Aplicación para la asignación automática de puertas de embarque a los vuelos de un aeropuerto

Carlos Rea Nogales

Resumen— En el siguiente artículo se explica el desarrollo de un sistema de asignación automática de puertas de embarque a los vuelos de un aeropuerto, utilizando técnicas heurísticas de búsqueda local. Los objetivos de este sistema son lograr reducir al mínimo el número de puertas que tienen vuelos en conflicto, el tiempo de retraso de los vuelos y la distancia que caminan los pasajeros en los trasbordos. El problema examinado es abordado como un problema de satisfacción de restricciones y búsqueda local. Este problema ha sido resuelto mediante la metaheuristica de búsqueda tabú y un algoritmo voraz .Se ha implementado el sistema en el lenguaje de programación java y se ha evaluado su rendimiento tomando como referencia datos reales del aeropuerto de Barcelona – El Prat. Los resultados obtenidos muestran que es un sistema robusto y eficaz en la asignación de los vuelos y que esta asignación depende de las variables y restricciones del problema y a su vez de los parámetros del algoritmo de búsqueda tabú.

Palabras clave— Asignación de puertas a los vuelos de un aeropuerto, búsqueda tabú, problema de satisfacción de restricciones, búsqueda local.

Abstract— The following article explains the development of an automatical Airport Gate Assignment system, using heuristic tecniques of local search. The objectives of this system are to achieve a minimization in the number of gates with flights in conflict, the delay of the flights and the distance walked by the passengers in a transfer. The examined problem is tackled as a Constraint satisfaction problem and local search. This problem has been solved through the metaheuristic of tabu search and a greedy algorithm. The system has been implemented in the programmation language java and its performance has been evaluated taking as a reference real data from Barcelona's Prat Airport. The obtained results show that this is a robust and efficient system for flight assignments and that this assignation depends both on restrictions and variables of the problem and the algorithm parameters of the tabu search.

Index Terms— AGAP (Airport Gate Assignment Problem), tabu search metaheuristic, CSPs (Constraint satisfaction problem).

1 Introducción

Este proyecto que se enmarca dentro del trabajo final de grado, está relacionado con el problema de la asignación de puertas en un aeropuerto (en inglés Airport Gate Assignment Problem, AGAP). Tal problema requiere la reasignación de las puertas para asignar vuelos dinámicos y optimizar el estado de las puertas para mejorar el nivel de servicio ofrecido a los pasajeros.

El problema AGAP, es un problema de satisfacción de restricciones (en inglés Constraint satisfaction problem, CSPs) que se puede resolver mediante búsqueda local. Su complejidad es NP-Hard, es decir no existe un algoritmo conocido para encontrar una solución óptima en tiempo polinomial. Por ello es un problema complejo y difícil de optimizar, ya que involucra la consideración de muchos factores, como el número de puertas disponibles, el número de vuelos para asignar, los trasbordos entre vuelos, los tipos de aviones que se pueden alojar en una puerta, etc. El aumento del tamaño de estas variables y restricciones hace que la complejidad del problema se incremente exponencialmente.

Del mismo modo este tipo de problemas de asignación de vuelos se puede ver como un problema de planificación de tareas (en inglés Job Shop Scheduling), los cuales son de mucho interés en otras áreas. Las soluciones encontradas mediante diversas técnicas computacionales tienen desde hace años mucha importancia en el área de la inteligencia artificial.

- *E-mail de contacto: carlos.rea@e-campus.uab.cat*
- *Mención realizada: Computación.*
- *Trabajo tutorado por: Robert Benavente (Ciencias de la computación)*
- *Curso 2013/14*

Para realizar este proyecto se han revisado diferentes trabajos en el campo en el que se realiza este proyecto, encontrando diferentes maneras de resolver el problema. Hay diversos algoritmos heurísticos con los que se han obtenido resultados óptimos en términos de maximizar o minimizar una función objetivo y que son efectivos para este tipo de problemas. Entre las diferentes técnicas encontradas, se mencionan: la metaheuristica de búsqueda tabú [1, 3, 4], el algoritmo de recocido simulado [2, 7], algoritmos voraces [1], y una serie de algoritmos híbridos en los que se combinan diferentes métodos para encontrar una solución mejorada [1, 7].

En este trabajo el método elegido es un algoritmo metaheuristico de búsqueda tabú que se encarga de llevar a cabo la asignación de los vuelos y un algoritmo voraz que se ocupa de generar la solución inicial con la que comenzara la búsqueda tabú.

Se utilizó la metaheurística de búsqueda tabú, ya que es un método robusto, bueno para encontrar soluciones optimizadas en espacios de búsqueda grandes y está probada su utilidad en la resolución de este tipo de problemas. [1, 3, 4]

Los datos con los que se ha trabajado son datos reales extraídos del aeropuerto de Barcelona – El Prat. Para poder completar datos que faltaban se han generado datos ficticios o aleatorios, que son coherentes con los datos reales. En la sección 3 se explica con más detalle cómo se generan estos datos. La aplicación está preparada para trabajar con datos reales y su funcionamiento garantiza encontrar una solución. La información extraída fue guardada en ficheros de texto y cargados por la aplicación para poder realizar el procesamiento y el análisis de los resultados obtenidos mediante la solución adoptada.

