Inteligencia Artificial

Informe Final: Problema ALSP

José Miguel Quezada Silva

10 de enero de 2021

Evaluación

Mejoras 1ra Entrega (10 %):	_____
Código Fuente (10 %):	_____
Representación (15 %):	_____
Descripción del algoritmo (20 %):	_____
Experimentos (10 %):	_____
Resultados (10 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100):	_____

Resumen

El problema de planificación de aterrizaje de aviones (ALSP o ALP, según sus siglas en inglés) consiste en la optimización de la asignación de tiempos de aterrizaje de un conjunto de aviones en una o más pistas, estando limitados por ventanas de tiempo para cada uno. Estas características resultan en la clasificación de este como NP-dificil o NP-hard [15]. El objetivo principal de esta documento es hacer un repaso de las distintas formas en que se ha resuelto el problema hasta el día de hoy, además de presentar una formalización matemática y los resultados de la experimentación realizada a partir de la implementación de un algoritmo de búsqueda Tabú en un programa computacional.

1. Introducción

A continuación, se presenta una descripción, antecedentes y resultados experimentales de un problema de optimización cuyo estudio y aplicación en la vida real es esencial para la realización de vuelos seguros por todo el mundo. Se trata del Aircraft Landing Schedule Problem (ASLP) o Problema de Planificación de Aterrizajes de Aviones, que consiste en la asignación óptima de los tiempos de aterrizaje de un conjunto de aviones limitados por distintos rangos de tiempo.
La motivación de este trabajo es la aplicación real de una de las distintas estrategias utilizadas en el campo de la inteligencia artificial para evaluar distintas soluciones hasta llegar a resultados satisfactorios. Se detalla su definición, la descripción de distintas estrategias utilizadas hasta el día de hoy para resolverlo y un planteamiento de un modelo matemático para su resolución, especificando las variables utilizadas y sus restricciones. Seguido a esto, se encuentra la descripción y detalles de una implementación de un algoritmo de búsqueda incompleto Tabú, que cuenta con la particularidad de guardar en una lista soluciones que se han evaluado, para no volver a instanciarlas. Se presenta además una representación del algoritmo acorde al problema y los resultados de una serie de experimentos probando distintas instancias y parámetros. Se

concluye que es díficil que este algoritmo de búsqueda incompleta obtenga buenos resultados de manera efectiva al inicializar las soluciones de manera aleatoria.

2. Definición del Problema

El problema se basa en el siguiente escenario: Existe un conjunto de aviones que aterrizarán en una sola pista de un aeropuerto. Cada uno cuenta con una ventana de tiempo en la que puede hacerlo, basada en el momento en que es detectado en el radar del aeropuerto y las velocidades que pueda alcanzar según se le indique. En caso de viajar a su velocidad crucero, no se consume gasolina adicional, llegando así en su tiempo óptimo. En caso de llegar antes o después de este, le es aplicado un costo de penalización, debido al gasto adicional producido por viajar más rápido o por más tiempo.
El objetivo es minimizar este costo, buscando que los aterrizajes estén lo más cerca posible de sus tiempos ideales, respetando además un tiempo de separación mínimo entre ellos. Las variables consideradas corresponden al tiempo de aterrizaje y dos que consideran cuánto se distancia un aterrizaje de su tiempo ideal: una en caso de ser más temprano y otra en caso de ser más tarde que este.
El modelo y sus variables se encuentran limitados por restricciones que aseguran que el aterrizaje se realice dentro de la ventana de tiempo de cada avión, que todos estos aterricen, que se respete el tiempo mínimo necesario entre dos aterrizajes consecutivos, por motivos de seguridad; que, para un par dado de aviones, solo una de las variables binarias asociadas a su orden de llegada esté activa; y restricciones relacionadas con las distancias entre el tiempo de aterrizaje escogido y el ideal.
Al ser este un problema tan específico, no cuenta con una gran variedad de aplicaciones ni problemas relacionados. Uno de los pocos encontrados trata el problema de las capacidades de las pistas de aterizaje [8]. Sin embargo, sí cuenta con dos grandes variantes, en las que se aplica el mismo modelo para el problema exclusivo de despegue de aviones (Aircraft Take-off Problem (ATP)) y para el mixto (Aircraft Scheduling Problem (ASP)), en que se debe agendar óptimamente tanto aterrizajes como despegues de aviones [**?**].

