Universitat Autònoma de Barcelona

Dipòsit digital de documents de la UAB

This is the **published version** of the bachelor thesis:

Kaplan, Martin; Benavente i Vidal, Robert, dir. Desarrollo de un sistema de gestión de vuelos y horarios para optimizar la asignación de recursos y reducir retrasos. 2024. (Enginyeria Informàtica)

This version is available at https://ddd.uab.cat/record/290091

under the terms of the BY-NC-ND license

Desarrollo de un sistema de gestión de vuelos y horarios para optimizar la asignación de recursos y reducir retrasos.

Martin Kaplan

8 de febrero de 2024

Resumen–Este trabajo trata sobre el desarrollo de un sistema de gestión de vuelos y horarios para aeropuertos. Su objetivo es reducir los retrasos y maximizar la asignación de recursos. Este sistema utiliza el algoritmo de optimización de Búsqueda Tabú para asignar puertas de embarque a los vuelos entrantes basándose en una variedad de factores, incluida la eficiencia operativa, las restricciones logísticas y la experiencia del pasajero. El objetivo del proyecto es encontrar una solución práctica al problema de la asignación de puertas en los aeropuertos, centrándose en mejorar la puntualidad de los vuelos y la utilización eficiente de los recursos, mejorar la experiencia del pasajero y reducir los costos operativos.

Palabras clave– Gestión de aeronaves, Optimización de aeropuertos, Asignación de puertas de embarque, Algoritmo de Búsqueda Tabú, Reducción de retrasos, Experiencia del pasajero, AGAP

Abstract–This project is about the development of a tool to manage flights and schedules for airports. Its objective is to reduce delays and maximize resource allocation. This tool uses the Tabu Search optimization algorithm to assign gates to incoming flights, taking into account a variety of factors, including operational efficiency, logistic restrictions and the passenger experience. The goal of the project is to find a practical solution to the Airport Gate Assignment Problem (AGAP) focusing on improving flight punctuality, the efficient use of resources, the passenger experience and reducing operative costs.

Keywords– Aircraft Management, Airport Optimization, Gate Assignment, Tabu Search Algorithm, Delay Reduction, Passenger Experience, Airport Gate Assignment Problem, AGAP

1 Introducción

Los aeropuertos son centros de actividad constantemente en movimiento, donde la eficiencia operativa es esencial para garantizar un flujo suave y eficiente de pasajeros y aeronaves. Uno de los desafíos críticos en la gestión de un aeropuerto es la asignación de puertas de embarque a aviones entrantes. Esta tarea, también conocida como Airport Gate Assignment Problem (AGAP) en inglés, tiene un impacto significativo en la puntualidad de los vuelos, la utilización de recursos y la experiencia de los pasajeros en un aeropuerto.

El AGAP se presenta como un problema de optimización multifactorial, en el cual se deben asignar a cada vuelo entrante una puerta de embarque de manera que se minimicen los costos operativos y se cumplan las restricciones logísticas y de seguridad. Una de estas restricciones puede ser, por ejemplo, el tiempo de espera de seguridad entre dos aviones que van a aparcar en la misma puerta. Este problema es particularmente desafiante debido a la variedad de factores que deben tenerse en cuenta, como las restricciones de capacidad de las puertas, la proximidad a la terminal y las conexiones de vuelos. Este trabajo, se centrará en abordar el problema desde una perspectiva práctica y se propondrá una herramienta que haga uso de un algoritmo de optimización para dar de manera eficiente una solución al problema. El objetivo último de dicho trabajo es el de mejorar la asigna-

• E-mail de contacto: martin.kaplan@autonoma.cat
• Mención realizada: Computación
• Trabajo tutorizado por: Robert Benavente Vidal (Ciencias de la computación)
• Curso 2023/2024

ción de puertas de embarque en aeropuertos, optimizando el uso de recursos, reduciendo retrasos y, de esta manera, mejorar la experiencia de los pasajeros. El logro del objetivo recién mencionado puede tener un impacto económico positivo tanto para las empresas de gestión de aeropuertos (por ejemplo, AENA en España) como para las aerolíneas ya que puede reducir costos operativos de ambas partes.

Este proyecto consistirá en desarrollar una herramienta que integre un sistema que gestionará las aeronaves en las pistas de un aeropuerto a las puertas de embarque correspondientes. Por tanto, el objetivo del proyecto será tener una herramienta con la que mediante uno o varios algoritmos que se adoptaran dada la situación del aeropuerto puedan gestionar las aeronaves con tal de evitar retrasos y se puedan optimizar los recursos. En este documento se incluirá la definición de los objetivos conseguidos en el proyecto y las consideraciones iniciales con las que se ha empezado a asumir el proyecto.

1.1. Organización de aeronaves en un aeropuerto

Tanto los aeropuertos como las aerolíneas invierten grandes esfuerzos para que la organización y gestión de las aeronaves sea la mas eficiente posible. Los aeropuertos están interesados en mejorar su gestión ya que aumenta la eficiencia del aeropuerto y por tanto puede generar más tráfico con el que se beneficia por diversos motivos, por ejemplo, el económico, ya que no solo se pueden gestionar más aviones, sino que también el coste humano y material se ve mejorado. El hecho de tener un aeropuerto eficiente es atractivo para las diversas aerolíneas para confiar en dicho aeropuerto y por tanto aumentar su volumen de vuelos en ese aeropuerto. También esta herramienta puede ayudar a reorganizar de manera más rápida y eficiente los vuelos ante un cambio repentino e inesperado como puede ser la inhabilitación de una puerta de embarque, reduciendo de esta manera posibles retrasos aumentando la satisfacción de los pasajeros. Para que nos hagamos una idea en cuanto al volumen de vuelos que un aeropuerto puede llegar a tener que gestionar, aeropuertos como el de Pekín, Londres-Heathrow y Atlanta manejan un tráfico anual de pasajeros de 204, 112 y 100 millones, respectivamente [1]. Estas cifras subrayan la magnitud de la gestión operativa y la importancia de una organización eficiente.

