$$y_{ij}^{kl}\in\{0,1\} \quad \forall\, k\in K,\, l\in N^{bd},\, (i,j)\in A^a, \quad (7)$$

y restricciones en el enlace entre las rutas de las tripulaciones y las rutas de los aviones, en las conexiones cortas y en los cambios de avión

$$\sum_{l\in N^{bd},\, R\in\mathcal{R}_c^{kl}:\, i\in R} x_R = \sum_{l\in N^{bd},\,(i,j)\in A^a} y_{ij}^{kl} \quad \forall\, k\in K,\, i\in N^f, \quad (8)$$

$$\sum_{l\in N^{bd},\, R\in\mathcal{R}_c^{kl}:(i,j)\in R} x_R = \sum_{l\in N^{bd}} y_{ij}^{kl} \quad \forall\, k\in K,\, i,j\in N^f:(i,j)\in A_s^c, \quad (9)$$

$$\sum_{k\in K,\, l\in N^{bd},\, R\in\mathcal{R}_c^{kl}:(i,j)\in R} x_R \le \sum_{k\in K,\, l\in N^{bd}} y_{ij}^{kl} + z_{ij} \quad \forall\, i,j\in N^f:(i,j)\in A^c\setminus A_s^c, \quad (10)$$

$$z_{ij}\in\{0,1\} \quad \forall\, i,j\in N^f:(i,j)\in A^c\setminus A_s^c. \quad (11)$$

Figura 2.3: Modelo matemático del problema

La función objetivo (1) presenta los cuatro criterios a través de una única función lastrada. Las restricciones (2) imponen que cada vuelo debe estar asignado a una y solamente una tripulación. Las restricciones (3) requiern respetar el número de tripulaciones disponibles para cada operador en cada base. Las restricción (5) impone la conservación del flujo en cada nodo correspondiente a un vuelo (y en conjungción con la restricción (1) obliga a que cada vuelo deba ser operado por un único avión. La restricción 18 exige respetar el número máximo de aviones para cada operardor y para cada base. Las restricciones (8) están enlazando las rutas de las tripulaciones con las de los aviones. Éstas últimas imponen que cada vuelo sea operado por una y solo una tripulación y por un solo avión, perteneciendo ambos a la misma operadora. Si en una ruta de avión hay un arco saliente desde un nodo i por una operadora k, entonces debe haber una tripulación de la misma operadora visitando el nodo i. Las restricciones (9) prohiben que se produzcan cambios avión en conecciones cortas. Dado un par de nodos correspondientes a dos vuelos tal que su tiempo de conexión es menor a 30 minutos, si este par pertenece a una ruta de tripulación de una operadora k, entonces debe haber un avión de la operadora k que ejecute la conexión del arco entre los dos nodos. Las restricciones (26) se usan para contar el número de cambios de aviones a través de la variable z. Dados dos vuelos i y j si son operados secuencialmente por una misma tripulación, entonces o bien son operados por el mismo avión o bien hay un cambio de avión.

Capítulo 3

Enfoque de la solución

Una planteado y explicado el modelo matemático es momento de hablar de la estrategia algoritmica que se va seguir para resolver el problema. En la sección 3.1 encontraremos una detallada descripción del proceso de generación por columnas que se ha seguido para generar las rutas de tripulación. En la sección 3.2 hallaremos detalles sobre las dos fases (Cota inferior y Cálculo de la solución óptima) de la resolución del problema

3.1. Generación por columnas

El modelo matemático expuesto en el Capítulo 2 tiene, exponencialmente, una gran cantidad de variables $x_R(R \in R_a^{kl})$. Para resolver la relajación lineal del problemas, es decir, permitiendo que las varibales tomen valores continuos, seguimos un procedimiento de generación por columnas que dinámicamente va introduciendo variables en el problema. Este enfoque genera iterativamente rutas de las tripulaciones con variables teniendo costes negativos reducidos. Este procedimiento está basado en resolver el *pricing problem* que busca la ruta con el mínimo costo reducido, y corresponde con un problema ESPRC (*Elementary Shortest Path with Elementary Constraints*) para ser calculado sobre los grafos acíclicos descritos en el capítulo anterior.

El *pricing problem* es resuelto por un procedimiento de programación dinámica que se pasará a describir a continuación.