3. Estado del Arte

Si bien este problema no ha sido tan extensamente documentado y estudiado como otros más generales y con la capacidad de ser aplicados en un amplio espectro de áreas, lleva décadas en desarrollo y ha sido abordado de varias maneras, con distintas heurísticas y modelos.
Sentando parte de las bases para su resolución, Blumstein (1959) [8] realizó uno de los primeros estudios sobre capacidad de pistas de aterrizaje, contando también con restricciones de distancia espacial y temporal entre aviones. Tobias (1972) [16] fue uno de los primeros en utilizar un método automatizado para determinar secuencias de aterrizaje, considerando múltiples pistas y tipos de aviones.

Andreussi et al (1981) [1] desarrollaron un modelo de simulación de eventos discretos para probar distintas estrategias. Dear and Sherif (1989, 1991) [22] [23] estudiaron tanto el caso estático como el dinámico, presentando un algoritmo heurístico para el problema dinámico de una pista, basado en el cambio de posición restringido. Brinton (1992) [12] realizó una búsqueda en profundidad en un árbol que representaba la secuencia en que debían aterrizar los aviones, enumerando todas las secuencias posibles de aterrizajes. Abela et al (1993) [14] aplicó una heurística basada en un algoritmo genético y Branch and Bound para resolver el problema de una sola pista, utilizando un modelo entero mixto con variables binarias. Venkatakrishnan et al (1993) [17] usaron el mismo método de Psaraftis mencionado previamente, usando heurísticas para analizar las mejoras producidas por realizar mejores secuencias, considerando las ventanas de tiempo. Desarrollaron dos formas de resolver el problema dinámico, una en la que se reducía

progresivamente la ventana de tiempo de los aviones al estar próximos a aterrizar, y la otra con la ventana fija.

Bianco et al (1993) [7] aplicaron el problema abierto del viajero ambulante (OTSP) para su modelo, tratando cada ciudad como un avión, y utilizando un algoritmo de programación dinámica para TSP con costos acumulados. Mientras que Bianco et al (1999) [6] analizaron el problema dinámico con una cota inferior y dos algorítmos heurísticos.

Ciesielski y Scerri (1997, 1998) [20] [21] utilizaron un algoritmo genético, considerando secciones de tiempo de 30 segundos para definir los tiempos de aterrizaje, resolviendo el problema nuevamente, utlizando los valores de los escenarios con los mejores resultados obtenidos en los primeros 3 minutos. Milan (1997) [13] define lotes formados por los aviones cuyo tiempo de aterrizaje ideal es cercano, para poder ordenarlos fácilmente, además de asignar prioridades a los aviones según factores externos, definidos arbitrariamente, y así ordenar los aterrizaje dentro de los lotes. Carr et al (1998, 1999, 2000) [9] [10] [11] modificaron la metodología FCFS (por orden de llegada) estándar, para que aviones de ciertas aerolíneas, por ejemplo, tuvieran mayor prioridad. Bolender and Slater (2000) [24] trataron el ALP dinámico usando teoría de colas y simulación de eventos discretos, bajo el supuesto de que los aviones aparecían según una distribución de Poisson.

Beasley et al (2000) [3] analizaron el caso estático usando búsqueda en árboles basada en problemas lineales, realizando primero el modelo para una pista, y luego extendiéndolo al caso de múltiples pistas, además de aplicar restricciones adicionales para distintas necesidades. Luego Beasley et al (2004) [4] modelaron el problema dinámico, mediante el problema del desplazamiento, y lo resolvieron tanto de manera heurística como con un algoritmo de optimización, obteniendo mejores resultados con este último.

Jung and Laguna (2003) [18] abordaron el problema mediante una segmentación de tiempo, dividir el problema en sub-problemas más pequeños, los que son resueltos óptimamente como problemas lineales. Pinol y Beasley (2006) [19] plantearon una función objetivo lineal y una no lineal, probada con dos métodos genéticos: búsqueda dispersa y un algoritmo bionómico. Bencheikh et al (2009) [5] modelaron el problema utilizando job shop y lo resolvieron con un algoritmo genético en conjunto con un algoritmo de optimización de colonias de hormigas, además de ajustar restricciones con tal de reducir trabajo computacional.