El resto del artículo se organiza de la siguiente manera. En la sección 2 se presenta el marco teórico del problema. A continuación en la sección 3 se explica la metodología aplicada en el proyecto. Posteriormente en la sección 4 se muestran los resultados obtenidos y finalmente en la sección 5 se presentan las conclusiones a las que se ha llegado con este trabajo.

2 MARCO TEÓRICO

2.1 Modelado CSPs

El problema AGAP se ha planteado de la siguiente manera, considerando las siguientes variables y restricciones:

Variables = Vuelos

Dominio = Puertas de embarque

Restricciones =

- Cada vuelo tiene que ser asignado a una sola puerta.
- Dos vuelos no pueden ser asignados a la misma puerta en el mismo tiempo.
- El tiempo entre dos vuelos en una misma puerta debe ser menor a un cierto tiempo.
- Hay puertas disponibles para unos modelos de avión únicamente.
- El tiempo de retraso de un vuelo debe ser mínimo.
- La distancia entre puertas en un trasbordo debe ser mínima.
- Capacidad de aviones más grande a puertas más cercanas a la puerta de acceso del aeropuerto.

2.2 Función objetivo

La función objetivo es la ecuación que será optimizada dadas las restricciones que necesitan ser minimizadas usando técnicas de programación lineal o no lineal.
Los objetivos que se busca con la función objetivo son:

- Minimizar el número de puertas que tienen vuelos en conflicto.
- Minimizar la distancia a pie que tienen que recorrer los pasajeros en un trasbordo.

Para este trabajo se ha establecido la siguiente función objetivo F, la cual busca minimizar el tiempo:

$$F = f\_ret\,(x) * a + f\_tras\,(x) * (1 - a) \quad (1)$$

a = peso asignado al tiempo (entre 0 y 1).

x = es el estado actual de la asignación de puertas y vuelos.

f_ret (x) = función que calcula el tiempo mínimo entre dos vuelos en una misma puerta, para evitar solapamientos y se expresa mediante la siguiente ecuación.

$$\sum_{i=1}^{np} \sum_{j=1}^{nv} \sum_{k=1}^{nv} tll(V_j) - ts(V_k)\ t.q. V_j = i, V_k = i, j \neq k \quad (2)$$

np = número de puertas.

nv = número de vuelos.

tll(A) = tiempo de llegada del vuelo A.

ts(B) = tiempo de salida del vuelo B.

V = vuelos

f_tras (x) = función que calcula el tiempo mínimo que tardan en desplazarse los pasajeros en un trasbordo. Esta expresada en la siguiente ecuación.

$$\sum_{i=1}^{nv} \sum_{j=1}^{nv} d\left(PosPuerta(V_i), PosPuerta(V_j)\right) * NT(V_i, V_j) \quad (3)$$

d(A, B) = distancia Euclidea entre las puertas A y B.

posPuerta = función que devuelve la posición (x, y) de la puerta asignada al vuelo V.

NT = número de pasajeros que han de hacer un trasbordo entre los vuelos V_i y V_j.

2.2 Metaheuristica de búsqueda tabú

La búsqueda tabú (en inglés Tabu Search, TS) es un procedimiento heurístico de memoria adaptativa propuesto por Fred Glover en 1986 para la búsqueda de los óptimos globales en problemas de optimización combinatoria.

La búsqueda tabú explora el espacio de soluciones a través de una sucesión de movimientos desde una solución a la mejor de sus vecinas tratando de evitar óptimos locales. El algoritmo realiza una búsqueda por entornos en la cual se desplaza en cada iteración a la mejor solución del vecindario de la solución actual. Los principales atributos de cada solución visitada son almacenados en una lista tabú por un determinado número de iteraciones para evitar que estas soluciones sean revisitadas, es decir, para evitar ciclos en la búsqueda por entornos. Así, un elemento del vecindario de la solución actual es declarado tabú, es decir es prohibido, si alguno de sus atributos está en la lista tabú. En general, un método basado en búsqueda tabú requiere de los siguientes elementos:

- **Solución inicial**. Es la configuración inicial a partir de la cual se comienza la búsqueda.
- **Movimiento**. Es un procedimiento determinista por el que se genera una solución admisible a partir de la soluciona inicial.
- **Vecindad**. Es el conjunto de todas las soluciones admisibles que pueden ser generadas por un movimiento sobre la solución actual.
- **Lista tabú.** Es un mecanismo de memoria adaptativa que evita que la búsqueda entre en un ciclo o quede atrapada en un óptimo local.
- **Criterio de parada**. La búsqueda termina después de un número determinado de iteraciones.

