UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN

Diseño, evaluación y comparación de técnicas de solución para el problema en inteligencia artificial: «Airline Crew Scheduling Problem»

T E S I S

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN TELECOMUNICACIONES, SISTEMAS Y ELECTRÓNICA

P R E S E N T A

Hernández Gastaldi Ernesto

Asesor: Ing. José Luis Barbosa Pacheco

Cuautitlán Izcalli, Estado de México 2020

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

VNIVERSIDAD NACIONAL AVPNOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN
SECRETARÍA GENERAL
DEPARTAMENTO DE EXÁMENES PROFESIONALES

U. N. A. M.
FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN

ASUNTO: VOTO APROBATORIO

M. en C. JORGE ALFREDO CUÉLLAR ORDAZ
DIRECTOR DE LA FES CUAUTITLAN
PRESENTE

ATN: I.A. LAURA MARGARITA CORTAZAR FIGUEROA
Jefa del Departamento de Exámenes Profesionales
de la FES Cuautitlán.

Con base en el Reglamento General de Exámenes, y la Dirección de la Facultad, nos permitimos comunicar a usted que revisamos el: **Trabajo de Tesis**

Diseño, evaluación y comparación de técnicas de solución para el problema en inteligencia artificial: «Airline Crew Scheduling Problem»

Que presenta el pasante: **ERNESTO HERNÁNDEZ GASTALDI**
Con número de cuenta: **31226381-6** para obtener el Título de la carrera: **Ingeniería en Telecomunicaciones, Sistemas y Electrónica**

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el **EXAMEN PROFESIONAL** correspondiente, otorgamos nuestro **VOTO APROBATORIO**.

ATENTAMENTE
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Cuautitlán Izcalli, Méx. a 28 de enero de 2020.

PROFESORES QUE INTEGRAN EL JURADO

	NOMBRE	FIRMA
PRESIDENTE	Mtro. Jorge Buendía Gómez	
VOCAL	Ing. José Luis Barbosa Pacheco	
SECRETARIO	Dr. David Tinoco Varela	
1er. SUPLENTE	Ing. Maria Guadalupe Vázquez Salazar	
2do. SUPLENTE	Mtra. Judith Mayte Flores Pérez	

NOTA: los sinodales suplentes están obligados a presentarse el día y hora del Examen Profesional (art. 127).

LMCF/ntm*

Agradecimientos

A mis papás, **Ernesto** y **Griselda**:

Por apoyarme en todas las decisiones que he tomado a lo largo de mi vida. Gracias por permitirme elegir la carrera que deseaba, aun cuando esto implicaba viajar 5 h diarias entre ir y volver, además de un gran gasto económico. Todos mis logros se los debo y dedico a ustedes, pues, a pesar de todo, siempre hemos logrado salir adelante.

A la familia **Gastaldi**:

Por ser quienes siempre han estado presentes en los buenos y malos momentos. Aunque somos una familia pequeña, cada uno de nosotros sabemos perfectamente que podemos contar con los demás en el momento que nos necesitemos.

A mi «main», **Mauricio Byrd**:

Por ser mi mejor amigo desde el principio de la carrera. Gracias por el incontable apoyo que recibí de tu parte; siempre que tuve preguntas sobre cualquier tema, estuviste allí para ayudarme. Cada semestre, sin duda, no hubiera sido lo mismo sin tu hermandad.

A la «Máxima Casa de Estudios», la **UNAM**:

Por formarme académicamente desde el bachillerato en la mejor universidad de México. Y por concederme, además, la oportunidad de cursar un semestre en el extranjero, dándome la inspiración clave y el conocimiento necesario para la realización de este trabajo.

Los Andes, Chile. 5 agosto 2018.

Índice

Introducción

La inteligencia artificial (IA) es una rama de la informática que está dedicada a la creación artificial de conocimiento; en otras palabras, es una ciencia cuyo objetivo principal es el diseño y proporción de algoritmos computacionales inteligentes, los cuales imiten el comportamiento de los seres humanos en el sentido de que sean capaces de adaptarse, aprender, recordar, y tomar decisiones [1].

En sus orígenes, la IA nació como una herramienta centrada en resolver juegos, así como problemas de propósito general. Los primeros algoritmos inteligentes que fueron creados eran precisamente enfocados en jugar algún juego, siendo el ajedrez y las damas los primeros puestos a prueba. Por ejemplo, uno de los primeros algoritmos, que era capaz de jugar a las damas, fue programado de tal manera que su creador puso a jugar dos réplicas del algoritmo, una contra la otra, para aprender mutuamente de los movimientos que realizaban, y así mejorarse [2].

Hoy en día, la inteligencia artificial se ha popularizado exponencialmente, es objeto de estudio y gran parte de la tecnología actual, tanto que ha llegado al punto de convertirse en un símbolo de marketing para la venta de distintos productos [3]. Esto ha provocado que ya no sea sólo estudiada por el ámbito científico-tecnológico, sino por el filosófico-social, argumentando de manera principal si realmente es «inteligencia» de la que se trata, o de sólo una serie de algoritmos que dan la ilusión de serlo; hasta la actualidad, es un debate que sigue abierto a discusión [4].

Los problemas de «scheduling» tratan, en forma general, de organizar temporalmente un conjunto de operaciones que forman parte de los servicios que ofrece una compañía, la organización de estos servicios está sujeta a una serie de restricciones que varían dependiendo de cada caso; el objetivo de resolver el problema es optimizar la o las funciones objetivo, las cuales se pueden centrar en ganancias, tiempos, o destinos, según lo que se necesite priorizar [5].

Para el caso del problema de planificación de tripulaciones de avión, una variante más del problema de «scheduling», consiste en organizar un conjunto de tripulaciones que se encargarán de cubrir una serie de vuelos que ya están previamente programados en un horario para cierto intervalo de tiempo; este periodo puede ser desde un día, una semana, o incluso un mes completo. Las restricciones pueden ir desde lo más básico que cualquier persona podría obviar,

hasta restricciones que se enfoquen en las características particulares de cada uno de los que forman las tripulaciones [6] [7].

Los sistemas inteligentes diseñados se encargan de encontrar soluciones al problema de acomodo de tripulaciones para una serie de vuelos que cuenten con un identificador (ID), origen, destino, hora de salida, y hora de llegada. Tales algoritmos asignan los vuelos de tal manera que se cumplan ciertas restricciones: las tripulaciones que cubrirán los vuelos deben, idealmente, partir de una base (aeropuerto) y regresar a esta misma en el último vuelo que se les asigne cubrir; no deben exceder de cierta cantidad de horas de trabajo; tiene que haber un tiempo mínimo y máximo de espera entre la llegada de un vuelo y la salida del siguiente; y la tripulación no debe pasar más de cierta cantidad de horas en el aire. El incumplimiento de alguna de las condiciones incrementa el costo de la solución como penalización, o directamente la descarta, dependiendo de la dureza de la restricción en cuestión. Las inteligencias artificiales son puestas a prueba mediante distintos horarios de vuelos, con el fin de ver su comportamiento ante variaciones en horarios, cantidad de vuelos, parámetros de las restricciones, etcétera; obteniendo así las características de cada una, y realizando comparaciones entre el desempeño de ambas.

Objetivos

Objetivo general:

Documentar el diseño y programación de dos técnicas de inteligencia artificial mediante algoritmos en lenguaje C, con el fin de que tales programas encuentren soluciones al problema de «Airline Crew Scheduling»; de esta manera, determinar el rendimiento de cada uno de ellos mediante distintas pruebas y comparativas entre sus respectivos funcionamientos.

Objetivos específicos:

- Diseñar un primer programa, el cual utilice técnicas de resolución de tipo completa, que sea capaz de encontrar soluciones al problema; plantearlo como un algoritmo y programarlo.
- Diseñar un segundo programa con una técnica de resolución de tipo incompleta, para el mismo propósito; plantearlo como algoritmo y posteriormente programarlo.
- Realizar pruebas de funcionamiento con distintos horarios de vuelo en ambos algoritmos.
- Realizar cambios en los valores de los parámetros de las restricciones y de los algoritmos, con el fin de ver su efecto en el desempeño de los programas.
- Analizar los resultados obtenidos y compararlos, para así observar en qué circunstancias actúa mejor o peor un tipo de técnica sobre el otro.

1. Conceptos generales sobre inteligencia artificial

En el presente capítulo se detalla el significado de la frase «Inteligencia Artificial» (IA); así como un breve resumen a manera de línea temporal sobre la historia de esta ciencia, proporcionando la explicación de algunos de los proyectos más importantes que se han realizado. De igual manera, se describe el estado actual en el que se encuentra el área tanto del lado del software como del hardware; los diferentes tipos de inteligencia artificial; la definición de problemas de optimización para sistemas inteligentes; y un recuento sobre las distintas técnicas de solución para problemas de optimización, estas últimas con énfasis de manera particular al propósito de este trabajo.

1.1 Introducción al significado

Comúnmente se ha dicho que la filosofía es considerada la madre de todas las ciencias, puesto que la curiosidad y el interés del ser humano de conocer el porqué de lo que está ocurriendo a su alrededor, han sido las principales razones de la mayoría de inventos y descubrimientos que la humanidad ha realizado. Desde siempre, los filósofos se han preguntado cómo es que funciona la mente humana, y si cualquier entidad que no sea humana puede tener una mente. Es una discusión cuya respuesta siempre dependerá de la persona que la responda. Algunos han optado por aceptar que las máquinas son o serán capaces de hacer lo mismo que los humanos, mientras que otros claman que la mentalidad humana siempre irá más allá de lo que cualquier máquina sea capaz de lograr [8].

Lo cierto es que hoy en día contamos con máquinas que atrevemos a llamarles «inteligentes», pero el significado de inteligencia es un concepto que también se encuentra a discusión abierta, es una palabra que no tiene un significado fijo, este dependerá directamente de la fuente que se consulte. Por ejemplo, el Diccionario de la Lengua Española (DLE) de la Real Academia Española (RAE) recoge, entre otras, estas 4 definiciones [9]:

1. f. Capacidad de entender o comprender.
2. f. Capacidad de resolver problemas.
3. f. Conocimiento, comprensión, acto de entender.
4. f. Sentido en que se puede tomar una proposición, un dicho o una expresión.

Algo en común que se puede observar en las definiciones mencionadas es que, para realizar cualquier acción que muestre inteligencia, debe existir la capacidad de poder pensar. Así que la siguiente interrogante viene inmediatamente después: «¿las máquinas son capaces de pensar?»; esta pregunta se ha puesto a discusión desde los orígenes del concepto de inteligencia artificial. La respuesta, como era de suponerse, no es un simple sí o no, sino que es una respuesta difusa y muchas veces contradictoria [8].

Los motivos anteriormente mencionados conllevaron a la creación de una frase que definiera el «comportamiento» de una máquina, el cual fuera capaz de asemejarse, o de al menos dar la ilusión de hacerlo, al comportamiento que tenemos los seres humanos, en el sentido de que sea capaz de razonar, tomar decisiones y recordar, dependiendo de las circunstancias que se le presenten; esta, precisamente, es la definición de inteligencia artificial. El DLE ofrece el siguiente concepto: «*Disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico*» [9]; si bien este se enfoca más a la disciplina en sí que al sustantivo, el concepto es básicamente el mismo. Actualmente es usado con significativa frecuencia para la tecnología y se ha vuelto, a la par que una disciplina de estudio de muy alta importancia, un objeto de marketing para el anuncio de diversos productos de tecnología [3].