De manera más reciente, Ma et al (2014) [15] plantearon el ALP mediante un problema de permutación restringido, y diseñaron un nuevo algoritmo de aproximación que dio mejores resultados que el algoritmo de colonia de hormigas, mencionado anteriormente, y CPLEX, un software utilizado por décadas para resolver este problema.

Además se cuenta con investigaciones que lograron mejores resultados utilizando metodologías similares a las ya existentes, como Awasthi et al (2013) [2], que realizó una metodología similar a la de Beasley et al (2000). Esta mejora se debe probablemente a la mejora en la tecnología más que en mejoras en restricciones o espacios de búsqueda, puesto que la diferencia no es realmente sustancial.

La mayoría de estos equipos realizó experimentos con un número mediano a grande de aviones (pasando del orden de decenas a cientos) y con múltiples pistas de aterrizaje, típicamente hasta 4 o 5, mediante cálculos computacionales.

4. Modelo Matemático

Basado en el modelo de Beasley para el caso estático [3], y utilizando ligeras modificaciones a la notación, se tiene:

4.1. Parámetros

Datos que permanecen constantes durante cada experimento
N: Número total de pistas
p: Número de aviones totales
E_i: Tiempo más temprano del aterrizaje del avión i, $\forall i \in \{1, ..., p\}$
T_i: Tiempo ideal de aterrizaje para el avion i, $\forall i \in \{1, ..., p\}$
L_i: Tiempo más tardío de aterrizaje del avión i, $\forall i \in \{1, ..., p\}$
g_i: Penalización del avión i, por unidad de tiempo, por aterrizar antes de T_i, $\forall i \in \{1, ..., p\}$
h_i: Penalización del avión i, por unidad de tiempo, por aterrizar después de T_i, $\forall i \in \{1, ..., p\}$
S_{ij}: Separación necesaria, en tiempo, entre el aterrizaje del avión i y el aterrizaje del avión j (i aterriza antes que j), $\forall i \in \{1, ..., p\},\ \forall j \in \{1, ..., p\},\ i \neq j$

Se asume que no hay un tiempo necesario de separación entre aterrizajes en distintas pistas ($s_{ij} = 0$)

4.2. Variables

Datos cuya variación afecta al resultado final:
t_i:Tiempo de aterrizaje de avión i, $\forall i \in \{1, ..., p\}$
α_i:Tiempo de separación entre aterrizaje y tiempo ideal en caso de llegar antes de este, $\forall i \in \{1, ..., p\}$
β_i:Tiempo de separación entre aterrizaje y tiempo ideal en caso de llegar después de este, $\forall i \in \{1, ..., p\}$

$$\delta_{ij} = \begin{cases} 1, & \text{si avión i aterriza antes que avión j} \\ 0, & \text{si no} \end{cases}$$

Cuyo espacio de búsqueda corresponde a:

$$\frac{p!}{2!(p-2)!}$$

Considerando todos los pares posibles de aviones...

4.3. Restricciones

Se requiere una restricción para asegurar que el tiempo de aterrizaje esté dentro de la ventana de tiempo.

$$E_i \leq t_i \leq L_i,\ \forall i \in \{1, ..., p\} \tag{1}$$

Asegurar que respete un solo orden de aterrizaje entre dos aviones i y j.

$$\delta_{ij} + \delta_{ji} = 1,\ \forall i \in \{1, ..., p\},\ \forall j \in \{1, ..., p\},\ j > i \tag{2}$$

Con tal de considerar todos los tipos de superposición de ventanas de tiempo, se generan tres conjuntos:
U, que consiste en los pares de aviones en que no se sabe el orden de aterrizaje de dos aviones i y j, el V, en que el avión i aterriza antes que el j, pero que no necesariamente respetan la separación mínima temporal neccesaria y el W, en que la ventana de tiempo del avión i se encuentra antes que la de j, por lo menos por el mínimo tiempo de separación requerido.

$$U = \{(i,j)|E_j \leq E_i \leq L_j \vee E_j \leq L_i \leq L_j \vee E_i \leq E_j \leq L_i \vee E_i \leq L_j \leq L_i,\ \forall i \in \{1,...,p\},\ \forall j \in \{1,...,p\},\ i \neq j\}$$

$$V = \{(i,j)|L_i < E_j \wedge L_i + S_{ij} \leq E_j,\ \forall i \in \{1,...,p\},\ \forall j \in \{1,...,p\},\ i \neq j\}$$

$$W = \{(i,j)|L_i < E_j \wedge L_i + S_{ij} > E_j,\ \forall i \in \{1,...,p\},\ \forall j \in \{1,...,p\},\ i \neq j\}$$