2.2 Generación de una solución factible y optimizada

El primer objetivo del trabajo es generar una solución factible que sirva como punto de partida para el algoritmo de búsqueda tabú. Esta solución inicial se genera mediante un algoritmo voraz que se encarga de obtener una configuración inicial de los datos respetando las restricciones del problema. La intención es incorporar información adicional del problema que genere datos de mayor calidad y de esta manera conseguir una buena solución inicial que hará más fácil y rápida la búsqueda de una solución factible y mejorada por parte del algoritmo de búsqueda tabú. Para ello lo primero que se hace es ordenar los vuelos según la hora de llegada al aeropuerto. Después se asigna cada vuelo a una puerta disponible si existe, sino se le asigna una puerta cualquiera.

Algoritmo Solución Inicial
ListaSolución = inicializar lista vacía
Mientras (no se hayan asignado todos los vuelos) **Hacer**
    Si (la puerta que se va asignar está vacía)
        ListaSolucion = asignar puerta
    Si no
        Mientras (no se haya encontrado una puerta disponible) **hacer**
        Si el vuelo que llega a la puerta no se solapa con el último vuelo que hay en la puerta
            ListaSolucion = asignar puerta
        Fin si
        Fin mientras
    Fin si
    Si no se asignó ninguna puerta
            ListaSolucion = asignar puerta.
    Fin si
Fin mientras
Retornar listaSolucion
Fin Algoritmo Solución Inicial

El segundo objetivo es realizar una optimización de la asignación de los vuelos mediante la búsqueda tabú. El algoritmo parte de la solución inicial y se genera una estructura de vecindario que contiene el conjunto de todos los posibles movimientos que se pueden hacer desde una asignación concreta, el algoritmo selecciona el siguiente movimiento de la lista del vecindario y calcula el mejor vecino mediante la función de coste. Cuando acaba de explorar todo el vecindario, se queda con el movimiento que dio el menor valor en la función de coste, y añade la solución a la lista tabú para prohibir que se pueda elegir ese movimiento en futuras iteraciones del algoritmo. Con la lista tabú se trata de evitar que la búsqueda entre un ciclo o quede atrapada en un óptimo local, la lista tabú tiene un tamaño determinado y cuando se llena sale el

primero que entro para dar espacio al nuevo movimiento que entra. El algoritmo acaba cuando llega a una condición de parada que es un número determinado de iteraciones o cuando encuentra una solución óptima que evita todas las posibles restricciones del problema. Entonces retorna la mejor solución encontrada.

Algoritmo Tabu Search
Sol-actual = generar solución inicial.
Lista-dominios = obtener dominios.
Lista-tabu = inicializar la lista vacía.
Mientras (No se cumpla el criterio de parada) **Hacer**
Generar el vecindario actual (Sol-actual, Lista-dominios).
Reducir el vecindario (Lista-tabu).
Seleccionar la mejor solución no tabú del vecindario y almacenar como Sol-actual.
Si (coste (Sol-actual) es mejor que coste (Mejor-sol))
Mejor-sol = sol-actual
Fin Si
Lista-tabu = Actualizar lista tabú (Sol-actual)
Fin Mientras
Retornar Mejor-sol
Fin Algoritmo Tabu Search

3 METODOLOGÍA APLICADA

3.1 Obtención de los datos

Como se ha adelantado en la introducción del documento, los datos utilizados para la evaluación del sistema, son datos reales, concretamente han sido obtenidos de la página web del aeropuerto de Barcelona – El Prat. Se ha recogido información referente a los vuelos [8], aviones [9] y puertas de embarque [10].

El motivo por el cual se ha optado por buscar datos reales es para que la aplicación pueda ejecutarse en un entorno real y probar su eficacia y rendimiento.

Para las puertas se tienen en cuenta las siguientes propiedades:

- Número de puerta.
- Distancia a las demás puertas.
- Posición relativa X, Y en el plano del aeropuerto.
- Modelos de avión que pueden estacionarse en la puerta.
- Lista de aviones asignados

De la lista anterior la información real que se ha recopilado es la del número de la puerta y la posición relativa x, y a escala real en el plano del aeropuerto. La distancia a las demás puertas es una lista que contiene la distancia de una puerta a las demás puertas, y se calcula con la distancia euclidea entre dos puertas utilizando las coordenadas x, y. Los modelos de avión que pueden estacionarse en la puerta están guardados en una lista que contiene modelos de aviones generados aleatoriamente para cada vuelo. En la aplicación las puertas más alejadas a la puerta de acceso al aeropuerto, solo se les permite alojar aviones pequeños o medianos, mientras que los aviones más grandes van a las puertas más cercanas a la puerta de acceso, el motivo es que haciendo esto se evita desplazamientos de grupos grandes de pasajeros transitando por los pasillos del aeropuerto. La lista de aviones asignados contiene todos los aviones que están asignados a esa puerta en la solución final.

Para los vuelos se tienen en cuenta las siguientes propiedades:

- Número de vuelo.
- Hora de llegada al aeropuerto.
- Hora de salida del aeropuerto.
- Aerolínea del vuelo.
- Avión asignado al vuelo.
- Puerta asignada al vuelo.
- Lista de trasbordos.