1.2 Historia

Desde los orígenes de la computación, los investigadores en el área se crearon inmensas expectativas acerca de lo que las computadoras serían capaces de realizar, se plantearon una enorme cantidad de proyectos que querían utilizar a las computadoras como agentes de resolución para todo tipo de problemas, desde que fueran capaces de jugar a algún juego contra cualquier ser humano y ganarle, hasta incluso máquinas que fueran capaces de resolver cualquier tipo de problema de propósito general.

Lo cierto fue que siempre existieron limitaciones tanto del lado del hardware como del software, las cuales dejaron a muchos proyectos como inconclusos o inviables, es decir, sólo fueron planteados de forma teórica, y estos terminaron siendo descartados o retomados en algún momento futuro cuando ya no había impedimentos que obstruyeran el desarrollo práctico de los mismos. En sus inicios, la capacidad de cómputo de las máquinas era muy reducida en

comparación con las ideas que se tenían, así que, el avance de la tecnología, específicamente en el mundo de la computación, permitió continuar con varios de los proyectos inviables en un inicio. Tal como lo indicaba la Ley de Moore: los transistores, principal componente de los circuitos integrados, se duplicaron en cantidad cada dos años durante varias décadas; haciendo que de igual manera los avances en computación mejoraran de forma exponencial [10].

El primer trabajo en ser conocido como parte del área de inteligencia artificial fue en 1943 por Warren McCulloch y Walter Pitts, quienes hicieron el primer modelo teórico de las neuronas del cerebro humano. Plantearon un modelo matemático en donde cada neurona podía estar en dos posibles estados: encendida o apagada; y que neuronas simples eran capaces de aprender. En su trabajo demostraron de manera teórica que su modelo de red neuronal era equivalente a la máquina de Turing, y que cualquier función computable era capaz de realizarse con algún tipo de red neuronal. Sin embargo, en la parte práctica se dieron cuenta de que su representación binaria del estado de una neurona no era correcta ni viable, sino que las neuronas eran entidades con características que las alejaban de tener un comportamiento mínimamente lineal. De cualquier manera, aun con el planteamiento erróneo de los estados en binario, se considera que Warren McCulloch y Walter Pitts fueron quienes sentaron la base de las redes neuronales [11].

En el año de 1950, uno de los trabajos más importantes que se han realizado en inteligencia artificial vio la luz. Alan Turing comenzó con la pregunta: «¿Las computadoras pueden pensar?», y partiendo de allí desarrolló todo un análisis. Propuso, a su vez, un juego como respuesta a la pregunta inicial: el juego de la imitación. Este consistiría en programar una computadora para que fuera capaz de entablar una conversación con una persona real, la persona sería puesta a platicar con otra real y con la computadora programada; el objetivo de la computadora sería intentar hacer creer a la persona que está hablando con ambas, de que la máquina es la persona real, y si lo consiguiera lograr, entonces podría considerarse como inteligente. El autor estimó que para el año 2000 esto se haría realidad; sin embargo, hasta hoy en día ninguna computadora ha sido capaz de ganar el juego [12].

Asimismo, en ese año, Claude Shannon publicó un trabajo acerca de una máquina capaz de jugar ajedrez. En este postuló que un juego típico de ajedrez podría involucrar aproximadamente 10^{120} posibles movimientos por realizar, y que, por lo tanto, si una computadora de tipo Von

Neumann podía examinar un movimiento cada microsegundo, tardaría $3x10^{106}$ años en hacer su primer movimiento. Concluyó su investigación con la necesidad de utilizar *heurísticas*[1] para la elección de los movimientos [13].

John McCarthy, quien fue el fundador del término «inteligencia artificial», presentó un trabajo en 1958, donde planteaba un programa llamado Advice Taker. Este se encargaba de buscar soluciones para problemas de propósito general basándose en axiomas simples iniciales, y permitiendo al usuario añadir más de ellos sin la necesidad de ser reprogramado. Este trabajo fue relevante debido a que es considerado el primer programa basado en conocimiento [14].

Para 1961, las ideas iniciales de McCulloch y Pitts fueron retomadas. Frank Rosenblatt publica un trabajo acerca de las redes neuronales en el que demuestra el teorema de la convergencia del perceptrón, el cual postula que el aprendizaje de una red neuronal se puede modificar mediante la variación de los pesos de sus conexiones entre sus neuronas [15].

Durante casi 10 años, desde 1961 hasta 1970, Allen Newell y Herbert Simon trabajaron en un programa que se encargara de resolver cualquier tipo de problema de propósito general, este se basaba en la técnica que es conocida como Means-Ends (MEA). Los autores postularon que cada problema puede ser representado en términos de una serie de estados, de los cuales, los estados que fueran una solución al problema eran llamados estados meta. Por lo tanto, esta técnica indica que, si el algoritmo se encuentra en algún estado y la solución se encuentra en el estado meta, el camino al estado meta puede ser descrito como una serie de acciones o cambios de estado, las cuales llevaban finalmente a la solución. Aunque teóricamente era posible, resultó inviable para resolver problemas complicados debido al incremento exponencial del número de estados, por lo que el proyecto terminó siendo dejado de lado [16].

En el año de 1965, Lotfi Zadeh publicó el trabajo que daría origen a lo que hoy se conoce como lógica difusa. Este tipo de lógica se centra en representar variables en distintos conjuntos, asignándoles un porcentaje o valor numérico mediante lenguaje informal [17].

[1] Las heurísticas, en inteligencia artificial, son reglas empíricas (*rules-of-thumb*) que ayudan en la búsqueda de la solución de un problema como parte del algoritmo, con el fin de encontrarla de manera más rápida o sencilla [18].

Como era de suponerse, las limitaciones del hardware y la capacidad de cómputo de aquel entonces fueron obstáculo para la mayoría de proyectos en el área de inteligencia artificial. Para la década de los 70, la gran parte de estos habían quedado abandonados y la euforia sobre esta área estaba prácticamente acabada. Las pocas aplicaciones exitosas eran centradas en juegos, pero realmente ninguna inteligencia artificial era capaz de lidiar con problemas del mundo real. Esto hizo que los aún interesados en la IA tuvieran que empezar a pensar en darle otro tipo de enfoque a la ciencia, y replantear la manera en que se estaban diseñando los algoritmos inteligentes.

Por ello fue que, en esa misma década, uno de los mayores avances que se dieron en inteligencia artificial fue el indispensable cambio de enfoque. Anteriormente, los investigadores se encontraban intentando resolver problemas grandes mediante métodos de resolución generales, en otras palabras, *métodos débiles* (ahora conocidos formalmente así) que deseaban que sirvieran para la resolución de problemas totalmente distintos, lo cual nunca terminó de funcionar. El cambio de enfoque consistió en darse cuenta de que, el dominio en el que los algoritmos inteligentes trabajaran tenía que ser lo suficientemente acotado para que estos funcionasen de forma exitosa, es decir, los sistemas inteligentes tenían que utilizar un razonamiento más profundo y enfocado de forma concreta en el problema a resolver, para que este fuera capaz de entregar resultados prácticos [8].

Uno de los proyectos que ayudaron a los investigadores a darse cuenta de la necesidad de un cambio de enfoque en esta área, fue el proyecto DENDRAL, realizado en el año 1969 por Edward Feigenbaum y Joshua Lederberg, además de ser apoyado por la NASA [19]. Este proyecto surgió de la idea de mandar una nave a Marte para analizar la tierra de aquel planeta mediante un espectrómetro de masa. Se necesitaba un programa que fuera capaz de encontrar la estructura molecular de la tierra marciana mediante los datos arrojados por el espectrómetro. Lo interesante del proyecto es que no existía ningún algoritmo científico para el mapeo de la estructura molecular, sino que solamente un químico con cierto nivel de experiencia, basándose en su conocimiento y habilidades, era capaz de reducir el dominio del problema a unas cuantas posibles soluciones para evaluar y corroborar, era Joshua Lederberg quien tenía el conocimiento para tal tarea. El reto consistió en diseñar un programa que incorporara el conocimiento y la experiencia de Lederberg para poder encontrar la estructura molecular. Los sistemas de tal tipo, por cierto, fueron formalmente nombrados como sistemas expertos.

Fiengenbaum, quien era el informático del equipo, se vio en la necesidad de adquirir algo de conocimiento en química y análisis espectral para poder entender las ideas y hablar el mismo vocabulario que Lederberg. Sin embargo, Lederberg no sólo utilizaba reglas formales de química, sino que tenía sus propias reglas empíricas y especulaciones basadas en su experiencia. Fiengenbaum de inmediato notó el problema mayor, el cual llamó «knowledge acquisition bottleneck», el problema de cómo transcribir conocimiento humano en lenguaje computacional. Para apoyarse mutuamente, Lederberg, de igual manera, tuvo que aprender las bases de computación y programación. Cabe mencionar que, en su trabajo, ellos también sentaron las bases de la ingeniería del conocimiento, la cual se centra en las técnicas de captura, análisis y expresión del conocimiento. El proyecto pudo considerarse exitoso, aunque, en su presentación escrita oficial, los mismos autores afirman que el programa no es perfecto, y que funciona mejor o peor para algunas familias de moléculas que para otras [8] [19].

En el año de 1976, John Holland introdujo el concepto de algoritmos genéticos en su trabajo «Adaptation in Natural and Artificial Systems». En él, hace una extrapolación de la teoría de la evolución de Charles Darwin hacia la computación; tal teoría menciona que sólo los individuos mejor adaptados son aquellos que sobreviven a lo largo del tiempo, y estos mismos son quienes tienen la mayor posibilidad de reproducirse y heredar su material genético a las siguientes generaciones [20]. Los algoritmos genéticos, por ende, aplican el mismo concepto y simulan una población de individuos, evalúan la calidad de su funcionamiento para esa población en concreto, y después generan otra población mediante la mutación y el cruzamiento de los individuos de la generación anterior, tal como ocurre en la naturaleza. Los mismos principios de la teoría de la evolución se aplican para los algoritmos, donde los seres que mejor funcionamiento tengan, son aquellos que tendrán más probabilidad de reproducirse y heredar su material genético, mientras que los de peor desempeño serán los que tendrán poca o nula probabilidad de seguir existiendo [21].

Para la década de los 80, los investigadores de IA decidieron retomar el área de las redes neuronales artificiales (RNA), pues, como se menciona anteriormente, gracias a los trabajos de McCulloch y Frank Rosenblatt, las bases de las redes neuronales ya estaban claramente establecidas, pero no fue sino hasta esta década que la capacidad de cómputo era suficiente para retomar y experimentar en esta área. Se realizaron importantes contribuciones en el área de redes

neuronales, por ejemplo: John Hopfield, en su trabajo «Neural Networks and Physical Systems with Emergent Collective Computational Abilities» introduce lo que hoy se conoce como las redes neuronales tipo Hopfield, las cuales también se llaman recurrentes por estar basadas en la retroalimentación como parte de su aprendizaje [22]; de igual manera, el algoritmo de aprendizaje conocido como «back-propagation», el cual es uno de los más importantes para esta área, fue planteado por David Rumelhart y James McClelland en su trabajo «Parallel Distributed Processing: Explorations in the Microstructures of Cognition» [23]. Desde entonces y hasta la actualidad, las RNA junto con las neurociencias aplicadas a la computación, volvieron a ser objetos de estudio para los investigadores.