A partir de los cuales, se originan las siguientes restricciones necesarias para que se respete el tiempo mínimo necesario entre dos aterrizajes consecutivos en la misma pista, por motivos de seguridad:

Para los conjuntos en que las ventanas de tiempo de cada par de aviones se encuentran separadas, según la definición de estos conjuntos, se tiene que siempre el avión i aterriza antes que el j.

$$\delta_{ij} = 1,\ \forall (i,j) \in W \cup V \tag{3}$$

Para respetar el tiempo de separación entre cualquier avión i y j, pertenecientes a V, el tiempo de aterrizaje de j debe ser mayor que el de i, sumado a la separación temporal mínima entre ellos.

$$t_j \geq t_i + S_{ij},\ \forall (i,j) \in V \tag{4}$$

Mientras que para que se respete la separación de tiempo en el conjunto U, el tiempo de aterrizaje del avión j debe ser mayor que el de i sumado a esta separación, en caso que $S_{ij} = 1$, o sea, que i llegue antes que j. Mientras que si $S_{ij} = 0$, el lado derecho de la ecuación se vuelve negativo, siendo M un número lo suficientemente grande, y así satisfaciéndola.

$$x_j \geq x_i + S_{ij} - M\delta_{ji},\ \forall (i,j) \in U \tag{5}$$

Sin embargo, para utilizar el menor M posible, este puede ser reemplazado por $L_i + S_{ij} - E_j$, o sea, la ventana de tiempo de i más su separación del avión j. Resultando así:

$$x_j \geq x_i + S_{ij} - (L_i + S_{ij} - E_j)\delta_{ji},\ \forall (i,j) \in U$$

$$x_j \geq x_i + S_{ij} - (L_i - E_j)\delta_{ji} + S_{ij}\delta_{ji}$$

Reemplazando δ_{ji} mediante la ecuación (5), se obtiene

$$x_j \geq x_i + S_{ij} - (L_i - E_j)\delta_{ji} - S_{ij}(1 - \delta_{ij})$$

$$x_j \geq x_i - (L_i - E_j)\delta_{ji} + S_{ij}\delta_{ij},\ \forall (i,j) \in U \tag{6}$$

Finalmente, para que las variables α y β cobren sentido en todo momento, se tiene:

α es la diferencia entre el tiempo ideal y el escogido o 0 en caso de existir $\beta > 0$

$$\alpha_i \geq T_i - t_i,\ \forall i \in \{1,...,p\} \tag{7}$$

Límites absolutos de α

$$0 \leq \alpha_i \leq T_i - E_i,\ \forall i \in \{1,...,p\} \tag{8}$$

El mismo caso que (10) pero con β

$$\beta_i \geq t_i - T_i,\ \forall i \in \{1, ..., p\} \tag{9}$$

Límites absolutos de β

$$0 \leq \beta_i \leq L_i - T_i,\ \forall i \in \{1, ..., p\} \tag{10}$$

Definición del tiempo de aterrizaje escogido, en función de α y β

$$t_i = T_i - \alpha_i + \beta_i,\ \forall i \in \{1, ..., p\} \tag{11}$$

4.4. Función Objetivo

La función busca minimizar el costo total que se acumula al hacer aterrizar un avión fuera de su tiempo ideal, siguiendo todas estas condiciones. Por lo que está compuesta por los costos adicionales de aterrizar antes o después del tiempo ideal, y los costos de cada uno de estos casos.

$$min \sum_{i=1}^{p} g_i * (T_i - t_i) + h_i * (t_i - T_i) \tag{12}$$

5. Representación

La representación escogida para las soluciones es directa y simple, consiste en vectores de tamaño p de números enteros y positivos que representan el tiempo de aterrizaje de cada avión.

$$\{13,\ 16,\ 8,\ 3,\ 25\}$$

Ejemplo de solución para un caso con 5 aviones

Al utilizar una estrategia de búsqueda incompleta, se generan vecindarios para cada solución con una representación similar, por lo que consisten en matrices pobladas con todos los movimientos posibles a la solución actual.