Por otro lado, las aerolíneas tienen interés en dicha herramienta con tal de reducir su gasto frente a un aeropuerto, ya que, por ejemplo, el tiempo que un avión está parado en una puerta de embarque genera un coste económico tanto por el aeropuerto que cobra por la ocupación de ese espacio como por no dar uso a la aeronave. El coste exacto que tendría que asumir una aerolínea es difícil de saber ya que, como se ha mencionado anteriormente, este depende de varios factores y el coste de cada uno de esos factores puede variar según la situación.

El problema de asignación de vuelos a puertas no es problema nuevo y constantemente se hacen investigaciones para encontrar cada vez soluciones más óptimas. Dicho problema es de complejidad NP-hard, por tanto, nunca podremos saber cuándo hemos alcanzado la solución óptima y encontrar una solución puede suponer un coste computacional alto que puede aumentar rápidamente conforme aumenta la complejidad de los datos sobre los que operar.

2 Estado del arte

Todos los aeropuertos implementan un software para gestionar las aeronaves en sus pistas y muchas investigaciones al respecto se han hecho para encontrar soluciones más personalizadas o eficientes en general. En la mayoría de los casos se usan algoritmos de optimización de funciones de coste. Por ejemplo, varios estudios usan el algoritmo de búsqueda tabú (algoritmo que tendremos en cuenta) como, por ejemplo, el estudio realizado por la Escuela de Tráfico y Transporte de la Universidad de Beijing Jiaotong, Bi, et al [2] en el que la función de coste a maximizar se basa en la cantidad de pasajeros que bajan en una puerta. En otro estudio que también usa el algoritmo tabú, realizado por la universidad China de Hong Kong, Kwan and Lam[4] la función de coste a minimizar en este caso es la distancia que tienen que caminar los pasajeros tanto para salir del aeropuerto, como la distancia que tienen que caminar los pasajeros en tránsito de una puerta de embarque a otra. Otros artículos usan otros varios algoritmos de manera independiente o en conjunto como podrían ser algoritmos de Branch&Bound, programación de enteros, programación lineal o métodos heurísticos como menciona Stefanie de Man [5]. Como se ha mencionado anteriormente el problema es NP-hard y por tanto la solución óptima será desconocida. Hay muchos algoritmos para minimizar o maximizar un objetivo y otros algoritmos que lo hacen a partir de una función de coste multiobjetivo. El set de restricciones es vital, como por ejemplo la compatibilidad de la aeronave con la puerta, la coordinación de entrada y salida a las puertas de las aeronaves y los tiempos de espera de seguridad. Los objetivos a maximizar o minimizar suelen coincidir en ser la distancia de desplazamiento entre puertas, el tiempo de espera de los pasajeros, eventos fuera de las puertas, etc...

3 Objetivos

3.1. Objetivos principales

El objetivo principal del proyecto es dar una solución al problema de la gestión de aeronaves en un aeropuerto. Para ello se pretende desarrollar y documentar la programación de una herramienta que pueda usar cualquier aeropuerto que al proporcionar un conjunto de vuelos y una disposición de puertas de embarque encuentre una solución en un tiempo y eficiencia razonable. La cantidad de vuelos no tiene un límite mientras conserve una proporcionalidad razonable con las puertas de embarque disponibles. Con tal de lograrlo se hará el uso del algoritmo de búsqueda tabú, en el que también podremos añadir las restricciones necesarias, como, por ejemplo, el tiempo de espera de seguridad entre vuelos para entrar en una puerta de embarque.

Como se ha mencionado anteriormente, este problema, al ser NP-hard, no podremos determinar de manera absoluta cuando hemos encontrado la solución óptima. Pese a eso, encontrar soluciones mejores que otras es posible, ya que podemos saber si la solución alcanzada es mejor que la otra midiendo, por ejemplo, como ha mejorado el valor de nuestra función de coste. Es por eso, que el objetivo más fundamental es el de reducir el máximo posible el valor de

la función de coste, y es en dónde se van a aunar la mayoría de los esfuerzos, ya que es el objetivo del conjunto del proyecto.

La función de coste que hemos mencionado que será el foco del proyecto, será una función que minimizará el tiempo de caminado de los pasajeros de un vuelo tanto de la puerta dónde llegan a otra puerta donde harán el trasbordo como a la salida del aeropuerto.

El algoritmo de búsqueda tabú que como se ha mencionado anteriormente será el que usaremos tiene ciertos parámetros, como, por ejemplo, el numero de iteraciones, la cantidad de iteraciones en la que caduca un movimiento tabú o cada cuantas iteraciones se hace un movimiento de búsqueda de soluciones vecinas alternativo. Encontrar dichos parámetros será otro de los objetivos ya que encontrar los valores óptimos para los parámetros puede proporcional una mejora substancial en la calidad de la solución.

La fecha en la que alcanzar los objetivos era el 21 de enero de 2024, por tanto, completar el proyecto en aproximadamente 16 semanas.

3.2. Objetivos secundarios

A continuación se expondrán los objetivos secundarios cuya realización queda condicionada al tiempo que quede disponible una vez se hayan completado el desarrollo de los objetivos principales.

- Implementar una interfaz con la que visualizar y gestionar la herramienta.
- Poder gestionar retrasos de aviones.
- Mejorar o añadir métodos para encontrar soluciones del algoritmo de búsqueda.
- Poder gestionar el caso en el que se satura el aeropuerto.

4 METODOLOGÍA

4.1. Propuesta

Para solucionar el problema planteado en este proyecto hemos estudiado el uso de dos de algoritmos de optimización que dadas unas restricciones, nos ayuden a solucionar el problema de manera eficiente. Para lograrlo hemos seguido la metodología que se explica en los siguientes subapartados:

4.1.1. Análisis de Requerimientos y Objetivos

Hemos identificado las necesidades específicas a considerar de los aeropuertos en términos de asignación de recursos y reducción de retrasos con tal de definir los objetivos específicos como, por ejemplo, mejorar la puntualidad de los vuelos o la eficiencia de la asignación de las puertas de embarque. También deberemos tener en cuenta con que restricciones contamos ya que, por ejemplo, la zona de embarque de un avión tiene un tiempo de espera una vez desocupado para que entre otro avión para garantizar la seguridad.