Sean φ_i $(i \in N^f)$ las variables duales de las restricciones (2), ψ^{kl} $(k \in K, l \in N^{bd})$ las variables duales de las restricciones (3), ζ_i^k $(k \in K, i \in N^f)$ las variables duales de las restricciones (8), ξ_{ij}^k $(k \in K, (i,j) \in A_s^c)$ las variables duales de las restricciones (9) y η_{ij} $((i,j) \in A^c/A_s^c)$ las variables duales de las restricciones (11). El *pricing problem* consiste en determinar para cualquier operador $k \in K$ y para cualquier base $l \in N^{BD}$, aquellas rutas tales que:

$$\alpha \cdot c_R + \beta_k - \sum_{i \in R}(\varphi_i + \zeta_i^k) - \sum_{(i,j) \in R} \eta_{ij} - \sum_{(i,j) \in R \cap A_s^c} \xi_{ij}^k - \psi^{kl} < 0$$

y que todas las restricciones descritas en el capítulo anterior para las rutas de las tripulaciones sean satisfechas

A continuación pasaremos a describir el enfoque de programación dinámica escogido para esta parte del problema. El algoritmo de programación dinámica es ejecutado para cada operadora $k \in K$ y para cada base de salida $l \in N^{bd}$. Por tanto, dados k y l, a cada vuelo $i \in N^f$ se le asigna un conjunto de etiquetas (en lugar de una única etiqueta).

Dados k y l, siendo λ_i una de las etiquetas del vuelo $i \in N^f$. La etiqueta almacena la siguiente información:

1. El beneficio de un camino R desde l hasta i, definido como

$$\pi(\lambda_i) = -\alpha \cdot c_R + \sum_{i \in R}(\varphi_i + \zeta_i^k) + \sum_{(i,j) \in R} \eta_{ij} + \sum_{(i,j) \in R \cap A_s^c} \xi_{ij}^k + \psi^{kl} < 0$$

2. El vuelo predecesor $v(\lambda_i)$ de i en el camino.
3. El número de vuelos $F(\lambda_i)$, incluyendo el vuelo i, operado en el camino.
4. La duración $D(\lambda_i)$ del camino hasta el final del vuelo i.

Los vuelos han de estar ordenados incrementalmente en tanto a su hora de salida. El algoritmo de programaciób dinñamica consiste en etiquetar, desde cada etiqueta de un vuelo i, cada sucesor j cuando las siguientes condiciones se satisfacen simultáneamente:

1. El número máximo de vuelos (incluyendo a j) es respetado.
2. La duración máxima de la ruta (incluyendo a j) es respetada.
3. Si el número máximo de vuelos es alcanzado por insertar a j en la ruta, entonces el aeropuerto de llefada de j debe coincidir con la base de salida del primer vuelo de la ruta.

Las restricciones en la secuenciación de vuelos en una ruta viene impuesta directamente por la definición de los arcos en el grafo G^c

Un método que permite acelerar sustancialmente el algoritmo es el de la aplicación de reglas de dominancia que permitan descartar etiquetas. Para aplicarlo, antes de empezar a etiquetar desde cada etiqueta de un vuelo i, cada sucesor j, comprobamos los siguiente. Siendo λ_i y n_i dos etiquestas de un vuelo i, entonces λ_i y n_i es dominada por n_i, y por tanto la etiqueta λ_i puede ser eliminada del conjunto de etiquetas del vuelo i si y solo si se cumple que:

$\pi(n_i) gt; \pi(\lambda_i)$, $F(n_i) \leq F(\lambda_i)$ y $D(n_i) \leq D(\lambda_i)$

Cuando ya tengamos todos los vuelos etiquetados, seleccionamos aquellas etiquetas cuyo beneficio sea mayor que cero, o dicho de otra manera, aquellas cuyo coste reducido sea menor que cero, cuyo aeropuerto de llegada coincida con su base y cuya operadora conincida con la cual estemos generando las rutas en este momento. Para cada una de las etiquetas seleccionadas, la ruta es reconstruida deede el vuelo seleccionado yendo marcha atrás hacia su prodecesor hasta que encontremos la base de salida $l \in N^{bd}$.