De la lista anterior la información real que se recopiló fue el número de vuelo, la hora de llegada al aeropuerto y la aerolínea del vuelo, para hacer esto, se guardó un listado de todos los vuelos de un día entero desde las 07:00 a.m. hasta las 12:00 p.m. La hora de salida es ficticia y se genera aumentando en 1 hora la hora de llegada para todos los vuelos. El avión asignado al vuelo se genera de forma aleatoria, de la lista de aviones se selecciona un avión cualquiera para cada vuelo. La puerta asignada al vuelo es la puerta que se asigna para ese vuelo en la solución final. La lista de trasbordos es una lista generada con datos ficticios, para obtener estos datos se crea la información en un fichero de texto con los datos de los trasbordos para algunos vuelos, asignando trasbordos a los primeros vuelos que llegan, con vuelos que están por llegar al aeropuerto. Esta lista de trasbordos se implementa como una tabla hash cuya variable se corresponde con el vuelo con el que realiza el trasbordo y el valor se corresponde con el número de pasajeros que realizan ese trasbordo. El número de pasajeros también es aleatorio y depende del número de pasajeros del avión.

Para los aviones se tienen en cuenta las siguientes propiedades:

- Modelo de avión.
- Número de asientos del avión.

De la lista anterior los datos reales que se recopilaron son el modelo de avión y el número de asientos del avión.

Toda la información real es almacenada en ficheros de texto que posteriormente es leída por la aplicación para realizar el análisis y las pruebas respectivas.

Conocer la hora de salida y la disponibilidad de la puerta fue necesario para determinar las siguientes restricciones:

- Cada vuelo tiene que ser asignado a una sola puerta.
- Dos vuelos no pueden ser asignados a la misma puerta en el mismo tiempo.
- El tiempo entre dos vuelos en una misma puerta debe ser menor a un cierto tiempo.

La lista de distancias es una lista para cada puerta y se utilizó para determinar la siguiente restricción:

- La distancia entre puertas en un trasbordo debe ser mínima.

La lista de modelos de aviones que pueden asignarse a una puerta determina la siguiente restricción:

- Hay puertas disponibles para unos modelos de avión únicamente.

3.2 Desarrollo de la aplicación de asignación de puertas

Para evaluar el sistema de asignación de puertas se implementó una aplicación en lenguaje java compuesta de tres módulos: un módulo para las clases que contienen información de los datos, otro para las clases que implementan el algoritmo y sus funciones útiles, y por último, un módulo que muestra los resultados de forma gráfica, se puede ver el resultado en la figura 1.

En un principio de vio la posibilidad de implementar la aplicación mediante alguna librería especializada en programación con restricciones para facilitar el trabajo y obtener mejores resultados pero la curva de aprendizaje de las librerías era alto con relación al tiempo disponible para realizar el proyecto. Debido a la falta de conocimiento a la hora de modelar las restricciones de un problema complejo empleando estas librerías, me decanté por realizar la aplicación implementando lo necesario para llevarla a cabo.

En una primera fase del proyecto se implementó el módulo de las clases que contienen la información de los datos que se utilizaron para hacer las pruebas. Las clases implementadas son las siguientes:

- Clase Vuelos. Contiene información de los vuelos.
- Clase Puertas. Contiene información de las puertas
- Clase Aviones. Contiene información de los aviones

En una segunda fase del proyecto se llevó a cabo la implementación del módulo del algoritmo de búsqueda tabú. Este algoritmo necesita de las siguientes clases:

Clase Move: esta clase se utiliza para generar los movimientos del vecindario en la solución. Un movimiento está formado por dos variables, un valor que se corresponde con el índice de la puerta, es decir hace referencia a una puerta y una variable que se corresponde con el índice del vuelo. Por lo tanto esta clase contiene la información de un movimiento del vecindario.

Clase BestCandidate: esta clase se utiliza para obtener el mejor candidato del vecindario actual. La información que se guarda es el coste que devuelve la función objetivo, el movimiento actual realizado del vecindario y una lista que contiene la solución actual, en la cual se corresponde el índice con los vuelos y el valor con las puertas.

Clase TabuList: es una clase que contiene información referente a la lista tabú, se guarda una variable MaxSize que contiene la longitud de la lista tabú, y la lista en sí que es una cola, en donde se almacenan los movimientos prohibidos por el algoritmo para que no se puedan utilizar en futuras iteraciones del algoritmo.

Clase Tools: esta clase contiene la función de coste dentro de la cual se llama a la función que calcula el coste de la restricción de los vuelos solapados y la función que calcula el coste de la restricción de los trasbordos.

Clase NeighBorHood: esta clase contiene tres listas, una lista para la solución actual, otra para el dominio de la solución y por ultimo una lista de los movimientos de la solución. Esta clase se encarga de generar el vecindario de la solución actual, de reducir el vecindario con la lista tabú y de encontrar el mejor candidato de la lista de movimientos de la solución actual.