4.1.2. Selección de Algoritmos de Optimización

Para lograr los objetivos mencionados, hemos considerado los siguientes algoritmos de optimización:

- **Algoritmo de Asignación de Puertas de Embarque basado en Branch&Bound:** Este algoritmo utiliza la técnica de ramificación y acotación para encontrar una asignación óptima de puertas de embarque, teniendo en cuenta restricciones de capacidad y evitando conflictos.
- **Algoritmo de Programación de Vuelos con Búsqueda Tabú:** La búsqueda tabú se emplea para programar los vuelos teniendo en cuenta restricciones de recursos, como la disponibilidad de pistas y puertas de embarque. La lista tabú garantiza que no se repitan asignaciones ineficientes.

La selección de estos algoritmos se basó en su capacidad comprobada para mejorar la asignación de recursos y reducir retrasos en la gestión de vuelos y horarios en aeropuertos. Para empezar, nos centraremos en el algoritmo de Búsqueda Tabú ya que es de el que hay más ejemplos e información. El algoritmo de búsqueda tabú también ha sido usado de manera exitosa en otros trabajos de AGAP (Airport Gate Assignment Problem)[2][4][5]. Tanto Branch&Bound como otros algoritmos serán considerados y quizás usados una vez hecha la fase de investigación y análisis de algoritmos.

4.1.3. Implementación y Evaluación

Una vez escogido el algoritmo lo implementamos y los evaluamos poniéndolo a prueba con los datasets [3] encontrados. Para ello antes hemos tratado los datos y nos hemos cerciorado de que los datos que nos proporcionan son realmente útiles y nos pueden ayudar a poner a prueba la implementación hecha. Para tener una comparativa con la que saber la calidad de los resultados del algoritmo generaremos de forma manual soluciones que sabemos que son óptimas, concretamente compararemos el valor de la función de coste de la solución óptima generada manualmente y la dada por el algoritmo.

4.2. Herramientas

Para la programación de la herramienta se consideró usar Rust o C++ como a lenguaje de programación ya que ambos tienen características y capacidades para llevar a cabo este tipo de tarea, pero la elección de cual se ha hecho después de hacer un análisis de los pros y los contras de cada lenguaje considerando, por ejemplo, la curva de aprendizaje de Rust (ya que es un lenguaje relativamente nuevo y que desconozco), las librerías que hay disponibles y otras consideraciones a tener en cuenta. También se ha usado Python para tratar los datasets junto a la librería de Pandas y también para poder dibujar diagramas de Gantt en los que poder ver de manera más clara las soluciones. Para generar los grafos que representen los aeropuertos con más facilidad he usado una herramienta online [11]

La herramienta con la que se ha programado ha sido CLion de JetBrains para la programación de la herramienta en si y Visual Studio Code para la programación y el tratamiento de los datos con Python.

4.3. Dataset

Para llevar a cabo la fase de testeo hemos usado el dataset proporcionado por el Gobierno de San Francisco [3]. Este contiene los vuelos del aeropuerto de San Francisco desde el 1 de Diciembre de 2015 hasta el 11 de Julio de 2023. Este dataset contiene las siguientes columnas:

- Time: Tiempo consistente en día y hora de la transacción
- Airline: Aerolínea del vuelo
- Flight_number: Identificador del vuelo
- Transaction: Si el vuelo es de llegada o salida
- Terminal: Terminal donde el vuelo llegará o saldrá
- Gate: Puerta en la que llegará o saldrá el vuelo
- Remark: Anotaciones sobre el vuelo (mayoritariamente vacías)

A este dataset se le ha hecho un tratamiento ya que para el código vamos a tener que relacionar los vuelos que están saliendo con su transacción correspondiente de llegada y viceversa. Una vez hecha la relación se tendrán que eliminar los vuelos que tenga fechas irracionales si se generan como, por ejemplo, que tengan la llegada y la salida a la misma hora. En cuanto a la cantidad de pasajeros en un vuelo, no es un dato del que disponga con el dataset por tanto, teniendo en cuenta que el avión estándar (Boeing 737) tiene 210 asientos, se ha generado el número de pasajeros con la siguiente sencilla formula:

$$p = 210 * U_d[0{,}7; 1{,}0]$$

Donde p es el número de pasajeros y Ud[0,7,1.0] es un valor aleatorio dentro de una distribución uniforme de entre 0.7 y 1. Otro dato que falta es la cantidad de pasajeros de un avión que van a otro vuelo, es decir, que hacen un trasbordo y a dónde lo hacen. Hacía donde hacen transbordo se asignara a una puerta aleatoria que no sea la misma que a la que esta asignada el vuelo. En cuanto al valor de los pasajeros que hacen transbordo, seguirá la siguiente formula:

$$tp = p * 0{,}3$$

Donde tp son los pasajeros que harán trasbordo, que será el 30 % de los pasajeros totales de un vuelo. Por tanto, también sabemos que los pasajeros que van a salir del aeropuerto seguirán la siguiente formula:

$$lp = p - tp$$

Dónde lp son los pasajeros que saldrán del aeropuerto.

5 Desarrollo

5.1. Selección y análisis de algoritmos

Después de revisar los diversos artículos y hacer una evaluación, los algoritmos que más habían sido mencionado fueron Búsqueda Tabú y Branch&Bound. Para hacer la selección he analizado algunas de las características principales de ambos algoritmos que me resultaron relevantes para el problema a tratar.