Posteriormente, de la década de los 90 en adelante, los investigadores se centraron en la computación con palabras, es decir, hubo mucho énfasis en el área de la ingeniería del conocimiento. La lógica difusa, si bien ya había sido anteriormente planteada por Lotfi Zadeh, no había sido de gran interés hasta esos tiempos. Se descubrió que los sistemas difusos poseían gran cantidad de ventajas y fueron objeto de estudio no sólo para los investigadores de inteligencia artificial, sino también de otras áreas, como la ingeniería de control [8].

A la lógica difusa se le comenzó a integrar como una rama de la inteligencia artificial debido a la manera en que funciona: los sistemas poseen entradas y salidas, las entradas pueden ser algún sensor, y la salida un actuador; la lógica difusa postula que el rango de valores posible de las entradas se puede dividir en conjuntos difusos, de tal manera que estos conjuntos pueden definir el valor de la variable mediante lenguaje informal. Si suponemos que una entrada es la temperatura, se puede dividir en los conjuntos: helada, fría, templada, caliente, e hirviendo; mientras que los valores que puede tomar cada conjunto podrían ser: muy, medio, y poco. Cada conjunto y valor tendría un rango de valores o porcentaje numérico en donde la temperatura del agua encaje. Lo mismo podría aplicarse para cuales fueran las salidas del sistema, y todos estos valores se utilizan para poder determinar el valor de las salidas mediante lenguaje no técnico ni matemático. Esto sirve, entre otros propósitos, para traspasar el conocimiento de un operador humano hacia un controlador automático que imite sus operaciones utilizando tal lógica. Como se observa, la lógica difusa se basa en gran parte en la ingeniería del conocimiento [24].

1.3 Actualidad

En el presente, la inteligencia artificial es una ciencia formalmente establecida y es una rama primordial para toda el área de ciencias de la computación. La evolución exponencial de la tecnología y su estado actual han impulsado significativamente la cantidad de aplicaciones que se le dan hoy en día. Se tienen enorme cantidad de productos y proyectos que utilizan algún tipo de inteligencia artificial, y cada vez es más frecuente escuchar su nombre como parte de la descripción de la tecnología cotidiana: refrigeradores inteligentes, lavadoras inteligentes, teléfonos inteligentes, etc.

La inteligencia artificial, además, forma parte de los servicios que distintas compañías ofrecen o utilizan: asistentes personales, videojuegos, conducción autónoma, análisis de datos, resolución de problemas de optimización, entre muchos otros. Es importante resaltar que, a su vez, se ha convertido en una herramienta de apoyo para distintas áreas de investigación como química, biotecnología, medicina, cartografía, traducción, construcción, marketing, etc. En la actualidad, los sistemas inteligentes ya se observan de manera cotidiana en la mayoría de áreas de investigación como parte clave, o como una herramienta complementaria [25].

Por la parte del hardware y la electrónica, la IA ha orillado a algunas de las compañías que diseñan componentes electrónicos para sistemas computacionales, a incluir hardware especializado en realizar operaciones que los algoritmos inteligentes utilicen con frecuencia, esto con el objetivo de acelerar los cálculos que necesiten para cualquier tarea que se encuentren ejecutando. Como un ejemplo reciente, las tarjetas gráficas Nvidia de la arquitectura Turing (Serie 2000) incluyen hardware dedicado para la tecnología «Deep Learning Super Sampling» (DLSS), la cual utiliza redes neuronales para ajustar la escala de las imágenes de una resolución menor, a una resolución mayor, sin perder apenas calidad en comparación con una imagen a la resolución mayor nativa, con poca o nula pérdida de rendimiento en la tarjeta gráfica gracias al hardware dedicado [26].

De igual manera, las compañías más grandes de tecnología como Google, Microsoft y Facebook, tienen servidores completos dedicados al análisis de grandes cantidades de datos, sobre las diferentes actividades de sus usuarios mediante inteligencia artificial, lo cual, por cierto, se

conoce como «big data» (macrodatos, en español). El objetivo del análisis de los macrodatos es obtener diferentes estadísticas acerca de tendencias, preferencias, e información en general sobre los servicios que cada una de ellas ofrece [27].

Si bien la inteligencia artificial es una ciencia perfectamente establecida en la actualidad, las distintas ramas en la que esta se divide no lo están del todo. Cada referencia consultada maneja una clasificación diferente a las demás; algunos basan sus clasificaciones en la capacidad de memoria, otros en el método de resolución que utilice, y otros en el nivel de consciencia que el sistema inteligente posea; incluso muchas de las clasificaciones que se le dan en ocasiones no son del todo correctas, debido a que estas son derivadas o combinaciones de otras, por ello es que no existe un listado exacto sobre los tipos de IA que existen [28] [29] [30]. Sin embargo, hay por lo menos tres clasificaciones que están presentes en la mayoría de listados de los autores, estas son: redes neuronales, lógica difusa y sistemas inteligentes.

- Redes neuronales artificiales: las redes neuronales artificiales son algoritmos computacionales que imitan el comportamiento de las neuronas del sistema nervioso. Una red se forma mediante conexiones de distintas neuronas agrupadas en capas; su aprendizaje se basa en la relación de la fuerza de sus conexiones con sus funciones matemáticas de activación. Algunos de los usos cotidianos de las redes neuronales son: reconocimiento de patrones, reconstrucción de señales, predicción de comportamientos, entre otros [31].
- Sistemas con lógica difusa: son aquellos sistemas que utilizan la clasificación de sus entradas y salidas en distintos conjuntos difusos; dichos conjuntos, aunque formalmente tienen asignado un valor numérico o porcentual, son expresados en lenguaje verbal informal, y sirven para facilitar el diseño de controladores de los cuales no se conoce de manera exacta el modelo matemático, pero se conocen las salidas que debe tener el sistema para las distintas combinaciones de entradas, este conocimiento puede provenir, por ejemplo, de experiencia meramente humana [24].
- Sistemas inteligentes: los sistemas inteligentes son algoritmos computacionales que tienen como objetivo encontrar la mejor solución posible para un problema en concreto. Regularmente son algoritmos hechos a medida, ya que están programados para resolver exactamente un problema, aunque pueden tener cierto grado de libertad, siempre y

cuando la base de este sea la misma. Los problemas que pueden resolver tienen, por lo menos, una función objetivo, la cual se desea maximizar o minimizar, según sea el caso, y una serie de restricciones o condiciones. La calidad de la solución entonces dependerá de si lo que se desea es minimizar o maximizar; aunque también existen problemas en donde cualquier solución es igual de buena que otra, como podría ser el caso de resolver un sudoku [32].

1.4 Problemas de optimización para sistemas inteligentes

Existen diferentes métodos para realizar la búsqueda de la solución de un problema, pero antes de comenzar a resolverlo, se debe entender a la perfección el problema a tratar, las partes que lo componen, y su relación con el método que se elegirá para resolverlo. Un problema de optimización consta de las siguientes partes [32] [33]:

- Constantes o parámetros: son aquellos valores que están previamente definidos y que no se modifican en ningún momento durante la búsqueda de la solución. Sirven como parte de las restricciones, como guía para la búsqueda de solución, o como relación directa al costo de cualquier solución encontrada. Por ejemplo: el precio de las distintas partes de un automóvil.
- Variables: son aquellas incógnitas a las cuales el algoritmo debe asignar sus respectivos valores, mismas que, en conjunto, forman la solución encontrada por la IA. Por ejemplo: el color del parachoques, color del chasis, color del techo, etc.
- Dominio de las variables: aquellos valores que cada una de las variables puede tomar, es decir, un conjunto de valores numéricos o verbales que pueden ser iguales o diferentes para cada variable. Por ejemplo: rojo, rosa, negro, blanco.
- Restricciones o condiciones: son aquellas relaciones entre las variables que están estricta o parcialmente prohibidas. Aquellas restricciones que son estrictas se les conoce como restricciones duras, si se incumple una de ellas, la solución queda descartada inmediatamente; mientras que las restricciones parciales son conocidas como restricciones blandas, estas pueden disminuir la calidad de la solución, pero no la descartan, solamente se sufre una penalización. Por ejemplo: el parachoques debe ser más claro que la carrocería; el techo debe ser más claro que la carrocería; la carrocería debe

tener el mismo color que el chasis; las puertas preferentemente deben ser más claras que el capó; etc.

- Función(es) objetivo: si una solución del problema tiene manera de ser mejor o peor que otra, las funciones objetivo son funciones matemáticas que involucran a las variables, y que se desean maximizar o minimizar, dependiendo de cada caso. No obstante, si todas las soluciones son igualmente válidas y ninguna es peor o mejor que otra, entonces, regularmente, la función objetivo no existe como tal, sino que se convierte en únicamente el cumplimiento de las mismas restricciones del problema.
- Espacio de búsqueda: matemáticamente, el espacio de búsqueda es el producto de todos los dominios de las variables del problema. Geométricamente, el espacio de búsqueda es un plano multidimensional en donde cada eje es cada una de las variables. Verbalmente, representa la cantidad completa de todas las posibles combinaciones que pueden existir entre los valores de las variables, por ende, en él se encuentran todas aquellas que son soluciones posibles, así como aquellas que no lo son. Dentro del espacio de búsqueda existen óptimos locales, es decir, soluciones de buena calidad, pero que no son la mejor solución posible; mientras que a las mejores soluciones del problema se les conoce como óptimos globales.[2]

1.5 Técnicas de solución para problemas de optimización

Las técnicas de solución son métodos algorítmicos para realizar el análisis, exploración y explotación del espacio de búsqueda, se dividen en dos grandes tipos [33]:

- Técnicas completas: son aquellas técnicas que revisan todo el espacio de búsqueda, por lo tanto, tienen la posibilidad de encontrar los óptimos globales dentro de él.
- Técnicas incompletas: se caracterizan por no revisar por completo el espacio de búsqueda, por ende, no garantizan llegar al óptimo global del problema; encuentran óptimos locales que pueden ser cercanos a este. Además, tienen la característica de ser más rápidas en ejecución comparadas con las técnicas completas.

[2] Si no se conocen las mejores soluciones al problema, nunca se tendrá total claridad sobre si la respuesta encontrada fue un óptimo global. Sin embargo, se sabe que una solución fue un óptimo local cuando anterior o posteriormente se encuentran soluciones mejores a esta.