Los movimientos consisten en la suma o resta de una cantidad específica, definida por el usuario, en una de las dimensiones de la solución. En otras palabras, se retrasa o adelanta el tiempo de aterrizaje de uno de los aviones cada vez.

6. Descripción del algoritmo

En primer lugar, se crea una clase para guardar los datos de cada avión, como los de tiempo, los costos y las distancias mínimas a mantener con los demás aviones. Estos son leídos a partir de un archivo de texto plano y almacenados en un vector con objetos de tipo Avion.

Se inicializa la lista Tabú con un vector de tamaño *tabuS*, definido al inicio del programa, para guardar los vectores de las *tabuS* últimas soluciones evaluadas, con tal de lograr diversificar las soluciones, lo que también depende de su tamaño. Mientras mayor sea este, mayor diversificación se producirá.

A continuación se genera la solución inicial (S_Actual) de manera aleatoria, iterando sobre el vector de aviones para asignar un tiempo que se encuentre dentro de las ventanas de tiempo de cada avión, pero sin considerar la restricción de la distancia mínima entre todos los aviones, con tal de contar con soluciones más variadas. Se obtiene además su costo (costo_Actual) mediante

la función objetivo.

Luego, comienza el proceso de búsqueda de soluciones, mediante la generación de un vecindario para la solución actual. Este es generado mediante movimientos de tamaño arbitrario a cada dimensión de la solución actual, generando así $2 * p$ vecinos

Para escoger al siguiente vecino, se evalúan los que no estén en la lista Tabú y cuyos tiempos asignados se encuentren dentro de sus ventanas respectivas, mediante la función objetivo definida en (12), y se escoge al que obtenga el costo más bajo. Sin embargo, en caso de sobrepasar una tolerancia a la cantidad de soluciones infactibles seguidas (variable maxInfactibles), se seleccionan los vecinos con la menor cantidad de conflictos con respecto a las restricciones de distancias de tiempo entre cada par de aviones, con tal de encontrar soluciones factibles, e intensificar a partir de ellas.

Una vez seleccionada una solución, se agrega a la lista Tabú, con tal de que no sea elegido durante los siguientes movimientos.

El procedimiento se repite hasta cumplir con un criterio de término o al ser finalizado por el usuario mediante una señal de término (enviada con ctrl + C). El criterio de término se cumple cuando la cantidad de soluciones con la misma cantidad de conflictos (con respecto a la restricción antes mencionada) escogidos de manera continua sobrepasa un valor máximo, definido en una variable denominada max_conflicto. El aumento de tal variable se encuentra dentro de la función para escoger al siguiente vecino.

Procedure main

inicializar vector de Aviones ← archivo de entrada
inicializar matriz tabuL vacía de tamaño tabuS x p

inicializar S_Actual aleatoriamente
inicializar costo_Actual ← costoFO(S_Actual)
inicializar best_Solucion
inicializar best_Costo ← 9999999

inicializar matriz Vecindario vacía de tamaño (2∗p) x p

maxInfactibles ← 2
numInfactibles ← 0

Repeat

Vecindario ← generarVecindario(S_Actual)

If numInfactibles < maxInfactibles **then**
S_Actual ← escoger el mejor vecino de Vecindario que no esté en tabuL y que respete restricción de ventanas
Else
S_Actual ← escoger el vecino que se aproxime más a la factibilidad que no esté en tabuL y que respete restricción de ventanas
EndIf

```
    If S_Actual es infactible then
        numInfactibles++
    Else
        numInfactibles ← 0
    End If

    Costo_Actual ← CostoFO(S_Actual)

    agregar S_Actual a tabuL

    If costo_Actual < best_Costo & S_Actual es factible then
        best_Solucion ← S_Actual
        best_Costo ← Costo_Actual
    EndIf

Until criterio de término
EndProcedure
```

Listing 1: Algoritmo Tabú para ALSP

Para generar el vecindario se agrega la solución actual a dos elementos consecutivos de la matriz Vecindario, y se les aplica un movimiento positivo y negativo al tiempo del avión en que se encuentre la iteración.

Al escoger un vecino, se implementa un parámetro booleano, que, al tener valor verdadero, la función escogerVecino() funciona normalmente, iterando sobre el vecindario y seleccionando al vecino de mejor calidad. Sin embargo, en caso de obtener soluciones infactibles y sobrepasar el valor de la variable maxInfactibles, se comienza a contar la cantidad de conflictos que presenta cada avión, con respecto a los demás, debido a la insatisfacción de la restricción de distancias mínimas de tiempo. Escogiéndose al vecino más cercano a ser una solución factible.