Branch&Bound:

- **Enfoque sistemático:** Examina el espacio de soluciones de manera jerárquica, dividiendo el problema en subproblemas más manejables.
- **Optimización Garantizada:** Garantiza encontrar la solución óptima, dado suficiente tiempo.
- **Complejidad computacional:** Tiende a ser elevada, especialmente para problemas de gran tamaño como el AGAP.
- **Limitaciones:** Puede ser lento e ineficiente en problemas con un amplio espacio de búsqueda.

Búsqueda Tabú:

- **Flexibilidad y Adaptabilidad::** Utiliza una lista tabú para evitar revisitar soluciones anteriores, lo que ayuda a explorar más eficientemente el espacio de soluciones.
- **Evita Óptimos Locales:** Capaz de salir de soluciones subóptimas, lo que es esencial en problemas complejos como el AGAP.
- **Eficiencia en tiempo real:** Más rápido que Branch&Bound, adecuado para situaciones donde el tiempo de respuesta es crítico.
- **Soluciones cercanas al óptimo:** Aunque no siempre garantiza la solución óptima, ofrece soluciones de alta calidad en un tiempo razonable.

Después del análisis concluí que debía basarme en la Búsqueda Tabú ya que consideré que dicho algoritmo se adapta mejor al dinamismo requerido para el problema, el tiempo de calculo que parece tener es menor, es decir, es más rápido, y la calidad de las soluciones proporcionadas por el algoritmo generalmente son de calidad suficiente para las necesidades del problema. Otros argumentos que me han hecho inclinarme por la Búsqueda Tabú es que he encontré más documentación y me parece una aproximación más sencilla inicialmente que Branch&Bound y que se le da un gran uso en los problemas de la misma naturaleza que el tratado en este proyecto. Para implementar este algoritmo he usado como mayor fuente de información los artículos ya mencionados anteriormente de “A Tabu Search-Based Algorithm for Airport Gate Assignment: A Case Study in Kunming, China”[2] y “An Improved Tabu Search for Airport Gate Assignment”[4].

5.2. Análisis de lenguajes de programación.

Tras un buen tiempo de reflexión y análisis decidí usar C++ como lenguaje de programación para realizar la herramienta ya que la otra opción, Rust, aunque me atraia mucho tiene una curva de aprendizaje que no estoy dispuesto de asumir con algo tan relevante como el trabajo de fin de grado y, por tanto, optaé por C++ ya que pese a estar igualados en cuanto a características estaba bastante más familiarizado y también hay mucha más documentación en el caso de que me encontrase con barreras del propio lenguaje.

5.3. Búsqueda Tabú.

5.3.1. Explicación del algoritmo

El algoritmo de búsqueda tabú es un método de optimización de búsqueda local. La búsqueda local es un método heurístico que suele ser usado en problemas que requieren de un alto nivel de computación que consiste en que en cada iteración mejoramos la solución con una candidata alternativa que llamamos solución vecina. Para escoger que solución candidata es mejor usamos una función de coste (en nuestro caso, mide el tiempo a caminar). La función de coste en nuestro caso serán minutos caminando, ya que la escala de los mapas del aeropuerto de San Francisco están en minutos de caminado. Siendo el valor a minimizar Z, la función de coste es la siguiente:

$$Z = \sum_{i \in G} \sum_{j \in G} p_{ij} t_{ij} + \sum_{i \in G} p_{i0} t_{i0}$$

- $\boldsymbol{G}$ es el conjunto de puertas.
- $\boldsymbol{p_{ab}}$ son los pasajeros que van de la puerta a a la puerta b. La puerta 0 es la salida.
- $\boldsymbol{t_{cd}}$ es el tiempo a caminar de la puerta c a la puerta d. La puerta es 0 es la salida del aeropuerto.

Con esta formula pretendemos dar peso a lo que cada pasajero tiene que caminar tanto para ir de un vuelo a otro como para salir del aeropuerto.

La particularidad del algoritmo tabú es que este contiene una estructura de memoria que limita movimientos que nos pueden llevar a regiones de la solución local que no nos interesan y de esta manera evitamos iteraciones innecesarias y nos dirigimos a mejores regiones de la solución.[6]

El algoritmo funciona a grandes rasgos de la siguiente manera: el algoritmo empieza encontrando una solución inicial, una vez obtenida dicha solución inicial se generan soluciones vecinas que no son más que variaciones de la solución inicial, una vez generadas las soluciones vecinas calculamos su función de coste y escogemos la solución con el mejor valor. Añadimos los movimientos realizados a la lista tabú para que en futuras iteraciones no se repitan esos movimiento y procedemos a realizar lo mismo durante las iteraciones necesarias[7]. El pseudocódigo 1 muestra una aproximación de como se vería el recién comentado código.

Código 1 Búsqueda de la mejor solución vecina

```
currentSolution = initialSolution
n = 0
while n < iterations do
    solutions = findNeighbours(currentSolution)
    bestZ = getCost(currentSolution)
    for all solution ∈ solutions do
        if bestZ > getCost(solution) then
            bestZ = getCost(solution)
            currentSolution = solution
        end if
    end for
    n++
end while
```

El movimiento de inserción es el movimiento por defecto del algoritmo para encontrar soluciones vecinas. Este movimiento consiste básicamente en insertar el vuelo que está programado en una puerta digamos A a otra puerta B siempre que quepa en ella (es decir, que no se solape con otro vuelo que ya esté en la puerta B) y ese movimiento no sea tabú. El pseudocódigo 2 muestra una aproximación de como es dicho movimiento.

Código 2 Soluciones con movimiento de inserción

```
function DoInsert(solution)
    solutions = [ ]
    flights = solution.flights
    gates = solution.gates
    for all f ∈ flights do
        for all g ∈ gates & g ≠ g_i do
            if g.CanGate(f) & !IsTabu(g, f) then
                solutions.push(new Solution(g, f))
                MakeMoveTabu(g,f)
            end if
        end for
    end for
    return solutions
end function
```

El movimiento de intercambio es el movimiento que usaremos cada N iteraciones que nos ayudará a salir de mínimos locales y a hacer una exploración más eficiente del espacio de soluciones. El movimiento de intercambio consiste en intercambiar dos vuelos siempre y cuando este intercambio no genere conflictos en las puertas dónde pertenecen dichos vuelos, y que el movimiento no sea tabú.El pseudocódigo 3 muestra una aproximación de como es dicho movimiento.