En el libro «How to solve it: Modern Heuristics» [34] y en la mayoría de referencias consultadas, se menciona que, al enfrentar un problema, lo más conveniente es utilizar inicialmente una técnica completa, por el hecho de que estas exploran el espacio de búsqueda por completo. Si en determinado tiempo no se ha encontrado una solución de buena calidad, se recurre a utilizar una técnica incompleta, pues estas entregan óptimos locales que, aunque no son la mejor solución, pueden resultar más convenientes por cuestiones de tiempo. La figura 1.5.1 muestra un cuadro sinóptico sobre las técnicas de solución de problemas, así como algunos ejemplos de ellas. Las técnicas resaltadas son las de interés para este trabajo:

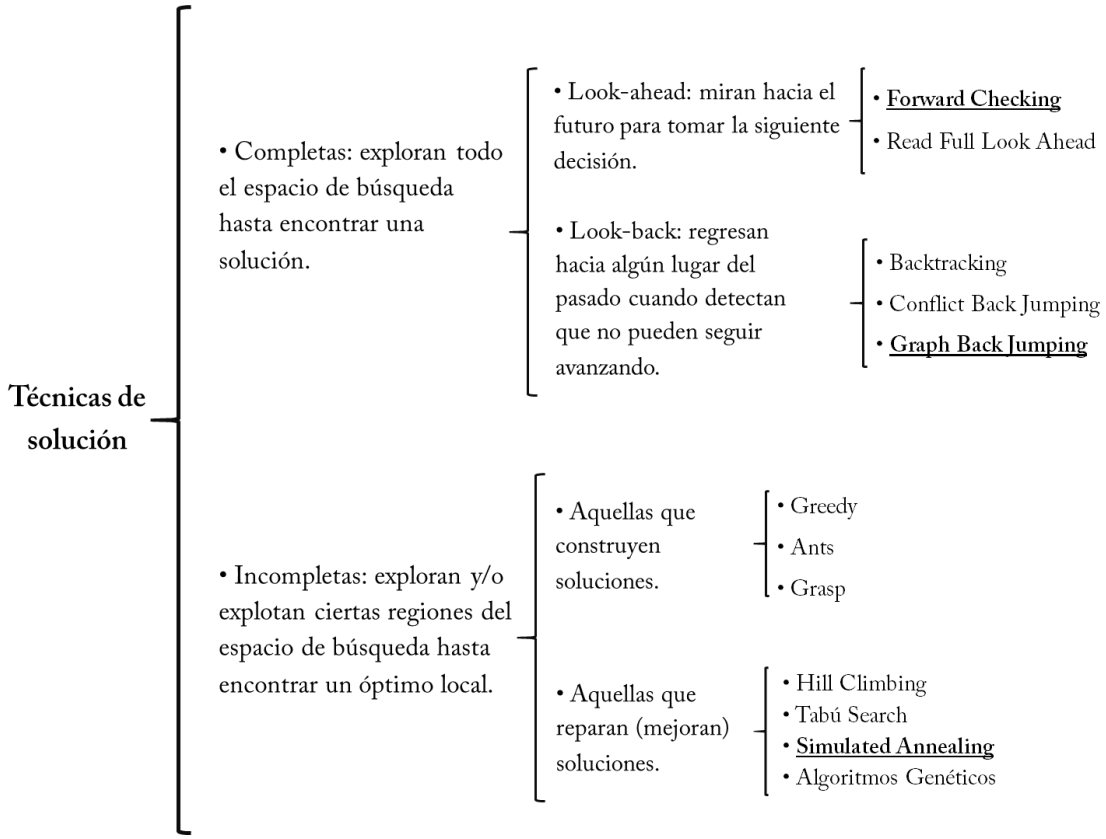


Figura 1.5.1. Las distintas técnicas de solución para problemas de optimización. Fuente: elaboración propia.

Es relevante mencionar que muchas de las distintas técnicas pueden complementarse y ayudarse unas a otras. En el caso de las técnicas completas, por ejemplo, una técnica de tipo «look-ahead» puede verse ayudada por una de tipo «look-back» para no tener que empezar la construcción de la solución desde cero si se viese en la necesidad de reiniciar. Para el caso de las incompletas, una

técnica que repare soluciones puede obtener una solución inicial mediante el uso de alguna técnica que construya, o incluso de una técnica completa.

Forward Checking (FC)

Es una técnica que funciona como una especie de filtro. Filtra los dominios de las variables basándose en el estado actual del algoritmo. Elimina cualquier valor de las variables que incumpla alguna de las restricciones del problema, dejando así, únicamente a aquellos valores del dominio de las variables que sean factibles en determinado momento para continuar con la construcción de la solución [35].

Graph Back Jumping (GBJ)

Las técnicas de «back jumping» son bastante parecidas a «backtracking», con la diferencia de que los «back jumping» vuelven a distintos lugares del pasado según sea su funcionamiento, mientras que «backtracking» siempre vuelve al estado inmediato anterior. En este caso, GBJ utiliza las relaciones (restricciones) entre variables para elegir hacia qué momento del pasado volver. Supóngase que una variable **Y** se encuentra relacionada por alguna restricción cualquiera con las variables **N** y **K**; el algoritmo se encuentra sobre la variable **Y** habiendo pasado previamente por **K**, **N**, **P** y **Q**; y se da cuenta de que estando allí, ya no puede avanzar (está atascado). Entonces, el GBJ lo regresará al momento donde estaba trabajando sobre la variable **N**, al ser esta la variable más cercana con la que **Y** se encuentra relacionada, y probar cambiando el valor asignado a **N** para así irse por otro camino [36]. La figura 1.5.2 es una sencilla ilustración acerca del funcionamiento del GBJ:

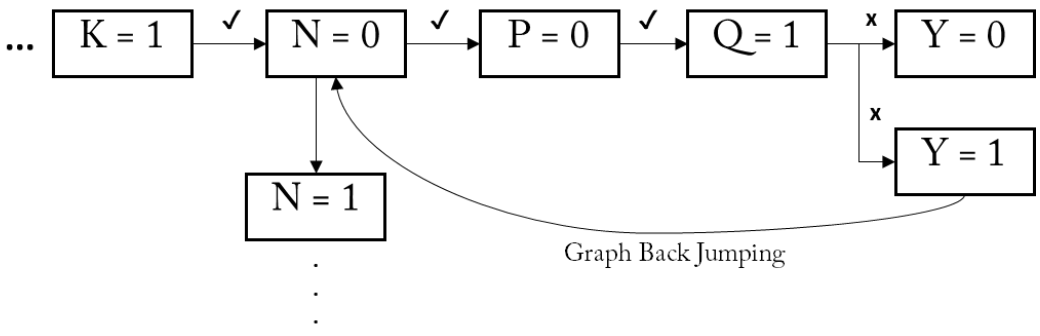


Figura 1.5.2. Gráfico del funcionamiento de la técnica Graph Back Jumping. Fuente: elaboración propia.

Simulated Annealing (SA)

Es una técnica incompleta que tiene la característica de permitirse explorar en un principio, disminuyendo gradualmente la exploración hasta quedarse en un lugar del espacio de búsqueda y comenzar a explotarlo. Tiene su base en la termodinámica, específicamente en la maleabilidad de los metales a altas temperaturas. Como es sabido, los metales a gran temperatura son perfectamente moldeables y pueden tomar la forma que se desee; mientras su temperatura disminuye, su maleabilidad también lo hace, y estos se van volviendo más rígidos y por ende menos flexibles y difíciles de trabajar. SA sigue el mismo principio, su funcionamiento es el siguiente [37]:

1. Se parte de una solución inicial (SI), así como un valor alto inicial de temperatura T.

2. Se modifica SI mediante algún movimiento en los valores de las variables de la solución, generando una solución futura (SF). Siempre se realiza un solo movimiento a la vez.

3. Si la SF es mejor que la SI (según sea el caso de minimizar o maximizar), SF se convierte en la nueva SI, se disminuye la temperatura en algún porcentaje y se vuelve al paso 2.

4. Si la SF no es mejor que la SI, es cuando entra en juego la temperatura:

5. Se calcula una probabilidad de elección mediante la operación $P = e^{\frac{-\Delta f}{T}}$, donde $\Delta f = costoSF - costoSI$, y T es la temperatura actual. Además, se genera un número real aleatorio entre 0 y 1, puesto que el valor de P también siempre estará entre 0 y 1.

6. Si la probabilidad es mayor que el número aleatorio generado, la SF se acepta y se convierte en la nueva SI; en caso contrario, la SI no se modifica y se vuelve al paso 2, realizándole una modificación diferente a la anterior. Se disminuye la temperatura en un porcentaje después de cualquiera de ambos casos.

Es de notarse que el exponente de la probabilidad tiene como denominador la temperatura, la cual, entre más pequeña sea, el valor de la probabilidad también lo será. Mencionándolo de una manera distinta, cuando la temperatura sea alta, la probabilidad de que se acepte la solución peor

será alta también, pero conforme la temperatura va disminuyendo, la probabilidad también lo hace hasta llegar a un punto en que el algoritmo nunca aceptará soluciones que sean peores. De esta manera es como se le permite al algoritmo explorar por varias zonas del espacio de búsqueda en un principio, para posteriormente quedarse en una de ellas a explotarla lo más posible. La aceptación de soluciones peores es el símil directo con la «flexibilidad» del metal a temperaturas altas.

2. El problema de planificación de tripulaciones (Airline Crew Scheduling Problem)

El problema de «scheduling» ha estado presente desde hace décadas en los medios de transporte. Aunque para este trabajo va referido especialmente al acomodo de tripulaciones de aeronaves, la planificación en sí presenta muchas variaciones que dependerán del enfoque deseado. El que se estudia para este propósito consiste, a grandes rasgos, en el reto de cubrir una serie de vuelos previamente definida en forma de horario, con sus respectivos datos de partida, llegada, duraciones y destinos, siendo el objetivo principal minimizar los costos de operación, los cuales están concentrados específicamente en los salarios de las tripulaciones, mismos que varían en función de las actividades que realicen a lo largo de su jornada de trabajo, ya sea si se encuentran en el aire atendiendo algún vuelo, o si se encuentran en un aeropuerto esperando su siguiente vuelo. Por supuesto, entre más realismo se le añade al problema, más cantidad de restricciones contendrá, y esto incrementará su complejidad de manera directa. El presente capítulo relata una investigación a detalle acerca de la definición del problema, el estado del arte, y su respectivo modelo matemático.

2.1 Definición del problema

Para las aerolíneas, los costos de sus vuelos están definidos por distintos factores, algunos de ellos son: los costos de operación, el combustible del avión, el mantenimiento de la nave, impuestos aeroportuarios, demanda, servicios a bordo, entre otros. Sin embargo, según estudios que se han realizado [38] [39], los dos factores principales del costo total de un vuelo lo representan el combustible de la aeronave y los salarios de la tripulación que se encarga de atenderlo; es por ello que el problema de «scheduling» ha sido desde hace más de 60 años [40] un objeto de estudio tanto en el ámbito académico como para las propias compañías. De forma común e independientemente de las variaciones que presenta, el «Airline Crew Scheduling Problem» (ACSP) se suele dividir en dos grandes ramas: «crew pairing» y «crew assignment», este último también conocido como «crew rostering» [6]. A continuación se explicará en qué consiste cada una de ellas:

El «crew pairing» consiste en organizar una serie de vuelos predefinidos para un número de tripulaciones no preestablecidas. Dicha organización tiene que estar estructurada de tal manera que su precio de operación total represente el menor costo posible para la aerolínea y, además, que cumpla con una serie de restricciones que estén impuestas por la misma empresa o por externos, más adelante se hablará acerca de dichas restricciones. El objetivo, claro está, es cubrir todos los vuelos que ya se encuentran previamente programados para un intervalo de tiempo establecido, con la menor cantidad de costos de tripulación posible. Es de suponerse que, por ende, se intenta aprovechar al máximo las jornadas de trabajo de las tripulaciones, y a su vez minimizarlas, ya que así los salarios que se tengan que pagar y el personal requerido serán menores. El «crew pairing» es el problema en específico de estudio en este trabajo.