En una tercera fase del proyecto se realizó el modulo que muestra los resultados de forma gráfica como se puede ver en la figura 1. Para ello se utilizó una librería de gráficos llamada JFreeChart [11] que facilita el trabajo a la hora de mostrar los resultados. Se ha elegido esta librería entre otras cosas porque es sencilla, rápida de usar y fácil de integrar en la aplicación. Se ha optado por utilizar una herramienta de diagrama de Gantt y adaptar el horario de los vuelos porque era más fácil de mostrar los resultados con esta estructura y queda más claro el resultado final de la asignación

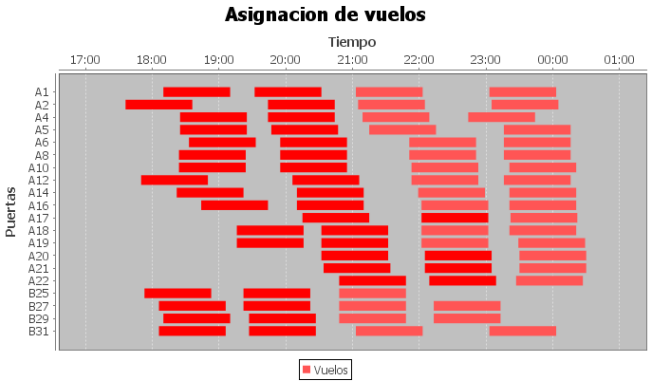

Fig. 1. Gráfico de los resultados obtenidos.

Como se puede observar en el gráfico de la figura 1, se muestra el listado de la asignación de los vuelos. En el eje vertical a la izquierda está el listado de puertas de embarque, en la parte superior se muestra el tiempo separado en lapsos de tiempo de 1 hora, las figuras de color rojo son los vuelos y hacen referencia al tiempo que permanece un vuelo en una puerta. El degradado del color solo sirve para diferenciar los vuelos en caso de que se solapen.

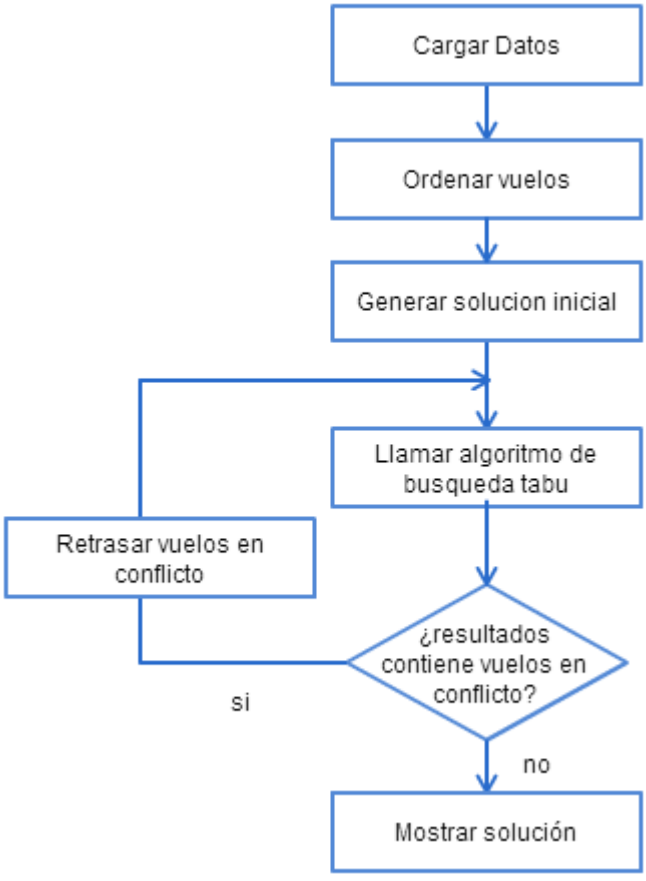

Fig. 2. Diagrama de flujo de la aplicación.

En el diagrama de la figura 2 se puede ver cada paso del proceso que sigue la aplicación desde que carga los datos hasta que los muestra de forma gráfica.

4 RESULTADOS OBTENIDOS

En este apartado voy a describir como se realizaron las pruebas y el informe de los resultados relevantes.

Para cada una de las pruebas se dispone de varias muestras de diferentes tamaños. Hay un conjunto de datos pequeño que consta de 17 vuelos y 8 puertas en total. Un conjunto de datos mediano que cuenta con 83 vuelos y 22 puertas. Por ultimo hay un conjunto de datos grande que consta de 803 vuelos y 94 puertas.

En todas las pruebas realizadas se supone que todos los aviones permanecen en el aeropuerto una hora desde la hora de llegada hasta la hora de salida. Además se ha establecido un tiempo fijo por defecto de 15 minutos entre dos vuelos en una misma puerta para solventar cualquier eventualidad en caso de haber algún retraso.

Experimento para ver la evolución del coste de la función objetivo en función del tamaño de la lista tabú: conjunto de datos mediano

Número de puertas=61
Número de vuelos=20
Número de iteraciones =100

tamaño de la lista tabú	Valor de la función objetivo
75	319864
50	319864
30	319786
20	319864
15	319777
10	319761
5	319582
1	320169

Tabla 1. Resultados obtenidos para del análisis del tamaño de la lista tabú.