El movimiento de inserción es el movimiento por defecto ya que en el proceso de búsqueda permite hacer una exploración de un espacio de la solución mas especifico. Mientras que el movimiento de intercambio ofrece una mayor diversificación de soluciones (razón por el que lo usamos sobretodo para salir de mínimos locales). Por tanto, usamos el movimiento de inserción por defecto por la capacidad de explorar de manera mas intensiva una región concreta del espacio de soluciones.

Código 3 Soluciones con movimiento de intercambio

```
function DoExchange(solution)
    solutions = [ ]
    flights = solution.flights
    for all f1 ∈ flights do
        for all f2 ∈ flights do
            if f1.CanExchange(f2) then
                if !IsTabu(f1, f2) then
                    solutions.push(new Solution(f1, f2))
                    MakeMoveTabu(f1,f2)
                end if
            end if
        end for
    end for
    return solutions
end function
```

En nuestro caso otros parámetros propios del algoritmo que usaremos serán la tenencia, que es la cantidad de iteraciones en la que caduca un movimiento tabú. Y también tendremos un parámetro que decidirá cada cuanto hacemos un movimiento de búsqueda de solución alternativo al movimiento por defecto. Este movimiento alternativo, como ya hemos comentado, tiene como objetivo evitar máximos locales a la vez que dar la posibilidad de alcanzar una mejor solución más rápido.

5.3.2. Implementación del algoritmo

El enfoque de la implementación esta basada en POO (Programación Orientada a Objetos) y por tanto la infraestructura del código está hecha en clases como, por ejemplo, un vuelo, una puerta, etc. Las fechas de, por ejemplo, la llegada o salida de un vuelo se guardan usando la clase de una librería[10] que ayuda a gestionar todo lo que tiene que ver con fechas y tiempo. También he hecho la implementación para importar los datos del dataset para, de esta manera, poder también ir probando que el código que se va haciendo funciona como es debido. También se ha creado un pequeño script de Python que simplifica los grafos creados por la herramienta[11] (esta genera un XML difícil de leer en C++) y este fichero con el grafo simplificado es el que es importado por la herramienta. Los grafos son importantes ya que con esta estructura de datos podemos representar las puertas del aeropuerto y la relación entre ellas para poder calcular el tiempo de caminado. Se ha realizado también la inicialización de estos datos importados asignado los valores a las estructuras creadas previamente. Para calcular las distancias entre las puertas, cuando se importa el grafo se usa una librería [12] que nos ayuda a crear un grafo y aplicar Dijkstra sobre él.

En cuanto al algoritmo en si, se genera la solución inicial que consiste en ordenar los vuelos que hay que organizar por hora de salida en orden descendente y asignarlos a las puertas que los puedan aparcar cumpliendo con las restricciones de seguridad (5 minutos de espera entre la salida del último vuelo y la entrada del siguiente), que un vuelo este en una puerta y solo una, y que un vuelo no se solape con otro en la misma puerta. Una vez generada la solución inicial, el algoritmo es capaz de encontrar soluciones vecinas usando el movimiento de inserción y el movimiento de intercambio. El algoritmo tiene una lista tabú que restringe movimientos hechos previamente, aunque no permanentemente, sino que hasta ciertas iteraciones futuras, para poder recorrer el gradiente de la solución de manera más eficiente.

5.3.3. Tratamiento de datos

Como se ha comentado en el apartado 4.3, el dataset del que disponemos no nos sirve de manera inmediata ya que los vuelos están separados por llegadas y salidas. Por tanto, tenemos que relacionar y combinar la llegada y la salida de un mismo vuelo. Primero de todo hemos ordenado el dataset por puertas y en el caso de ser de la misma puerta por la hora de llegada (a_i) o salida (d_i) (dependiendo de si es un vuelo de salida o llegada) en orden ascendente. Una vez ordenado el dataset procedemos con el algoritmo mostrado en el diagrama de la figura 3.

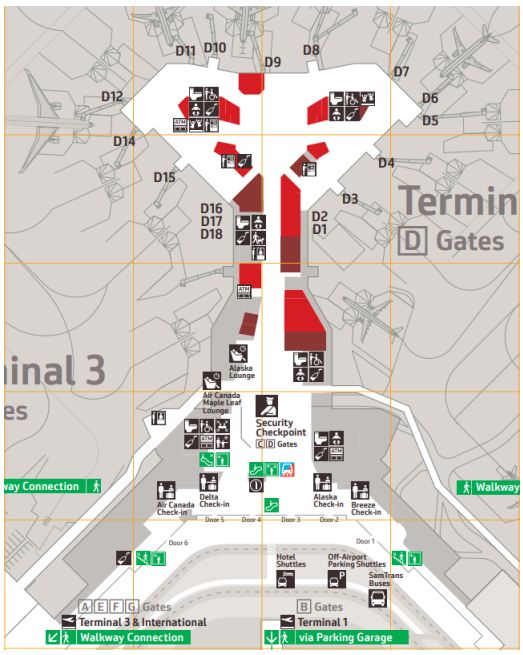

Fig. 1: Mapa oficial de las puertas D de la terminal 2 del aeropuerto de San Francisco

Fig. 2: Puertas D de la terminal 2 del aeropuerto de San Francisco representadas en un grafo