Las restricciones que se pueden presentar dependerán de lo estricto que se quiera ser en la organización, ya que, si bien algunas restricciones son bastante indispensables y se entienden de manera obvia, como el hecho de que las tripulaciones tienen que descansar después de cierta cantidad de horas de vuelo; existen otras que no son tan generales y que, por ejemplo, pueden depender del lugar donde se encuentren laborando, como las regulaciones impuestas por dependencias del gobierno. Como dato extra, también existen factores llamados KPI (Key Performance Indicators) [41] los cuales, si bien sirven principalmente para medir de forma numérica el desempeño de las tripulaciones, también pueden actuar como restricciones para mantener cierto desempeño mínimo o máximo en ellas.

Por otro lado, el «crew assignment» o «crew rostering» no solo satisface las peticiones establecidas por la aerolínea, sino que toma en cuenta las particularidades de cada persona que formará parte de las tripulaciones en específico, ya sea considerando sus habilidades, sus preferencias de horarios, sus días preferidos de descanso, entre muchos otros factores que lo pueden volver completamente personalizable. No obstante, debido a que satisfacer por completo (o en su mayoría) las necesidades de cada persona no ofrece grandes aportes en el ahorro económico de la empresa, no se le da tanta prioridad y se le da mucha mayor importancia y énfasis a la parte de «crew pairing»; tal como ocurre en este trabajo. La figura 2.1.1 muestra un diagrama acerca de las algunos de los distintos factores que involucran ambas partes del ACSP:

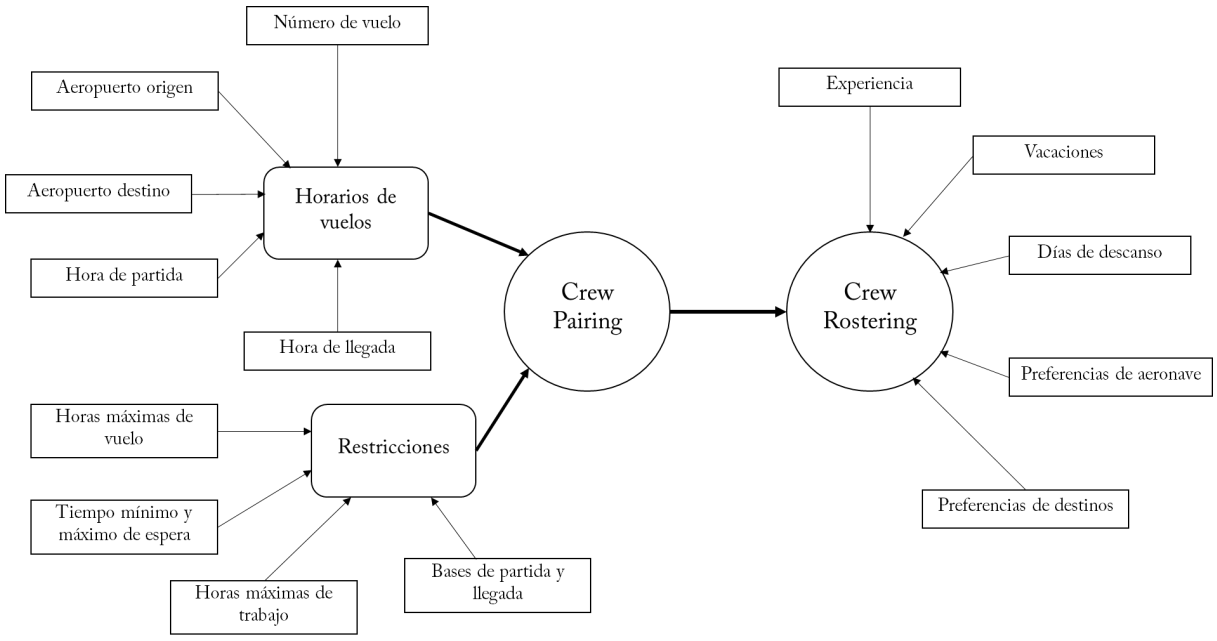


Figura 2.1.1. Procesos de «crew pairing» y «crew rostering». Fuente: «Algorithm creation and optimization for the Crew Pairing Problem» [39].

Se considera importante dejar en claro que el tema que se está tratando en el presente trabajo es una particularidad de la gran cantidad de variaciones y distintos enfoques que se le pueden dar al problema de «scheduling». Aun así, para no dejar sueltas algunas menciones de otros enfoques, a continuación se describirán brevemente un par de ellos:

Si, por ejemplo, se viera al problema de «scheduling» desde el enfoque de la aeronave, entonces surgirían otro tipo de restricciones, las cuales se centrarían alrededor del funcionamiento del avión. En ese caso, los vuelos ya no podrían estar estrictamente predefinidos, debido a que se tendrían que tomar en cuenta factores como las horas máximas de funcionamiento del avión, los mantenimientos que se le tienen que realizar cada cierto tiempo, los cuales, además, son de distinto tipo, y dejan a la nave inoperativa desde unas pocas horas hasta incluso un mes [42]. Dichos motivos darían un enfoque totalmente distinto, al que se le tendría que hacer un análisis completamente aparte del realizado en este trabajo.

Por otro lado, si se pensara en la organización de vuelos desde el punto de vista de la demanda, en ese caso se tendrían que considerar los diferentes modelos de aviones disponibles, para así tomar en consideración las capacidades de cada uno de ellos y, con base en la demanda según la

época del año, la cual fuese obtenida de estudios que hayan sido realizados en fechas anteriores para tener un estimado, asignar las aeronaves a vuelos que se adecúen según la cantidad de personas que lo soliciten.

2.2 Estado del arte

El problema de «scheduling», a pesar de que hoy en día posee enfoques en distintos sectores del transporte, tiene sus orígenes desde la década de los 50 en el sector de la aviación [40]. El origen se remonta a cuando las aerolíneas empezaron a notar el exponencial crecimiento en la demanda de vuelos, estas se vieron en la necesidad de buscar alternativas automatizables que se encargaran de organizar los horarios para las tripulaciones, pues, eventualmente, llegó un momento en que se volvió infactible el ser realizado por las propias personas; debido a ello es que se comenzó con la investigación acerca de métodos encargados de realizar dichas tareas de forma automática, con la prioridad, además, de siempre procurar minimizar los costos de operación para conservar la mayor ganancia posible [43].

Puesto que este problema ha sido de gran interés para las empresas que se dedican al transporte, a lo largo del tiempo se han desarrollado diferentes y variadas formas de solucionarlo, en otras palabras, se habla de distintos softwares que han sido vendidos a las aerolíneas, los cuales permiten cubrir y personalizar los horarios preestablecidos del conjunto de vuelos para un determinado tiempo. Este periodo regularmente suele ser de un mes, debido a que los lapsos de un día y una semana se consideran como insuficientes para poder tomar en cuenta algunas otras consideraciones, como los periodos de descanso semanales, las vacaciones, entre otros. Así, con periodos más largos, se pueden ofrecer alternativas de mayor variedad a las tripulaciones [6].

Uno de los enfoques que se han estudiado es verlo como un «Set Partitioning Problem» (SPP), en donde cada vuelo es una constante, puesto que ya tiene asignado un lugar de partida y un destino, así como un horario de llegada y uno de salida en una fecha también ya definida. Las variables del problema son las tripulaciones que serán asignadas a tales vuelos (estrictamente una por vuelo), mientras que las restricciones que se deben cumplir son la serie de reglas de la empresa y las establecidas por el gobierno, las cuales cambiarán para cada caso. La característica esencial del SPP es que los vuelos deben ser cubiertos exactamente por una tripulación.

Por otro lado, también existe la posibilidad de tratar el problema como un «Set Covering Problem» (SCP), el cual mantiene un gran parecido con el SPP, modificando únicamente la restricción que exige que cada vuelo tiene que ser cubierto exactamente por una tripulación; el SCP, por su lado, altera esta restricción de igualdad, y establece que los vuelos tienen que ser cubiertos, por lo menos, por una tripulación. Sin embargo, el problema de alterarla es que hace que se generen los vuelos llamados «dead-head» [44], los cuales se intentan minimizar, puesto que en esos vuelos una (o incluso más) tripulación tiene que viajar como pasajera para llegar a otro destino y continuar con su labor, lo cual, para la empresa, es tiempo y dinero perdido.

Siendo lo más realista posible, resulta relevante considerar que, aunque en la mayoría de casos de estudio, la información de cada vuelo es tratada como invariable, en la práctica existen cientos de factores que pueden alterar sus valores y es algo muy común, pues se ven afectados por distintos retrasos que modifican a las «constantes», y estas dejan de serlo.

En lo que respecta a las técnicas que han sido desarrolladas para resolver este problema, existen muchas referencias que pueden ser encontradas en la web, donde cada autor explica las ventajas, desventajas, y el porqué del método de resolución empleado. Utilizan estructuras que, de hecho, son bastante similares a lo que se documenta en este trabajo. A continuación se mencionan algunos de los enfoques que han sido desarrollados por distintas personas en el ámbito de la investigación:

Harry Kornilakis y Panagiotis Stamatopoulos dan solución al problema mediante la técnica incompleta de algoritmos genéticos [45]. En su trabajo, el problema es visto como un SCP, es decir, que existe la probabilidad de que se produzcan vuelos «dead-head». Su algoritmo funciona de la siguiente manera: inicialmente, se generan las listas de tripulaciones que cubren los vuelos, y posteriormente es cuando se optimizan mediante la evolución de los genes de los individuos. Cuando recién se crean, estas tienen muchos vuelos en los que una tripulación, o incluso más, tienen que viajar como pasajeros, es decir, los costos totales son bastante altos debido a estos sucesos. Sin embargo, puede observarse que, conforme el algoritmo evoluciona, las tripulaciones van mejorando, llegando a un punto en el que el número de vuelos «dead-head» tiende a ser 0. Los resultados que obtienen al final de su desarrollo son bastante buenos, los realizan con un mismo conjunto de datos de una referencia para poder compararlos, e incluso descubren que los

resultados obtenidos por su algoritmo son mejores que los del otro proyecto, concluyendo finalmente que su solución es viable.