7. Experimentos

Se realiza una serie de experimentos basados en el uso de distintas instancias para el problema, a partir de las cuales se modifican distintos parámetros, como el tamaño de la lista Tabú y de los movimientos, y la cantidad máxima de soluciones infactibles seguidas y de iteraciones para el criterio de término, de manera manual.
Los datos obtenidos consisten en el tiempo total de ejecución, el tiempo de búsqueda y el costo de la mejor solución, en caso de ser hallada, y la cantidad de soluciones evaluadas.

Se realizan 2 experimentos, en los que se asigna un valor alto y uno bajo, respectivamente, a cada uno de los parámetros mencionados previamente, mientras los demás se mantienen con valores intermedios. Esto se prueba con tres instancias de aviones, resultando así en 18 experimentos cuyos resultados se encuentran a continuación.

Las pruebas son realizadas en un sistema operativo Linux Mint 20 con g++ 9.3.0.

8. Resultados

Para instancias de pocos aviones, dígase menos de 30, el algoritmo logra encontrar soluciones de manera bastante rápida, como se ve en la Tabla 1, para la mayoría de los casos. Sin embargo,

con una mayor cantidad, se dificulta la búsqueda de soluciones factibles, en cuanto a la restricción de las distancias mínimas que debe haber entre todo par de aviones.

Al evaluar una cantidad definida de soluciones infactibles de manera seguida, el algoritmo comienza a buscar soluciones con la menor cantidad de conflictos posibles, en cuanto a la restricción mencionada. Sin embargo, al ir generando los vecindarios y seleccionando soluciones, no se tiene registro de los últimos aviones cuyos tiempos de aterrizaje fueron variados, de esta manera estancándose el programa en una cantidad específica de conflictos.

#	instancia	TabuS	paso	maxInfactibles	max_conflicto	T$_{best}$	T total	best_Costo	#repeticiones
1	1	1000	30	2	2000	2,5 s	5,8 s	5940	735
2	1	10	30	2	2000	0,4 s	8,5 s	12130	2062
3	1	100	80	2	2000	0,1 s	43,5 s	16760	10001
4	1	100	5	2	2000	0,1 s	8,9 s	62490	2029
5	1	100	30	100	2000	2,5 s	41,7 s	4860	10001
6	1	100	30	2	30	0,1 s	0,1 s	28040	32
7	2	1500	30	3	2000	20,6 s	77,3 s	5890	3840
8	2	15	30	3	2000	-	3,3 s	-	2009
9	2	150	80	3	2000	0,3 s	19,4 s	19080	2057
10	2	150	5	3	2000	-	9,9 s	-	2016
11	2	150	30	200	2000	0,2 s	21,8 s	32580	2852
12	2	150	30	3	30	0,2 s	0,2 s	57600	32
13	3	2000	30	5	3000	0,3 s	65,9 s	66070	3064
14	3	20	30	5	3000	0,8 s	11,1 s	34240	3135
15	3	200	80	5	3000	0,3 s	35,4 s	55110	3049
16	3	200	5	5	3000	-	9,1 s	-	3017
17	3	200	30	200	3000	-	12,1 s	-	3214
18	3	200	30	5	40	-	0,1 s	-	52

Tabla 1: Experimentos realizados para 3 instancias

9. Conclusiones

Si bien Tabú es una idea buena y simple para poder diversificar en una búsqueda incompleta, por sí sola no es una buena forma de resolver el problema en el caso de inicializar las soluciones de manera aleatoria, ya que lo más probable es que en casos con una cantidad considerable de aviones se quede estancado intentando buscar la mejor solución, debido a las restricciones de distancias entre todos los pares de aviones. Sin embargo también es necesario mencionar que el algoritmo implementado fallaba en estos casos al estancarse en conflictos de restricciones, como fue mencionado anteriormente.

10. Bibliografía

Referencias

[1] Bianco L y Ricciardelli S Andreussi A. A simulation model for aircraft sequencing in the near terminal area. *European Journal of Operational Research*, 8(4):345–354, 1981.