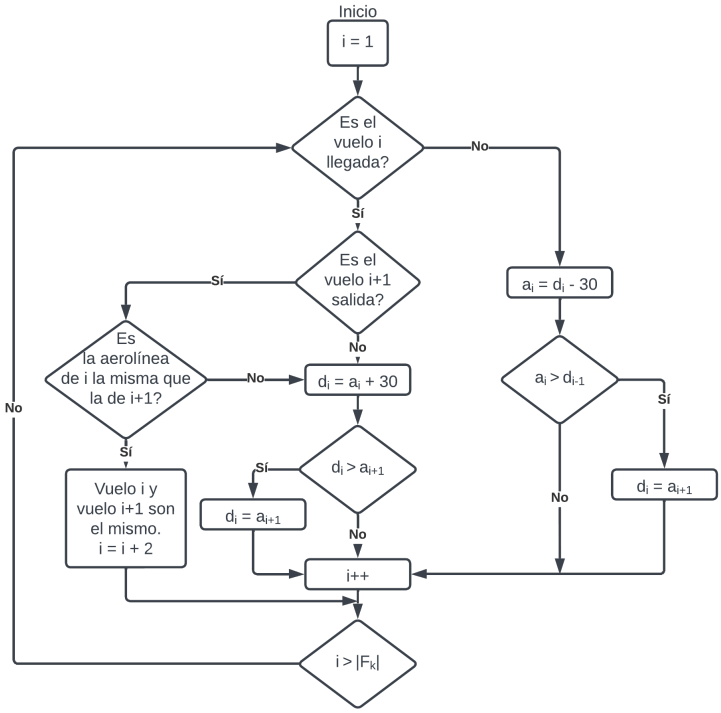

Fig. 3: Diagrama de flujo del tratamiento de datos de vuelos

En cuanto a las puertas se ha usado los mapas proporcionados por la página oficial del aeropuerto de San Francisco [8]. Para hacer las pruebas se ha usado concretamente el mapa de las puertas D y G de la terminal 2 [9]. La zona del aeropuerto con las puertas D de la terminal 2 a replicar en el grafo se muestra en la figura 1. El grafo generado representando las puertas D1 a D17 se puede ver en la figura 2.

6 Resultados

Para empezar el algoritmo ha sido probado con un dataset muy simplificado consistente de solo 15 vuelos distribuidos durante un día y una disposición de 16 puertas de embarque. La solución inicial generada tiene un valor de la función de coste de 16906.2 (puede variar un poco ya que los valores de las cantidades de pasajeros son aleatorios). El diagrama de Gantt con la organización generada en la solución inicial se puede observar en la figura 4. Recordemos que la solución inicial es la solución generada con una asignación básica.

En el diagrama tenemos en el eje de las ordenadas las puertas que tienen como a mínimo 1 vuelo, en el eje de abscisas tenemos el tiempo. Cada rectángulo del diagrama representa un vuelo, el color de este es determinado por la puerta en la que se encuentra y la anchura por el tiempo que el vuelo ocupa la puerta.

La solución final generada después de 50 iteraciones haciendo el movimiento alternativo de intercambio de vuelos cada 5 iteraciones, tiene un valor de la función de coste de 10878, es decir, que podemos decir que la solución generada reduce en un 35.65 % el valor inicial de la función de coste. El diagrama de Gantt con la organización generada en la solución final se puede observar en la figura 5.

Recordemos que el valor de la función de coste son minutos caminados por todos los pasajeros de la solución. Si tenemos en cuenta que por cada vuelo hay de media 178.5 pasajeros y tenemos 15 vuelos, en toda la solución tenemos aproximadamente 2678. Si dividimos el tiempo total por la cantidad de pasajeros obtenemos que cada pasajero en la solución inicial camina 6.5 minutos. Con el mismo cálculo pero con la solución podemos ver que cada pasajero pasa a caminar 4.7 minutos.

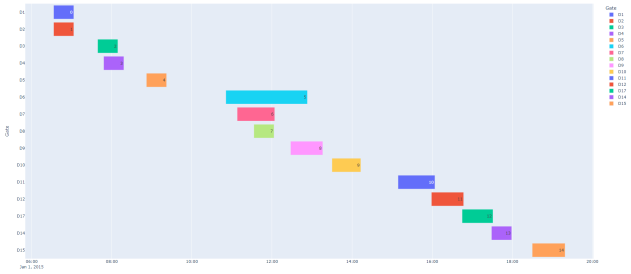

Fig. 4: Diagrama de Gantt de la solución inicial. En el eje vertical las puertas, en el eje horizontal el tiempo. Cada rectángulo es un vuelo, el color de este es la puerta en la que se encuentra y la anchura el tiempo que el vuelo ocupa la puerta.

Para encontrar los parámetros ideales hemos usado el método de Grid Search, que consiste en establecer un rango para cada uno de los parametros que queremos encontrar y probamos sus combinaciones ejecutando el programa con los mismos datos cada vez. En nuestro caso hemos hecho la búsqueda de la mejor cantidad de iteraciones totales y de las iteraciones de caducidad de los movimientos tabú (a partir de ahora le llamaremos tenencia).

El mejor resultado, como está marcado en la tabla 1 de color rojo, es el resultante de aplicar el algoritmo durante

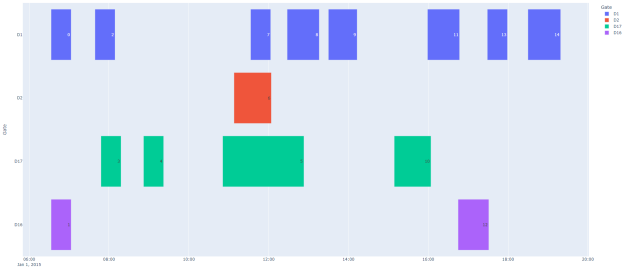

Fig. 5: Diagrama de Gantt de la solución final. Los aviones se acumulan solo en 4 de las 16 puertas totales ya que estas 4 son las más cercanas a la salida, especialmente la primera puerta.