Matias Sevel Rasmussen et. al., en su proyecto, utilizan la técnica de generación de subsecuencias [46]. Esta técnica es bastante similar a la generación de columnas, de hecho, incluye a la generación de columnas dentro de sí, excepto que dichas columnas son modificadas de forma dinámica conforme se avanza en el proceso. De hecho, su investigación se centra en comparar resultados de la generación de subsecuencias, contra la generación de columnas. Los resultados obtenidos son menos fraccionarios, es decir, que la cantidad de vuelos cubierta y sin «dead-head» es mayor, sin embargo, el uso de esta técnica tiene un impacto en la calidad de las soluciones, es decir, los costos totales de las tripulaciones no son tan bajos en comparación de la técnica de generación de columnas.

Dejando a un lado el ámbito de investigación y entrando al ámbito comercial, se han desarrollado distintos algoritmos que han sido vendidos exitosamente a las aerolíneas, la mayoría se aprovechan de las heurísticas para dar soluciones acertadas. Por ejemplo, uno de los programas software más populares que se han puesto a la venta para el ACSP lleva por nombre Carmen, pertenece a la empresa Jeppesen, y este presume de ser la mejor alternativa, además de que es mencionado de manera frecuente en los distintos textos académicos. Según la limitada información proporcionada, este software puede trabajar hasta con 25,000 vuelos programados [47].

Existen muchas otras opciones, pero claro está que por temas de derechos de autor se desconoce el tipo de algoritmos que utilizan para entregar las soluciones. Sin lugar a dudas se podría decir que prácticamente ningún software que se encuentre a la venta para resolver los problemas de «scheduling» revela cómo funciona; toda la información que se puede encontrar acerca de los algoritmos que se han implementado provienen de textos meramente académicos, pues, al ser este un problema tan conocido, es típico de estudiarse en las distintas universidades del mundo, como el par de casos mencionados anteriormente.

2.3 Modelo matemático

A continuación, se define la función objetivo, misma que es utilizada por Roy E. Martsen [48] para representar el problema del ACSP como un «Set Partitioning Problem», donde la función a minimizar es (1):

$$mín \sum_{j=1}^{n} c_j x_j \tag{1}$$

$x_j = \{0, 1\}$ para cada j = 1, ..., n; n simboliza el número de jornadas o tripulaciones; c_j simboliza los costos de operación de cada jornada j.

La ecuación (1) indica que la función que se intenta minimizar es la suma de los costos de cada jornada de trabajo. Lo cual está sujeto a la restricción obligatoria (2):

$$\sum_{j=1}^{n} a_{ij} x_j = 1 \tag{2}$$

donde a_{ij} es 1 si el vuelo i se encuentra cubierto por la jornada j, y es 0 en caso contrario. Una jornada j consta de una tripulación j que parte de una base, cubre cierta cantidad de vuelos, y regresar a la misma base de donde partió.

La ecuación (2) establece la condición fundamental del SPP, donde cada vuelo debe estar cubierto exactamente por una tripulación, y no más.

Por otro lado, si se desea considerar el problema como un «Set Covering Problem», entonces la ecuación (2) queda reescrita de la siguiente manera (3), y lo demás se mantiene exactamente igual:

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1 \tag{3}$$

La ecuación (3), característica principal del SPP, indica que cada vuelo puede estar cubierto por una o más tripulaciones, permitiendo la posibilidad de «deadheading».

El modelo, entonces, se resume en encontrar la *n* cantidad de tripulaciones necesaria, que minimice los costos de operación, pero que a la vez satisfaga la condición de que todos los vuelos se encuentren cubiertos por alguna de ellas. Sin embargo, es de notarse que esta representación matemática aún no posee el suficiente realismo, al no tomar en cuenta otras tantas consideraciones. Las restricciones que serán consideradas para este trabajo serán las siguientes:

Restricciones espaciales

- La tripulación siempre debe partir de alguna de las bases con las que la aerolínea cuente, es decir, aeropuertos donde tengan personal listo para partir:

$$Orig_{1j} = B1 \,||\, B2 \,||\ldots||\, Bn \tag{4}$$

  Para la ecuación (4), $Orig_{1j}$ es el aeropuerto origen del vuelo 1 que cubra la tripulación *j*; y *B1, B2, …, Bn*, son cualquiera de las bases con las que cuente la aerolínea. || simboliza la función lógica *or*.

- El destino del vuelo anterior debe ser igual al origen del vuelo siguiente a cubrir:

$$Dest_{(i-1)j} = Orig_{ij} \tag{5}$$

  Para la ecuación (5), $Dest_{(i-1)j}$ es el destino del vuelo $i - 1$ (anterior) cubierto por la tripulación *j*, mientras que $Orig_{ij}$ es el origen del vuelo *i* (siguiente) cubierto por la misma tripulación.

- El destino del último vuelo cubierto por una tripulación *j*, debe ser igual al origen de su primer vuelo, es decir, a la base de donde partió:

$$Dest_{finalj} = Orig_{1j} \tag{6}$$

Para la ecuación (6), $Dest_{finalj}$ es la ciudad destino del último vuelo (*final*) cubierto por la tripulación *j*, y $Orig_{1j}$ es la ciudad origen de su primer vuelo, es decir, la base de donde partió.

Restricciones temporales

- La tripulación no debe pasar más de cierta cantidad de horas de vuelo en su jornada de trabajo:

$$T_{Tvuelo} = \sum_{j=1}^{n} a_{ij} d_i \leq T_{vmáx} \quad (7)$$

Para la ecuación (7), d_i es la duración del vuelo *i* cubierto por la tripulación *j*, y $T_{vmáx}$ es el tiempo máximo establecido que pueden pasar en el aire.

- La tripulación no debe exceder más de cierta cantidad de horas totales de trabajo:

$$T_{Tvuelo} + T_{Tespera} \leq T_{máxtrabajo} \quad (8)$$

Para la ecuación (8), T_{Tvuelo} representa el tiempo total de vuelo que una tripulación pasó en su jornada, mientras que $T_{Tespera}$ representa el tiempo total de espera, y $T_{máxtrabajo}$ es el tiempo máximo que pueden pasar trabajando de manera continua.

- La tripulación debe tener un tiempo mínimo entre la llegada de un vuelo y la salida del siguiente, para poder hacer el cambio de vuelo de manera exitosa:

$$Dep_{ij} - Arr_{(i-1)j} \geq T_{emín} \quad (9)$$

Para la ecuación (9), Dep_{ij} es la hora de salida del vuelo i que es cubierto por la tripulación *j*, $Arr_{(i-1)j}$ es la hora de llegada del vuelo $i - 1$ (anterior) cubierto por la misma tripulación, y $T_{emín}$ es el tiempo de espera mínimo establecido.

- Debe haber un tiempo máximo de espera entre la llegada de un vuelo y la salida del siguiente, para que la tripulación no pase demasiadas horas en un aeropuerto esperando cubrir el próximo:

$$Dep_{ij} - Arr_{(i-1)j} \leq T_{emáx} \tag{10}$$

Para la ecuación (10), Dep_{ij} es la hora de salida del vuelo i que es cubierto por la tripulación j, $Arr_{(i-1)j}$ es la hora de llegada del vuelo $i - 1$ (anterior) cubierto por la misma, y $T_{emáx}$ es el tiempo de espera máximo establecido.

Las desigualdades (9) y (10) pueden agruparse y formar una restricción doble, la cual da lugar a la (11):

$$T_{emín} \leq Dep_{ij} - Arr_{(i-1)j} \leq T_{emáx} \tag{11}$$

Por lo tanto, la ecuación (11) indica que la tripulación debe tener un tiempo mínimo y máximo de espera entre la llegada de su vuelo anterior y la salida del siguiente.

Como puede observarse, la función objetivo se mantiene mayormente constante; sin embargo, las restricciones de este son las que pueden cambiar directamente con las consideraciones que se quieran tomar en cuenta. Claro está que no todas las restricciones pueden tratarse como estrictas, ya que, si este fuera el caso, podría haber ocasiones en las que no se tuviera solución al problema. El espacio de búsqueda puede ser tan grande que realmente nunca se tendrá completa certeza de si existe una solución que satisfaga al 100 % todas las restricciones, por ello es que se opta por flexibilizar algunas de ellas.

Las decisiones acerca de qué restricciones son tomadas como estrictas y cuáles no, dependen directamente de las prioridades de la aerolínea, según sean sus conocimientos acerca de lo que les salga menos costoso si hay que darlo a cambio. Para este trabajo, las restricciones que serán consideradas como «blandas» son las siguientes:

- Se permitirán soluciones que no cubran todos los vuelos, añadiendo una penalización al costo total de esta por cada vuelo que quede sin ser atendido.
- Se permitirá que los «crews» no retornen a sus bases, a cambio de un pago extra como compensación para viáticos por pasar el resto del día en una ciudad distinta a su origen.
- Se permitirá la presencia de «deadheads», también con un costo adicional por cada tripulación que viaje como pasajera en un vuelo.

3. Diseño del primer sistema inteligente con técnicas completas

En el presente capítulo se describe detalladamente el diseño del primer algoritmo, el cual utilizará técnicas de tipo completa en la búsqueda de soluciones para el problema de ACSP. Las técnicas elegidas fueron una de tipo «look-ahead»: Forward Checking; y otra de tipo «look-back»: Graph Back Jumping. Se eligieron ambas técnicas con el objetivo de crear un algoritmo más eficiente, en el sentido de que sea capaz de «mirar al futuro» mediante FC, y a su vez, poder «regresar al pasado» cuando se vea «atascado» en alguna situación, mediante GBJ.

3.1 Planteamiento del algoritmo

Si bien todas las técnicas son algoritmos que poseen sus respectivos procedimientos ya establecidos, se debe encontrar la manera de implementar tales procedimientos al problema en cuestión, por ello, a continuación se describe la manera en que fueron implementadas las técnicas de FC y GBJ para el problema de ACSP en específico:

Implementación de la técnica Forward Checking

FC, como se menciona previamente, es una técnica que funciona como un filtro encargado de eliminar aquellos valores del dominio de las variables que son infactibles para continuar la construcción de la solución, por lo tanto, será aplicado para la etapa de elección del vuelo siguiente que cubrirá la tripulación sobre la que se esté trabajando.

En este caso, desde la perspectiva del algoritmo existe una sola variable: el vuelo siguiente. El dominio de dicha variable es todos los vuelos que hasta el momento no han sido cubiertos por ninguna tripulación, por lo que podría elegirse cualquiera de ellos y comprobar una vez elegido si este cumple con las restricciones o no, pero hacerlo de esta manera sería muy poco eficiente porque tendría que probar con el dominio completo hasta encontrar un vuelo que las satisfaga, por ello es que el FC será de gran utilidad.

Con la información del último vuelo cubierto y la lista de vuelos sin cubrir, el Forward Checking se encargará de generar una lista de potenciales vuelos siguientes factibles, los cuales satisfarán las restricciones (5-10), de tal manera que el algoritmo escoja alguno de dicha lista con la

completa certeza de que, al hacerlo, respetará todas esas condiciones; el vuelo futuro podrá ser elegido al azar, o mediante heurísticas, según sea la condición que se presente.