3.2. Algoritmos

Nuestro problema se resuelve en dos fases. A continuación se pasará a describir cada una de las fases

3.2.1. Cota inferior

La cota inferior se calcula resolviendo la relación lineal de nuestro modelo haciendo uso del procedimiento de programación dinámica, descrito en el apartado anterior, en las variables de las tripulaciones. El fundamento del algoritmo es el siguiente:

1. Calculamos un conjunto de rutas iniciales
2. Ejecutamos la relajación lineal de nuestro modelo con las rutas que se tengan hasta el momento
3. Lanzamos nuestro algoritmo de programación dinámica, que me genera tantas rutas como hayan etiquetas con un coste reducido negativo
4. Repetimos los pasos 2 y 3 hasta que el algoritmo de programación dinámica no encuentre etiquetas con un coste reducido negativo

Capítulo 4

Análisis de resultados

4.1. Resultados computacionales

Con el objetivo de poder cuantificar el rendimiento del enfoque heurístico propuesto, se han obtenido datos reales de vuelos de una aerolínea regional establecida en las Islas Canarias. Los datos presentados se corresponden con los vuelos ofertados por la compañía el 15 de marzo de 2020. La Tabla 4.1 cuenta con un identificador (inst.) de instancia de la prueba, el numero de vuelos (#f) que se utilizaron en dicha prueba y la cantidad de conexiones cortas y largas, representadas por (#OC) y (#LC) respectivamente.

inst.	#f	#SC	#LC
1	84	14	424
2	94	15	351
3	114	23	614
4	100	17	475
5	86	13	367
6	104	20	519
7	98	18	479
8	110	22	570

Tabla 4.1: Detalle de las instancias de prueba

Para estas instancias de pruebas, se han considerado los siguientes valores para los pesos en la duración de los tiempos de conexión entre las rutas. 2 para los tiempos de conexión entre 20 y 30 minutos, 1 paara los tiempos de conexión entre 30 minutos y una hora, 3 para los tiempos de conexión entre una hora y dos horas y finalmente 5 para los tiempos de conexión entre dos y tres horas. De igual forma, hemos escogido los siguientes pesos en la función objetivo: α = 1, $\beta_K = 1000(k \in K)$, $\gamma_k = 10(k \in K)$ y δ = 100. Tal y como se puede apreciar en los pesos, el objetivo principal es minimizar el número tripulaciones utilizadas para cubrir todos los vuelos, aunque también es importante minimizar los cambios de avión, los tiempos de conexión y el número de aviones usados para cubrir todos los vuelos.

Todos los tests fueron ejecutados en un portátil de uso personal, con procesador i7-8550 a 1.90GHz y una memoria RAM de 8 Gb. Como herramienta para resolver los problemas de programación lineal se esta usando CBC. Un único hilo de ejecución.

Para cada instancia de prueba la Tabla 4.2 nos da información sobre el número de variables de tripulación y de avión durante el proceso de resolución del problema. En particular dice que:

- El número $({}_RLP)x$ de variables de tripulación generadas en la primera fase
- El número (y_aLP) de las variables de aviones generadas en la primera fase

inst.	#f	x_RLP	x_aLP
1	84	932	2660
2	94	1522	3464
3	114	2412	5268
4	100	1659	3844
5	86	1276	2928
6	104	1911	4280
7	98	1701	3908
7	110	2237	4080

Tabla 4.2: Detalle de las instancias de prueba

Finalmente, nuestra tercer y última Tabla 4.3 nos compara como cambia el reindimiento del algoritmo en función del tamaño de cada instancia de prueba. Para esta tabla tenemos los siguientes atributos:

- Número de vuelos (#f) de la instancia de prueba
- El resultado obtenido por la cota inferior (LB) en la primera fase
- El resultado obtenido por el modelo completo en la segunda fase
- tiempo en segundos de la ejecución de la cota inferior T_{LB}
- tiempo en segundos de la ejecución del modelo completo T_{MILP}
- tiempo total en segundos de la ejecución

inst.	#f	T_{LB}	LB	T_{MILP}	MILP	T_{Tot}
1	84	157	16595	1.49	16615	158.49
2	94	875	17975	5.77	18010	880.77
3	114	3830	21560	19.98	21620	3849.98
4	100	1280	18175	15.53	18285	1295.53
5	86	341	15760	2.68	17790	343.68
6	104	2051	19270	89.53	19430	2140.53
7	98	1730	18905	4.86	19000	1734.86
8	110	3900	20370	27.32	20475	3927.32

Tabla 4.3: Rendimiento del algoritmo para instancias de diferentes tamaños

Capítulo 5

Conclusiones y líneas futuras

En este proyecto se ha presentado un problema que si bien cuenta con la base no tiene la complejidad suficiente como para considerarse aplicable al mundo real. Este tipo de problemas de optimización en el contexto profesional de empresa tienden a contar con una serie de características más complejas que no se han contemplado en este trabajo.