[2] Kramer O y Laessig J Awasthi A. Aircraft landing problem: Efficient algorithm for a given landing sequence. *Proceedings - 16th IEEE International Conference on Computational Science and Engineering, CSE 2013*, 2013.

[3] Sharaiha YM y Abramson D Beasley JE, Krishnamoorthy M. Scheduling aircraft landings — the static case. *Transportation Science*, 34(2):180–197, 2000.

[4] Sharaiha YM y Abramson D Beasley JE, Krishnamoorthy M. Displacement problem and dynamically scheduling aircraft landing. *Journal of the Operational Research Society*, 55(1):54–64, 2004.

[5] El Hilali Alaoui A y El Khoukhi F Bencheikh G, Boukachour J. Hybrid method for aircraft landing scheduling based on a job shop formulation. *International Journal of Computer Science and Network Security*, 9(8):78–88, 2009.

[6] Dell'Olmo P y Giordani S Bianco L. Minimizing total completion time subject to release dates and sequence dependent processing times. *Annals of Operations Research*, 86:393–415, 1999.

[7] Mingozzi A y Ricciardelli S Bianco L. The travelling salesman problem with cumulative costs. *Networks*, 23:81–91, 1993.

[8] Alfred Blumstein. The landing capacity of a runway. *Operations Research*, 7(6):752–763, 1959.

[9] Erzberger H y Neuman F Carr GC. Airline arrival prioritization in sequencing and scheduling. *second USA/Europe Air Traffic Management RD Seminar, Orlando*, 1998.

[10] Erzberger H y Neuman F Carr GC. Delay exchanges in arrival sequencing and scheduling. *Journal of Aircraft*, 36(5):785–791, 1999.

[11] Erzberger H y Neuman F Carr GC. Fast-time study of airline-influenced arrival sequencing and scheduling. *Journal of Guidance, Control, and Dynamics*, 23(3):526–531, 2000.

[12] Brinton CR. An implicit enumeration algorithm for arrival aircraft scheduling. *Proceedings IEEE/AIAA 11th Digital Avionics Systems Conference*, page 268–274, 1992.

[13] Milan J. The flow management problem in air traffic control: a model of assigning priorities for landings at a congested airport. *Transportation Planning and Technology*, 20(2):131–162, 1997.

[14] M. Krishnamoorthy A. De Silva y G. Mills J. Abela, D. Abramson. An implicit enumeration algorithm for arrival aircraft scheduling. *Proceedings 12th National ASOR Conference, Adelaide, Australia*, pages 71–90, 1993.

[15] Liu M y Huang H Ma W, Xu B. An efficient approximation algorithm for aircraft arrival sequencing and scheduling problem. *Mathematical Problems in Engineering*, 2014:1–8, 2014.

[16] Leonard Tobias. Automated aircraft scheduling methods in the near terminal area. *Journal of Aircraft*, 9(8):520–524, 1972.

[17] Barnett A y Odoni AR Venkatakrishnan CS. Landings at logan airport: describing and increasing airport capacity. *Transportation Science*, 27(3):211–227, 1993.

[18] Jung G y Laguna M. Time segmenting heuristic for an aircraft landing problem. *Leeds School of Business, University of Colorado, Boulder, USA. Working paper*, 2003.

[19] Beasley JE y Pinol H. Scatter search and bionomic algorithms for the aircraft landing problem. *European Journal of Operational Research*, 127(2):439–462, 2006.

[20] Ciesielski V y Scerri P. An anytime algorithm for scheduling of aircraft landing times using genetic algorithms. *Australian Journal of Intelligent Information Processing Systems*, 4(3-4):206–213, 1997.

[21] Ciesielski V y Scerri P. Real time genetic scheduling of aircraft landing times. *Fogel D (ed) Proceedings of the 1998 IEEE International Conference on Evolutionary Computation. IEEE, NY, USA*, page 360–364, 1998.

[22] Dear RG y Sherif YS. The dynamic scheduling of aircraft in high density terminal areas. *Microelectron Reliab*, 29(5):743–749, 1989.

[23] Dear RG y Sherif YS. An algorithm for computer assisted sequencing and scheduling of terminal area operations. *Transportation Research Part A: General*, 25(2-3):129–139, 1991.

[24] Bolender MA y Slater GL. Evaluation of scheduling methods for multiple runways. *Journal of Aircraft*, 37(3):410–416, 2000.