La figura 3.1.1 es una ilustración del funcionamiento, donde el algoritmo está construyendo la séptima tripulación, la cual lleva cubiertos los vuelos 6 y 26 hasta el momento; mediante FC se filtra la enorme lista de vuelos disponibles por cubrir, dejando así sólo los factibles y escogiendo uno de ellos, para después continuar:

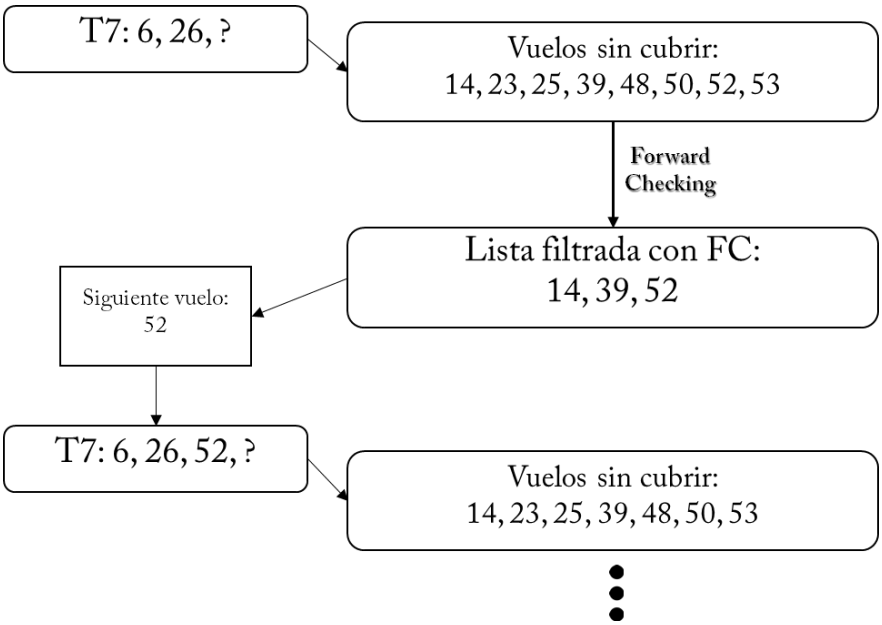


Figura 3.1.1. Implementación del Forward Checking. Fuente: elaboración propia.

Implementación de la técnica Graph Back Jumping

La técnica de GBJ vuelve al pasado apoyándose en las relaciones entre variables. Si el algoritmo encuentra un «atasco» mientras trabajaba sobre una variable *x*, este volverá al momento en que se encontraba trabajando con la variable más cercana relacionada con *x*. Sin embargo, como se comenta previamente, para el algoritmo sólo existe una variable: el vuelo siguiente; el cual siempre se encontrará relacionado (restringido) con el vuelo anterior, pues las restricciones se centran en parte en vuelos consecutivos. Por lo tanto, implementar el GBJ de esta manera haría que se convirtiera en un simple «backtracking», ya que volvería siempre al vuelo inmediato anterior, por ello es que se plantea una forma alternativa de implementarlo.

La alternativa planteada parte del motivo fundamental sobre el porqué las tripulaciones se «atascan». Resulta que, una tripulación puede «atascarse» cuando no tiene ningún vuelo factible por cubrir y esta tampoco se encuentra en su base, es decir, el último vuelo que cubrió no la regresó a su base, ni existe alguno que satisfaga las restricciones y a su vez la devuelva al aeropuerto de donde partió. Así es que surge la idea de crear un GBJ que tome en cuenta esta situación y que, en vez de volver al vuelo anterior, regrese al momento más cercano en que la tripulación tenía manera de regresar a su base, en otras palabras, al momento en que, de la lista filtrada de vuelos por FC, exista por lo menos uno que la haga volver a su base. Haciendo esto, puede permitirse al algoritmo explorar cuantos caminos sean posibles, pero siempre teniendo manera de escaparse en caso de algún tipo de «atasco» y poder regresar a su aeropuerto de origen.

La figura 3.1.2 ilustra el funcionamiento del GBJ. Supóngase que el algoritmo se encuentra construyendo cualquier tripulación, y cuando se encuentra sobre el vuelo A, observa que de la lista filtrada de vuelos disponibles, existe alguno que lo lleva de vuelta a la base de partida, entonces el sistema «recordará» ese estado en su memoria. Posteriormente elige los vuelos B, C, D, y en ninguno de ellos vuelve a haber un vuelo que lo regrese a base; además, «se da cuenta» de que cuando está en el vuelo D, ya no le queda tiempo para cubrir ningún vuelo que no sea alguno que lo lleve a su base, como no hay ningún vuelo con ese destino y la tripulación tampoco se encuentra en su base, entonces se «atasca». Por ende, se utiliza el GBJ para volver al momento en que había elegido el vuelo A, y posteriormente elegir el vuelo E en vez del B, para así irse a explorar por otro camino y evitar el «atasco» previamente encontrado.

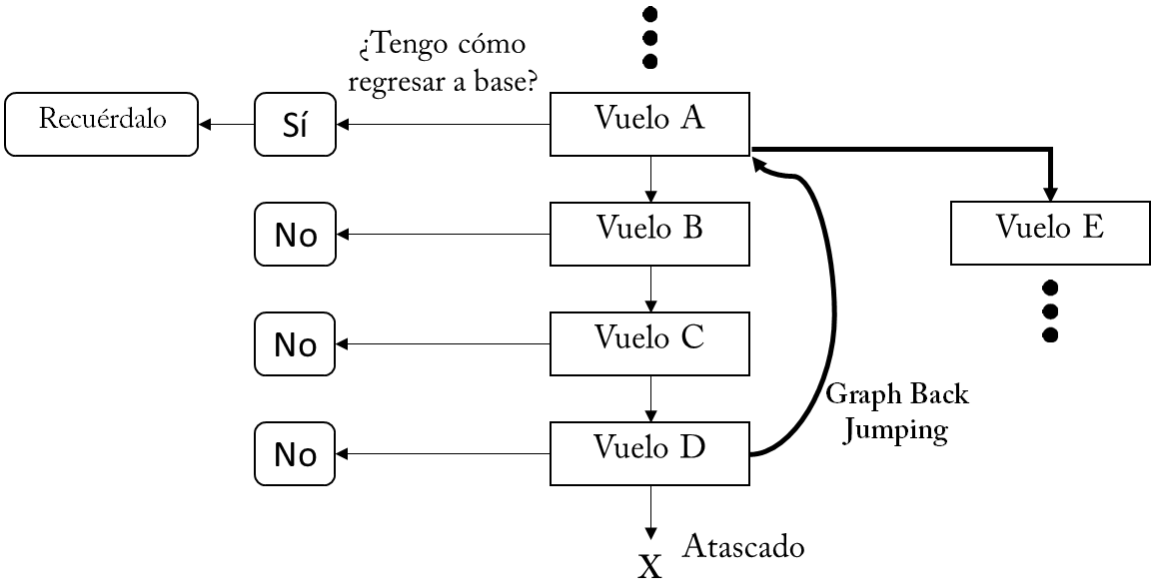


Figura 3.1.2. Implementación del Graph Back Jumping. Fuente: elaboración propia.

Una vez teniendo claramente definidas las implementaciones que se harán para cada técnica, puede comenzar a plantearse el algoritmo completo. Lo primero que se tiene que hacer, claro está, es obtener toda la información del horario en cuestión, de este se tienen que recuperar:

- Número de vuelos del horario: para saber qué cantidad de vuelos se manejará.
- ID de los vuelos: para tener algún número identificador de cada vuelo.
- Lugares de partida: uno asociado a cada ID de vuelo.
- Lugares de llegada: uno asociado a cada ID de vuelo.
- Horas de salida: una hora de salida asociada a cada ID.
- Horas de llegada: una hora de llegada asociada a cada ID.

Una vez recuperada la información de algún archivo de texto previamente existente, el algoritmo puede comenzar a trabajar con los datos obtenidos.

Paso 1: que el algoritmo genere una lista que contenga las ID de los vuelos que partan de alguna de las bases de la aerolínea, supóngase el nombre: *vuelos_from_bases*. Como vuelo inicial para toda tripulación que se genere, siempre se escogerá algún vuelo de esta lista, pues la restricción de la ecuación (4) indica que no puede ser de otra manera.

Paso 2: la lista *vuelos_from_bases* debe reacomodarse en forma temporal por hora de salida, es decir, del vuelo que sale más temprano al vuelo que sale más tarde, esto con el objetivo de que el algoritmo priorice los vuelos que más temprano salen, para que estos no se queden sin ser cubiertos si no se eligen con anticipación. Por ende, el primer vuelo escogido por el algoritmo siempre será el que más temprano salga de todos.

Paso 3: se elegirá el siguiente vuelo mediante la técnica de FC. A partir del vuelo anterior, el FC filtrará todos los vuelos que no satisfagan las restricciones antes mencionadas, dejando una única lista de vuelos factibles que podrán elegirse como el siguiente. Entonces, se escogerá un vuelo de la lista filtrada, y este será el vuelo siguiente de la tripulación en cuestión.

Para el paso 3 deben tenerse varias consideraciones extra. Se propone que el paso 3 esté alojado en una función llamada *siguiente_vuelo*, esta función debe ser llamada desde el programa principal, ingresándole el vuelo anterior y los demás datos necesarios para que esta realice su trabajo, pero *siguiente_vuelo* no sólo devolverá la ID de vuelo siguiente, también tiene que entregar varios valores extra que le servirán al programa *main* para saber cómo continuar, estos son:

- Una variable o bandera que le avise al *main* si la función ha detectado que el siguiente vuelo elegido, o el vuelo presente, será el último que cubra la tripulación, ya sea porque se han quedado sin tiempo de vuelo o de jornada, porque ya no les quedará tiempo suficiente para cubrir otro después, o porque ya no hay ningún vuelo disponible por cubrir y la tripulación ya se encuentra en su base. Supóngase el nombre *fin_tripulación*, variable de tipo booleana al ser sólo verdadera o falsa.
- Una variable o bandera que le haga saber al *main* si la función ha detectado que la tripulación se encuentra atascada, es decir, que la función no haya podido elegir un vuelo siguiente que satisfaga las restricciones, y que la tripulación tampoco se encuentre en su base como para poner la variable *fin_tripulacion* en verdadera. Esta variable cobrará relevancia en la etapa de GBJ, pues es la que le indicará al programa principal si necesita regresar en el tiempo. Se asignará el nombre de *atascado*, variable también de tipo booleana.

- Finalmente, una variable que le indique al programa *main* si, de la lista de vuelos disponibles a elegir, se encontraba por lo menos uno que lo llevara de vuelta a su base, esto también tiene relevancia para la etapa de GBJ. Se propone el nombre de *hay_retorno* para esta variable que, al igual que las anteriores, será de tipo booleana.