Como se pude ver en la tabla 1 el mejor resultado se obtiene para una lista tabú de tamaño 5. Los resultados muestran que no hay mucha diferencia entre una lista de 5, una lista de 10 o una de 15. Para estas condiciones sobre este conjunto de datos el mejor tamaño es de 5 pero sobre otro conjunto de datos habrá que buscar el tamaño adecuado de lista tabú con la que se obtienen los mejores resultados.

Experimento para ver la evolución del coste de la función objetivo en función del número de iteraciones del algoritmo de búsqueda tabú: conjunto de datos mediano

Número de puertas=61
Número de vuelos=20
Tamaño de la lista tabú=5

Numero de iteraciones	Valor de la función objetivo	Tiempo de ejecución (ms)
25	319846	874
50	319628	1274
75	319583	1469
100	319582	1857
150	319578	2473
200	319578	3160

Tabla 2. Resultados obtenidos para evaluar el número de iteraciones.

Como se puede ver en la tabla 2 el mejor resultado se produce con 150 iteraciones del algoritmo de búsqueda tabú, sin embargo si se busca la mejor relación entre coste de la función objetivo y tiempo de ejecución se puede ver que la mejor opción se encuentra entorno a las 75-100 iteraciones del algoritmo, también se puede observar que el valor que encuentra la función objetivo con pocas iteraciones son óptimos locales, es decir el resultado es una buena solución pero no es la mejor solución.

De los dos experimentos anteriores se llega a la conclusión de que para un conjunto de datos con 61 vuelos y 20 puertas de embarque la mejor configuración de los parámetros del algoritmo de búsqueda tabú son 75 iteraciones y 5 el tamaño de la lista tabú.

Experimento para ver la evolución del coste de la función objetivo en función del número de puertas: conjunto de datos pequeño

Número de vuelos=17
Número de iteraciones=50
Tamaño de la lista tabú=5

Número de puertas	Valor de la función objetivo	Tiempo de ejecución (ms)	tiempo de retraso (min)
1	49978564	6539	6480
2	15273842	5454	4020
3	7003985	4089	2190
4	2746579	2384	960
5	2006546	2036	585
6	450099	956	135
7	207146	1288	30
8	50478	692	0
9	40298	591	0
10	35241	600	0
12	25132	673	0
16	5020	575	0
17	0	456	0

Tabla 3. Resultados obtenidos para evaluar el número de puertas.

Para este experimento se usó un conjunto pequeño de datos para asegurarse de que el modelo puede obtener una salida correcta. Podemos ver en la tabla 3 que con solo una puerta el valor de la función objetivo es el más alto lo que indica que los conflictos de los vuelos en la puerta son inevitables. Lo mismo ocurre cuando el número de puertas disponibles es demasiado pequeño, cuando ampliamos el número de puertas a 6 el conflicto en la puerta disminuye drásticamente y es mucho menor que el valor en 3 puertas. Otro dato a tener en cuenta es el tiempo de retraso de los vuelos, se puede ver en la tabla 1 como era de esperar que cuando hay solo 1 puerta es cuando se produce el mayor número de retrasos. El tiempo de retraso va disminuyendo a medida que se añaden más puertas, hasta que finalmente no hace falta retrasar ningún vuelo, a partir de 8 puertas ya se consigue asignar todos los vuelos sin retrasar ninguno.

Experimento para ver la evolución del coste de la función objetivo en función del número de vuelos: conjunto de datos mediano.

Número de puertas=20
Número de iteraciones=100
Tamaño de la lista tabú=15

Número de vuelos	Valor de la función de objetivo	Tiempo de ejecución (ms)	tiempo de retraso (min)
11	0	446	0
18	0	444	0
26	30104	601	0
33	65280	687	0
41	110712	844	0
51	212172	1122	0
61	319777	1764	0
75	534643	2728	0
83	4809960	24137	210

Tabla 4. Resultados obtenidos para evaluar el número de vuelos.

Como se puede ver en la tabla 4 como era de esperar cuando hay puertas disponibles para alojar vuelos el valor de la función objetivo es 0 ya que cada vuelo es asignado a una puerta diferente. A medida que se van añadiendo más vuelos el valor de la función objetivo se incrementa, este valor es el resultado de minimizar el tiempo entre vuelos en una puerta. A mayor número de vuelos, mayor es el espacio de búsqueda y el tiempo de ejecución del algoritmo. También se puede ver que cuando se intenta asignar 83 vuelos se producen conflictos de los vuelos en las puertas, debido a que hay vuelos que llegan a la misma hora y no hay puertas disponibles en ese momento, por eso el valor de la función objetivo se incrementa notablemente ya que tiene que retrasar vuelos un total de 210 minutos.

Experimento para ver la evolución del coste de la función objetivo en función de los trasbordos: conjunto de datos mediano

Número de vuelos=75
Número de puertas=20
Número de iteraciones=75
Tamaño de la lista tabú=10

Peso (solapamientos - trasbordos)	función de coste	tiempo de retrasos
1 - 0	534600	0
0.9 - 0.1	481197	0
0.8 - 0.2	427767	0
0.7 - 0.3	374345	0
0.6 - 0.4	320944	0
0.5 - 0.5	267474	0
0.4 - 0.6	214100	0
0.3 - 0.7	160622	0
0.2 - 0.8	107132	0
0.1 0.9	53694	0
0 - 1	9750	975

Tabla 5. Resultados obtenidos para el análisis de los trasbordos.