Por tanto, si bien este trabajo sirve como un buen punto de partida para plantear soluciones eficientes a problemas de optimización de esta índole todavía hay un largo camino que recorrer para ofrecer soluciones que puedan ser valiosas a nivel profesional para las empresas del sector de la aviación.

Otro aspecto clave a mejorar en futuros trabajos es la eficiencia de los algoritmos. Actualmente el programa es capaz de encontrar soluciones óptimas a problemas con 120 vuelos a planificar, en 1 hora y unos 30 minutoS, lo cual deja un amplio margen de mejora en este sentido. En gran medida esto se debe a al propio lenguaje Python, que al ser un lenguaje interpretado no puede competir con el redimiento que nos dan otros lenguajes de más bajo nivel como puede ser C. Es por ello que para realmente poder dar soluciones eficientes a este tipo de problemas en un lenguaje de alto nivel como Python hay que encapsular la mayor cantidad de lógica posible en funciones de liberías como Numpy, la cual nos ofrece un gran repertorio de funciones matemática de alto nivel muy eficientes, ya que éstas están desarrolladas en C y luego importadas a Python.

Finalmente quiero remarcar como un éxito el que mediante el uso de Numpy en la recta final del trabajo y el cambio de planteamiento de algunos de los algoritmos se consiguió mejorar la eficiencia del programa entorno a un 75 % pasando de que resolver el problema con más de 90 vuelos fuera una tarea imposible a poder resolver un problema con 120 vuelos en un tiempo que si bien no es para nada excelente al menos es mínimamente razonable.

Capítulo 6

Summary and Conclusions

In this project I have introduced a problem that although it resembles a real-world problem it lacks some of the complexity inherent to that kind of problems. In the context of an airline carrier this type of optimization problem have some characteristics the have not been address in this project.

For that matter, even though this work serves as a good starting point to propound efficient solutions to this kind of optimization problems there's still a long way to go to offer solutions that may be valuable in a professional level for real airline carriers

Another key aspect to improve in future works is the efficiency of the algorithms. At the moment, the program is able to find optimal solutions, to problems that haves around 120 flights to schedule, in somewhere near one and a half hour, which leaves a wide margin for improvement. To a large extent this is due to the Python language itself, because by being a interpreted language it cannot compete with lower an very efficient languages as C. That it's why to make really efficient solutions to this problem family in a high level interpreted language as Python its almost mandatory that you use libraries like Numpy, which offers a huge amount of high level very efficient math functions which are compiled in C and imported to python.

Lastly, I want to remark as a success that, by using Numpy in the final stretch of the development of this project and by changing the approach of some of the algorithms, i achieved a 75 % improvement in the efficiency of my solution, going from being almost impossible to solve a problem with more that 90 flights to being able to solve a problem with 120 flights in a time interval that although its not excellent at all, at least is more or less reasonable

Capítulo 7

Presupuesto

Para poder realizar este proyecto ha sido necesario invertir unas 300 horas de trabajo. El salario medio [10] por hora de un perfil técnico como el nuestro es de 16.60€. A parte de las horas de trabajo para este proyecto e ha usado un portátil personal valorado en 370€. Lo cual nos deja con los siguientes resultados.

Recurso	Cantidad	Coste
Hora de trabajo	300	16.60€
Ordenador	1	370€
Coste total:	**5350€**	

Tabla 7.1: Detalle del presupuesto del proyecto

Apéndice A

Título del Apéndice 1

A.1. Algoritmo de obtención Cota Inferior

```
def lower_bound():
    prob = None
    optimizable = True
    while(optimizable == True):
        prob = master_problem(new_Routes)
        prob.solve(pulp.PULP_CBC_CMD(msg=True))
        optimizable, routes = subproblem(prob, new_Routes)

        for keys, values in routes.items():
            for val in values:
                new_Routes[keys].append(val)
    return prob
```