Resultados del Grid Search					
Ten./ Iter.	10	20	30	40	50
75	113264.2	108981.6	112273.0	112185.0	111448.8
150	110083.0	112473.6	107340.2	108613.8	110357.0
225	111908.0	109375.8	107888.0	108350.4	109887.4
300	108752.8	108431.0	105385.4	105940.2	110132.0
375	110214.8	108743.4	107774.0	108434.6	109934.0
450	109778.2	108492.2	110820.6	107264.4	108535.6
525	107502.4	108191.2	107951.8	108874.6	109243.6

TABLA 1: RESULTADOS DE CADA COMBINACIÓN DENTRO DEL RANGO DE CADA PARÁMETRO. CADA FILA ES UN NUMERO DE ITERACIONES Y CADA COLUMNA ES UNA TENENCIA.

300 iteraciones con una tenencia de 30 iteraciones. La solución inicial sobre la que hemos aplicado el Grid Search tenía un valor de coste de 129133, por tanto la mejora respecto a la solución inicial es del 18.39 %. Podemos también observar que la mejora de la solución final respecto a la solución inicial es mucho más substancial cuanto más pequeño es el problema sobre el que se aplica el algoritmo.

Tamaño	BT Ideal		BT Manual		% de Mejora
	Z	t(s)	Z	t(s)	
15 x 4	10878	0.043	10878	0.047	0
50 x 8	46667.4	25	46831.4	23	0.35
80 x 12	31904.6	22	34562.8	18	7.69
120 x 16	105385.4	83	109121.6	121	3.42
155 x 20	99627.4	265	103758.8	282	3.98
190 x 24	105461.4	304	109248	385	3.46
240 x 26	145777	566	146027	875	0.17

TABLA 2: COMPARATIVA DE LOS DISTINTOS RESULTADOS ENTRE EL ALGORITMO CON LOS PARÁMETROS IDEALES Y LOS PARÁMETROS MANUALES. EL TAMAÑO ES VUELOS X PUERTAS. LAS Z SON EL VALOR FINAL. LA T ES EL TIEMPO EN SEGUNDOS. BT COMO ABREVIACIÓN DE BÚSQUEDA TABÚ.

Como podemos ver en la tabla 2 la mejora obtenida entre el algoritmo con y sin parámetros ideales no parece tener un patrón de en que casos marca mayor diferencia pero si que podemos ver que aunque la diferencia no sea muy grande si que se producen resultados mejores con los parámetros ideales. Incluso en casos en los que la diferencia entre los valores de la función de coste no son muy grandes si que hay una diferencia en el tiempo de ejecución, llegando antes a la solución el algoritmo con parámetros ideales. El cálculo del porcentaje de mejora sigue la fórmula 1.

$$\%Mejora = \frac{Original - Tuned}{Original} \times 100 \qquad (1)$$

Otra cosa que podemos observar para dar validez al algoritmo es que como podemos observar en la figura 6 el valor de la función de coste escapa del mínimo local. Esto es debido en gran parte al movimiento alternativo que abre el abanico de soluciones posibles cada ciertas iteraciones haciendo posible la exploración de espacios nuevos sobre los que seguir con el movimiento por defecto.

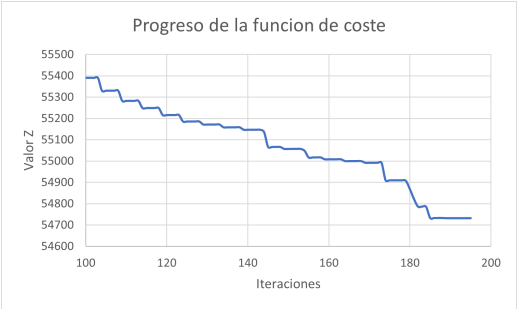

Fig. 6: Gráfico en el que se muestra el progreso de reducción del valor de la función de coste. En el eje vertical, el valor de la función de coste. En el eje horizontal las iteraciones.

7 CONCLUSIONES

El objetivo principal del proyecto era desarrollar un sistema para optimizar la asignación de recursos y reducir retrasos en aeropuertos utilizando un algoritmo de optimización. Se propuso este objetivo ya que, una herramienta que cumpliera con dicho objetivo mejoraría la experiencia de los pasajeros y los gastos de tanto las aerolíneas como las encargadas del mantenimiento del aeropuerto.

Al final del trabajo se ha conseguido crear una herramienta que dado un conjunto de vuelos y puertas encuentra una solución mejor que la solución que se puede llegar a hacer a mano o con la asignación básica, mejorando de esta manera la experiencia de los pasajeros y la gestión de recursos del aeropuerto en un tiempo razonable. Por tanto, podemos concluir finalmente que el algoritmo de búsqueda tabú es un método eficiente para encontrar soluciones en problemas NP-hard con las características del AGAP, ya que ha mostrado ser capaz de no quedarse estancado en mínimos locales y a través de función de coste mostrar mejoras substanciales en una variedad de casos.

En el futuro esta herramienta podría mejorar no solo abordando los objetivos secundarios que se han propuesto, sino que probablemente también se podría optimizar más en coste de computo mejorando de esta manera el tiempo de ejecución. Y por último, está herramienta no tan solo tiene un uso en el ámbito aeroportuario, sino que también puede usar en la gestión de cualquier otro problema

de asignación de recursos temporales con restricciones similares como podría ser, por ejemplo, la asignación de camiones a puertas de descarga de una fábrica, problema que origino la idea de este trabajo.

AGRADECIMIENTOS

Primero de todo quería agradecer a mi tutor del trabajo, Robert Benavente, que me ha ayudado desde el momento en el que hicimos la primera reunión en la que acepto mi propuesta de TFG proporcionándome ideas, soluciones a dudas y buenas sugerencias durante todo este trabajo. También quería agradecer a Josep Maria, que me ha acompañado prácticamente durante todo el mi paso en la UAB y con el que he sufrido las penurias y los altos de la carrera, sin él probablemente lo hubiera pasado mucho peor durante esta época académica. Otra persona que ha sido importante para llegar hasta aquí ha sido mi profesor de matemáticas y tutor del instituto, Carles de Cubas, que revivió con gran fuerza mi pasión por la ciéncia y especialmente las matemáticas. Y por último, a Gonzalo "The Salinator" Salinas, con quien sin duda no estaría dónde estoy, la persona con la que más buenos tiempos comparto y con la que más aprendo (NH).