En este experimento se analiza el resultado según el coste asociado a las dos restricciones en la función objetivo. Como se puede ver en la tabla 5, en la primera columna de la izquierda se destaca la configuración de los pesos asignados a la función objetivo, en primera posición aparece el peso para los solapamientos y en segunda posición después del guion el peso para los trasbordos. Como se puede observar cuando se da más prioridad a los solapamientos el resultado de la función objetivo es el más grande, esto ocurre porque cuando se suma el resultado de las dos restricciones asociadas a la función objetivo, el coste de la restricción de los solapamientos es mucho mayor que el coste de los trasbordos, y el valor de la función objetivo disminuye a medida que se da más prioridad a los trasbordos. En el caso en el que solo se tiene en cuenta los trasbordos, se intenta asignar los vuelos con los que hay trasbordos a la misma puerta o a una puerta cercana, al no tener en cuenta los conflictos entre los vuelos en una misma puerta, se solapan vuelos y el sistema los retrasa hasta asignarlos a la puerta de menor distancia, con el inconveniente de que se retrasan un total de 975 minutos lo cual es excesivo, y se podría evitar si considerásemos la restricción de los vuelos en conflicto, por eso esta configuración de los pesos no es deseable en ningún caso. Si se quiere dar la misma prioridad a los trasbordos y solapamientos conviene una asignación de 0.5 para ambos. Por lo tanto según la restricción que se

quiera minimizar el algoritmo encontrara una solución en función de la prioridad asignada, ya sea darle más prioridad al tiempo mínimo entre vuelos en una puerta, o a la distancia que caminan los pasajeros en un trasbordo, no se puede obtener el mejor beneficio de los dos al mismo tiempo.

Experimento de análisis de la función objetivo: conjunto de datos grande

Número de vuelos=803
Número de puertas=94
Número de iteraciones=75
Tamaño de la lista tabú=10

Para este experimento el conjunto de datos presenta conflictos entre vuelos lo cual hace que se tenga que llamar muchas veces al algoritmo de búsqueda tabú y retrasar muchos vuelos, debido a esto el tiempo de ejecución se extiende mucho más del previsto, se ha calculado el tiempo de ejecución del algoritmo en unas 4 horas aproximadamente por lo que encontrar una solución al cabo de un cierto número de llamadas al algoritmo de búsqueda tabú para intentar asignar todos los vuelos, puede demorar más de 1 día.

Experimento para ver el resultado de la asignación de los vuelos en conflicto de forma gráfica: conjunto de datos mediano

Número de vuelos=81
Número de puertas=20
Número de iteraciones=75
Tamaño de la lista tabú=10

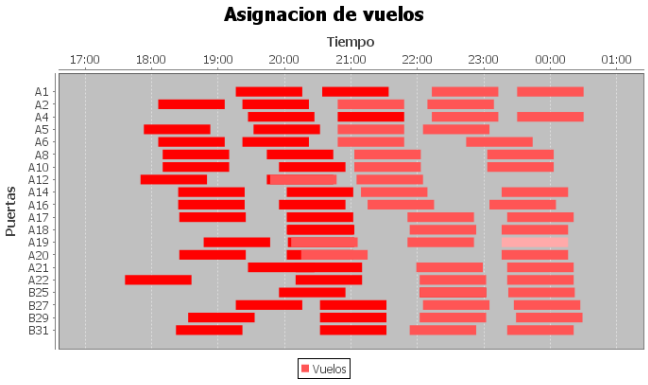

Fig. 3. Gráfico de los resultados obtenidos en la primera llamada al algoritmo de búsqueda tabú.

Como se puede ver en la figura 3, hay 6 puertas que tienen vuelos en conflicto, las puertas: A12, A18, A19, A20, A21 y A25. Las puertas A25 y A18 contienen vuelos solapados a la misma hora por eso no se distinguen. El sistema retrasa los vuelos en conflicto 15 minutos y vuelve a llamar al algoritmo de búsqueda tabú. Si en la siguiente iteración encuentra vuelos solapados los retrasa otros 15 minutos hasta asignarlos. Después de 22 llamadas al algoritmo de búsqueda tabú, se obtienen los resultados de la figura 4 que muestra una asignación correcta de los vuelos, en detrimento de los vuelos que ha tenido que retrasar, en total el tiempo de retraso es de 1240 minutos con un tiempo de ejecución de 52 segundos aproximadamente. Esta situación puede darse si se tiene en cuenta que eran 6 vuelos que llegaron al aeropuerto de improviso y no hay puertas disponibles para alojarlos entonces se opta por retrasar los vuelos un cierto tiempo hasta asignarlos.