A.2. Algoritmo de obtención de nuevas columnas

```
def subproblem(prob, crewRoutes):
    lower_routes = { (i,j) : [] for i in Operator for j in bases}
    optimizable = False
    for k in Operator:
        for l in bases:
            labels = [None] * len(flights)
            for i in rangeflights:
                if(flightDep[i] == l):
                    ilabel = labelStruct(flights[i], zero, None, 1, flightDepTime
                       [i], flightDur[i])
                    if(labels[i] == None):
                        labels[i] = [ilabel]
                    else:
                        labels[i].append(ilabel)
                    dominance(labels[i])
                if(labels[i] != None):
                    for label in labels[i]:
                        dynamic_prog(labels, prob, label, i, l, k)
            fmin = None
            start_time = time.time()
            for i in range(len(labels)):
                if(labels[i] != None):
                    for label in labels[i]:
                        if(label != None):
                            if((flightArr[i] == l) and (((label).profit - zero )
                               > np.finfo(np.float32).eps)):
                                route = [label.flight]
                                pred = label.pred
                                while(pred != None):
                                    route.append(pred.flight)
                                    pred = pred.pred
                                if(route != [None]):
                                    route.reverse()
                                    rdur=0
                                    rcost=0
                                    for i in range(len(route)-1):
                                        rdur += int(df_marzo15.loc[df_marzo15['
                                           FlightID'] == route[i], '
                                           FlightDuration']) + cgraph[route[i],
                                           route[i+1]]
                                        rcost += cgraph[route[i],route[i+1]]
                                    rdur += int(df_marzo15.loc[df_marzo15['
                                       FlightID'] == route[-1], 'FlightDuration'
                                       ])
                                    new_route = routeStruct(route, rcost, rdur)
                                    lower_routes[k,l].append(new_route)
                                    optimizable = True
    return (optimizable, lower_routes)
```

A.3. Algoritmo de Regla de dominancia

```
def dominance(f_list):
    finish = False
    i = 0
    while(finish == False):
        if(len(f_list) == 1):
            finish = True
        else:
            found = False
            j = 0
            while((found == False) and (j < len(f_list))):
                if(i != j):
                    if((f_list[i].profit > f_list[j].profit) and (f_list[i].n_f
                       <= f_list[j].n_f) and
                       (f_list[i].p_duration <= f_list[j].p_duration)):
                            found = True
                            f_list.pop(j)
                    else:
                        j+=1
                else:
                    j+=1
            if(found == True):
                i = 0
            elif(i >= len(f_list)-1):
                finish = True
            else:
                i+=1
```

A.4. Algoritmo de Programación dinámica

```
def dynamic_prog(labels, prob, label, i, inidep, op):
    duals = []
    for name, c in list(prob.constraints.items()):
        duals.append(c.pi)
    for j in range(i+1, len(flights)):
        if ((flights[i],flights[j]) in cgraph):
            route = get_route(label)
            route.append(flights[j])
            cost = get_cost(route)
            dur = label.p_duration+cgraph[(flights[i],flights[j])]+flightDur[j]
            rProfit = -(cost*alpha) - beta[Operator.index(op)] + get_duals(duals,
                prob, route, op, inidep)
            labelj = labelStruct(flights[j], rProfit, label, label.n_f+1,
               flightDepTime[j], dur)
            if(labelj.n_f <= maxFlightsRoute and labelj.p_duration <=
               maxWaitDuration):
                to_label = True
                if(labelj.n_f == 8):
                    if(inidep != flightArr[j]):
                        to_label = False
                if(to_label == True):
                    if(labels[j] == None):
                        labels[j] = [labelj]
                    else:
                        labels[j].append(labelj)
```

Bibliografía

[1] Soumis F Desrosiers J Cordeau JF, Stojkovic G. Benders decomposition for simultaneous aircraft routing and crew scheduling. 2001.

[2] François Soumis Anne Mercier. An integrated aircraft routing, crew scheduling and flight retiming model. 2007.

[3] Documentación de python 3.8.

[4] Documentación del proyecto jupyter.

[5] Documentación de numpy 1.15.

[6] Documentación de pandas 1.1.2.

[7] Documentación de cbc 2.10.

[8] Documentación de pulp 3.6.0.

[9] Juan José Salazar González Valentina Cacchiani. Heuristic approaches for flight retiming in an integrated airline scheduling problem of a regional carrier. 2019.

[10] Ine, salario medio por grupos de población.