REFERENCIAS

[1] Wikipedia. "Anexo: Aeropuertos por tráfico de pasajeros". Wikipedia, [Online]. Disponible en: https://es.wikipedia.org/wiki/Anexo:Aeropuertos_por_tr\%C3\%A1fico_de_pasajeros. Accedido el: 8 de febrero de 2024.

[2] Jun Bi, Zhen Wu, Lei Wang, Dongfan Xie, Xiaomei Zhao, .A Tabu Search-Based Algorithm for Airport Gate Assignment: A Case Study in Kunming, China", Journal of Advanced Transportation, vol. 2020, Article ID 8835201, 13 pages, 2020.

[3] San Francisco Government. SFO Gate and Stand Assignment Information Created: December 1, 2015. Last Updated: July 11, 2023 https://data.sfgov.org/Transportation/SFO-Gate-and-Stand-Assignment-Information/chfu-j7tc

[4] KWAN, Chuek-Lam, "An Improved Tabu Search for Airport Gate Assignment", The chinese university of Hong Kong, 118 pages, September 2009.

[5] Stefanie de Man, "Flight to gate assignment: methods and complexities at different planning horizons", Aerlines Magazine, e-zine issue: 53, 5 pages, 2011.

[6] Wikipedia. "Local Search (optimization)".Wikipedia, [Online]. Disponible en: https://en.wikipedia.org/wiki/Local_search_(optimization). Accedido el: 8 de febrero de 2024

[7] Wikipedia. "Tabu Search". Wikipedia, [Online]. Disponible en: https://en.wikipedia.org/wiki/Tabu_search. Accedido el: 8 de febrero de 2024

[8] San Francisco International Airport. Static Airport Maps. https://www.flysfo.com/maps/static-maps (accedido 8 de febrero de 2024).

[9] San Francisco International Airport, "Mapa de T2, San Francisco International Airport"del Agosto 2023. [Online]. Available: https://www.flysfo.com/sites/default/files/2023-08/230828_SFO_Tabloid_Face%20A_T2_Web.pdf. Accedido el: 8 de febrero de 2024

[10] J. Dumais. C++ DateTime Library. https://github.com/jeremydumais/CPP-DateTime-library. (accedido 8 de febrero de 2024).

[11] UnickSoft. Graph Online. https://graphonline.ru/en/\#. (accedido 8 de febrero de 2024).

[12] R. Peters. LibGraph. https://github.com/peters-r/libgraph. (accedido 8 de febrero de 2024).

APÉNDICE

A.1. Planificación del proyecto

A.1.1. Planificación para los objetivos principales

Con tal de lograr los objetivos se han llevado a cabo varias tareas. El propósito de cumplir con estos objetivos es disponer de una herramienta efectiva y usable, que funcione de manera adecuada en términos de eficiencia y calidad de solución, utilizando para ello un conjunto de datos coherentes. En última instancia, esto servirá para demostrar la habilidad de proponer y resolver problemas, habilidades que se han adquirido a lo largo del grado.

Para obtener el resultado final esperado se han realizado las siguientes tareas:

- Investigación y análisis de algoritmos de optimización existentes usados en otros estudios para abordar el problema.
    - Búsqueda Tabú
    - Branch & Bound
    - Buscar otros algoritmos
- Hacer un análisis y consideración de los pros y contras de usar Rust o C++ como lenguaje de programación con el que hacer un proyecto.
- Implementar los algoritmos escogidos en la tarea de análisis anterior.
    - Crear toda la infraestructura de entidades/clases
    - Parseo de datos
    - Inicialización de los datos con los que se hará el desarrollo inicial

  - Generar solución inicial
  - Generar soluciones vecinas
  - Lógica de iteración

- Recolección y tratamiento de los datos de los datasets para poner con ellos a prueba la implementación hecha.
  - Limpiar datos irrelevantes o no válidos
  - Tratar los datos (juntar vuelos de salida y entrada en una misma puerta)
  - Generar dataset final estructurado para el parseo del codigo
- Testeo de los algoritmos con los datos tratados.
  - Implementar un protocolo de evaluación
  - Con distintos conjuntos del dataset
  - Con distintos parámetros del algoritmo implementado

A.1.2. Objetivos secundarios

Estos objetivos son los que se habrían intentado alcanzar si hubiese habido más tiempo una vez los objetivos generales se lograron:

- Implementar una interfaz con la que ver y gestionar la herramienta.
  - Visualización del aeropuerto y aviones
  - Introducción de datos
  - Introducción de parámetros del algoritmo
  - Otras opciones
- Añadir otros algoritmos con los que mejorar la herramienta.

A.1.3. Diagrama de planificación

En la figura 7 se puede ver el calendario con la planificación del proyecto y el índice con los colores respectivos a sus fases y tareas del proyecto en la figura 8.

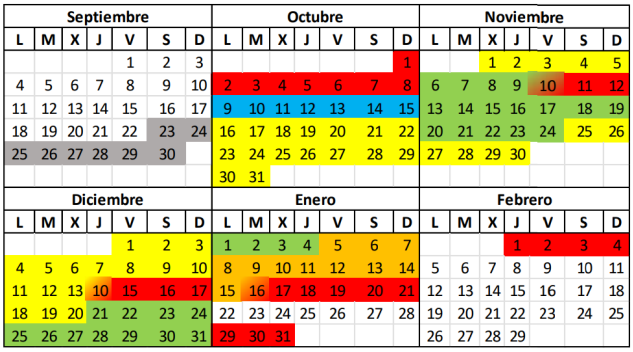

Fig. 7: Calendario de la planificación llevada a cabo en el proyecto.

Fig. 8: Índice con los colores respectivos a sus fases y tareas del proyecto mostrados en el calendario.