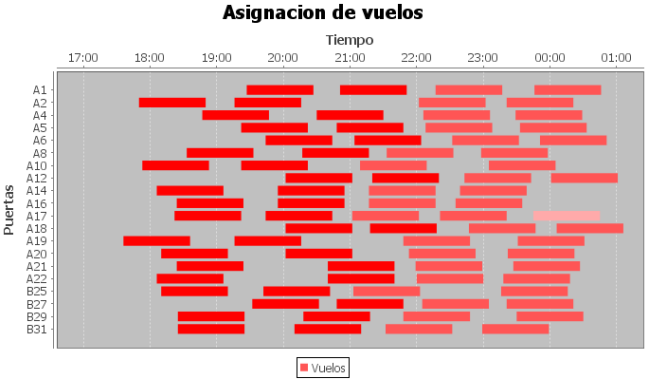

Fig. 4. Gráfico de los resultados obtenidos en la solución final.

5 Conclusión

En primer lugar, se ha recopilado la información necesaria para evaluar la aplicación. Se ha implementado el algoritmo de búsqueda tabú. Se ha realizado una presentación grafica de los resultados obtenidos. Se ha conseguido minimizar la función objetivo considerando las restricciones del problema obteniendo una buena solución para los vuelos en conflicto y los trasbordos de los pasajeros. Se ha hecho un análisis del rendimiento de la aplicación obteniendo resultados favorables. Por todo lo anterior se puede concluir que se ha logrado cumplir el objetivo de crear un sistema de asignación de vuelos a las puertas de un aeropuerto.

Como trabajo futuro se pueden realizar las siguientes mejoras:

- Realizar la asignación de puertas aplicando otras heurísticas de ordenación de los vuelos o las puertas de embarque.
- Implementar otros algoritmos de búsqueda local como el algoritmo de recocido simulado y comparar los resultados para ver con cual se obtiene un mejor rendimiento.
- Implementar la solución actual mediante software especializado en programación con restricciones.
- Crear una interfaz gráfica de usuario en donde se puedan modificar parámetros del algoritmo como el tamaño de la lista tabú o el número de iteraciones y otros atributos relacionados con los vuelos, puertas o aviones.

AGRADECIMIENTOS

Me gustaría expresar mi gratitud a Robert Benavente, tutor del proyecto por sus consejos y recomendaciones en este trabajo.

BIBLIOGRAFÍA

[1] Wipro Technologies. (2009). Gate Assignment Solution (GAM) Using Hybrid Heuristics Algorithm. Draft technical white paper on gate assignment solution. Team Airline Innovation Centre of Excellence, Wipro Technologies.
[Accedido el día 25 de febrero de 2014].

[2] Andreas Drexl and Yury Nikulin. Multicriteria airport gate assignment and Pareto simulated annealing. IEE Transactions, Vol. 40(4): 385-397. Apr 1, 2008
[Accedido el día 25 de febrero de 2014].

[3] H Ding, A Lim, B Rodrigues and YZhu. "New heuristics for over-constrained flight to gate assignments".Journal of the Operational Research Society (2004), Vol. 55(7): p. 760-768. Jul, 2004.
[Accedido el día 26 de febrero de 2014].

[4] Sang Hyun Kim, Eric Feron, John-Paul Clarke, Aude Marzuoli, Daniel Delahaye. Airport Gate Scheduling for Passengers, Aircraft, and Operations. Tenth USA/Europe Air Traffic Management Research and Development Seminar (ATM 2013).
[Accedido el día 27 de febrero de 2014].

[5] Soi-Hoi Lam, Jia-Meng Cao and Henry Fan. Development of an intelligent agent for airport gate assignment. Journal of Air Transportation; 2002, Vol. 7 Issue 2, p 103. June 2002
[Accedido el día 28 de febrero de 2014].

[6] S. H. Kim, E. Feron, and J.-P. Clarke, "Gate assignment to minimize passenger transit time and aircraft taxi time," AIAA J. of Guidance, Control, and Dynamics, vol. 36, no. 2, pp. 467-475, 2013.
[Accedido el día 3 de marzo de 2014].

[7] H. Ding, A. Lim, B. Rodrigues and Y. Zhu. The Airport Gate Assignment Problem. Computers & Operations Research, vol. 32, p.1867-1880. 2003.
[Accedido el día 4 de marzo de 2014].

[8] Salidas y llegadas de vuelos del aeropuerto de Barcelona- El Prat.
<http://www.aeropuertos.net/aeropuerto-de-barcelona-llegadas-de-vuelos/>
[Accedido el día 26 de marzo de 2014]

[9] Aviones comerciales
<http://turinegocios.com/cms/front_content.php?idart=182>
[Accedido el día 27 de marzo de 2014]

[10] Mapa del aeropuerto de Barcelona – El Prat.
<http://www.barcelona-tourist-guide.com/sp/aeropuerto/mapas/indicaciones-para-ir-en-automovil-desde-el-aeropuerto-de-barcelona-hasta-el-centro-de-la-ciudad.html>.
[Accedido el día 28 de marzo de 2014]

[11] Librería de gráficos para Java, JFreeChart.
< http://www.jfree.org/jfreechart/>.
[Accedido el día 2 de junio de 2014]