Una vez obtenida la información devuelta por la función *siguiente_vuelo*, el algoritmo tomará distintas decisiones dependiendo de la situación que se presente, las cuales se enlistan a continuación:

- Si la variable *hay_retorno* es verdadera, se realizará un «backup» del estado actual de la solución, con el fin de volver a esta en cuanto sea necesario realizar un GBJ.
- Si la variable *atascado* es verdadera, se importará el último «backup» realizado, es decir, se realizará el GBJ, y el siguiente vuelo elegido será uno distinto al elegido previamente, para que no haya la posibilidad de caer en el mismo atasco.
- Si la variable *atascado* es falsa, podrán presentarse dos situaciones más:
- Cuando *fin_tripulación* sea verdadera, entonces la tripulación terminará su jornada, ya sea añadiendo la ID del vuelo siguiente a su lista de vuelos cubiertos, si lo hay, o simplemente quedándose en donde estaba, si es que esta ya se encontraba previamente en su base y *siguiente_vuelo* decidió que lo mejor es quedarse allí.
- Cuando *fin_tripulación* sea falsa, entonces el algoritmo añadirá la ID del vuelo elegido a la lista de vuelos cubiertos, y continuará de nuevo con el paso 3 hasta que llegue el fin de la tripulación.

Paso 4: una vez que se termine con la construcción de una tripulación completa, el algoritmo escogerá el vuelo inicial para la siguiente, este será aquel que salga más temprano de la lista de *vuelos_from_bases,* por supuesto, que tampoco haya sido cubierto, ya que existe la posibilidad de que una tripulación cubra como vuelo intermedio alguno de los que salen de las bases, así que la lista *vuelos_from_bases* deberá actualizarse y tener únicamente aquellos que no hayan sido cubiertos, siempre en orden cronológico. El algoritmo se mantendrá en los pasos 3-4 mientras la lista *vuelos_from_bases* siga conteniendo vuelos.

Paso 5: una vez que *vuelos_from_bases* ya no contenga más vuelos, se hará un recuento de los vuelos que no se pudieron cubrir, y se calculará el costo total de la solución encontrada, sumando los costos de cada jornada de trabajo y agregando las penalizaciones correspondientes.

3.2 Diagrama de flujo

A continuación, en la figura 3.2 se muestra el diagrama de flujo del funcionamiento del algoritmo. Este se encuentra simplificado, pues no muestra mas que los procedimientos principales que realiza el sistema, sin énfasis en los subprocesos y funciones complementarias que todas las etapas conllevan, ya que el diagrama se tornaría muy enredado por la cantidad de conexiones. La figura 3.2 se encuentra dividida en 2 partes (a y b) debido a su tamaño:

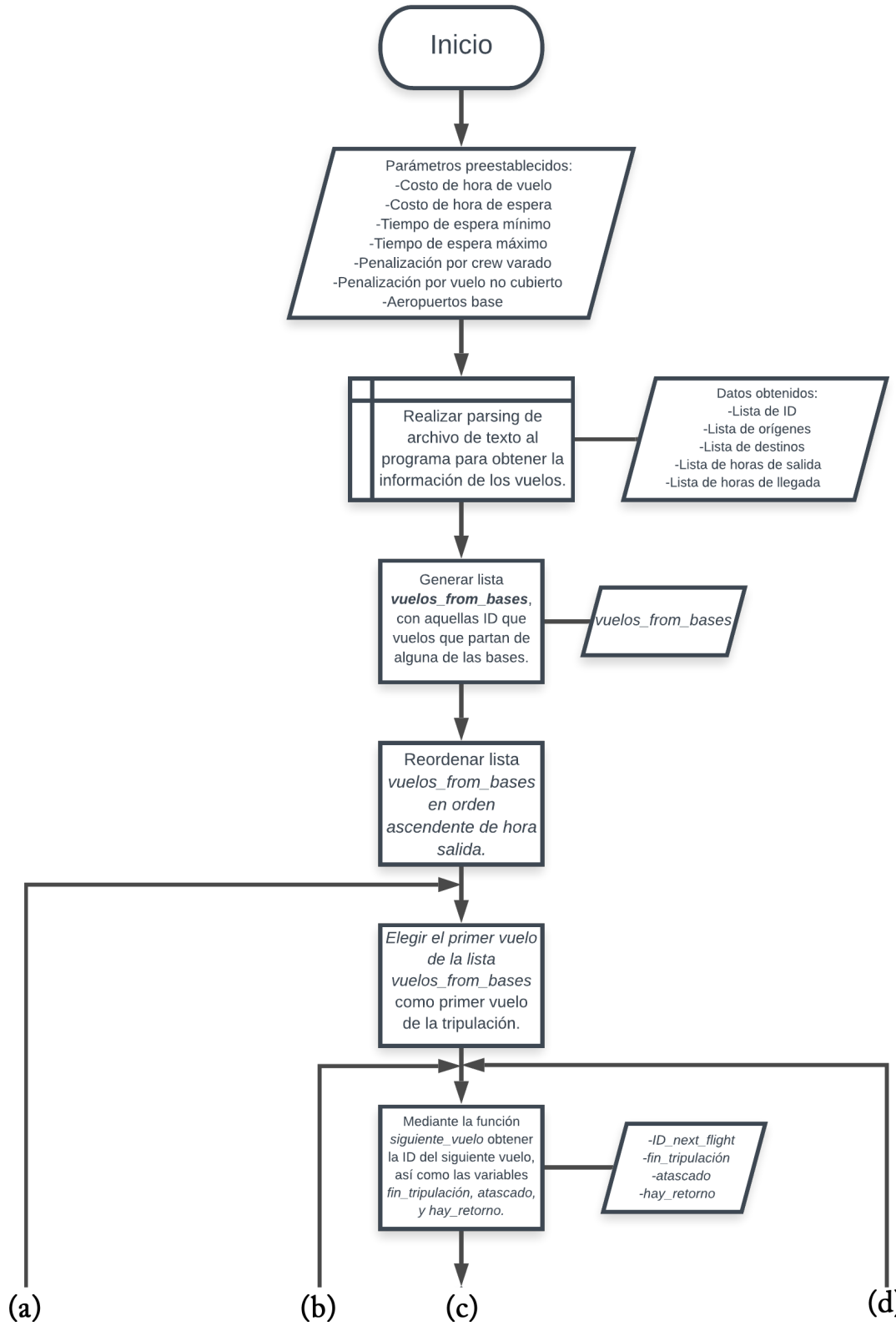


Figura 3.2a. Diagrama de flujo del primer sistema inteligente, parte 1. Fuente: elaboración propia.

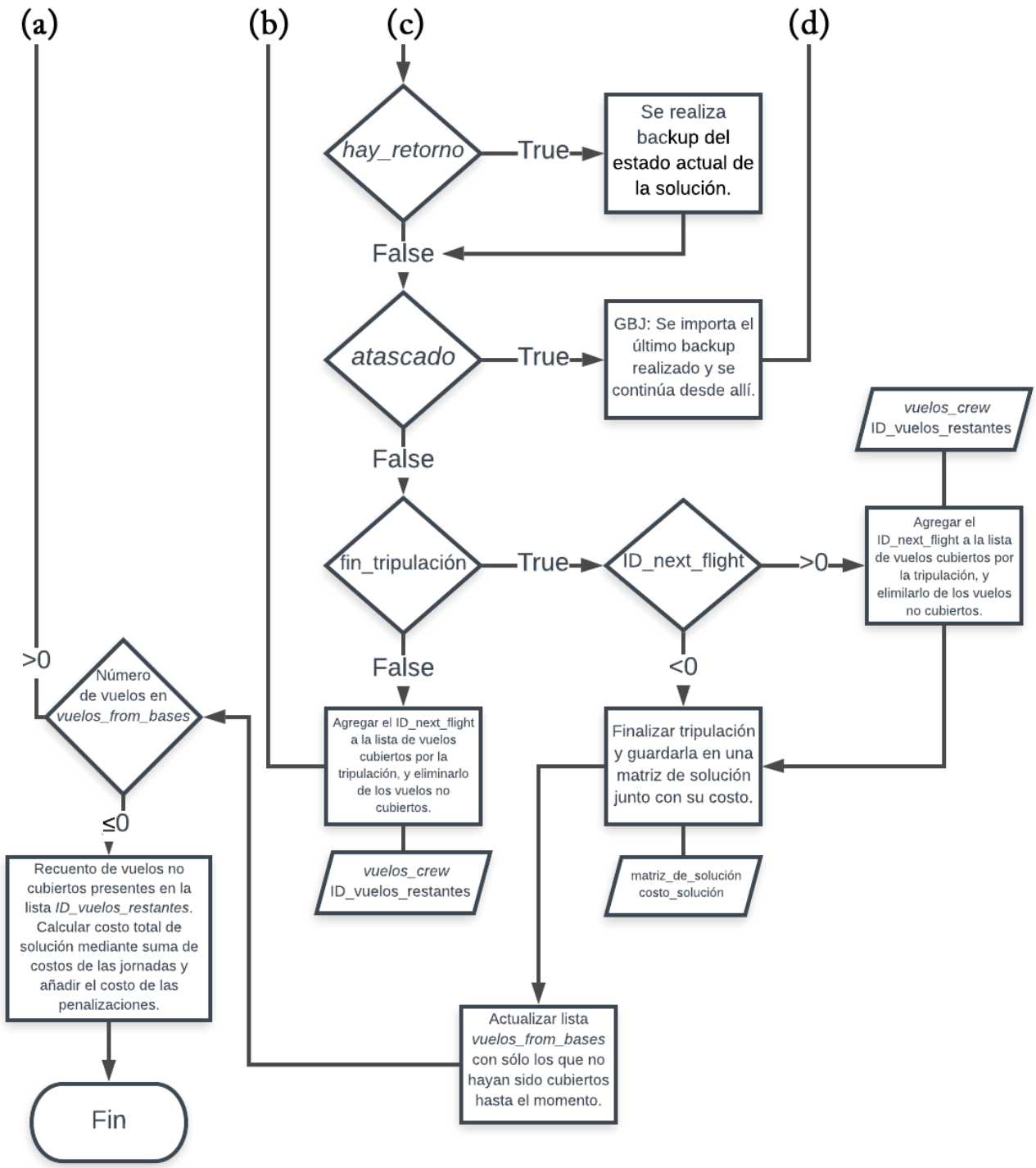


Figura 3.2b. Diagrama de flujo del primer sistema inteligente, parte 2. Fuente: elaboración propia.

3.3 Representación en pseudocódigo

Antes de comenzar la parte de programación, se debe tener una idea aún más detallada acerca de los procesos y funciones que se ejecutarán durante todo el programa, por ello es que es necesaria la representación en pseudocódigo, servirá además como guía fundamental para cualquier persona que desee replicar el algoritmo en algún otro lenguaje de programación. La representación en pseudocódigo del algoritmo queda de la siguiente manera:

```
función primer_vuelo () { //Esta función elegirá el primer vuelo de un crew.
       actualiza la lista vuelos_from_bases con sólo vuelos que no estén cubiertos
       hasta el momento;
       reordena vuelos_from_bases por orden de salida ascendiente;
       vuelo_inicial = primer elemento de vuelos_from_bases;
       reinicia los valores de vuelos_crew, horas_de_trabajo, horas_de_vuelo,
       horas_de_espera;
       horas_de_vuelo = duración de vuelo_inicial;
       horas_de_trabajo = horas_de_vuelo + horas_de_espera;
       agrega a vuelos_crew el vuelo_inicial;
       quita el vuelo_inicial de ID_vuelos_restantes;
       pos_solución = 0; //Número de vuelos en el vector vuelos_crew
}

función reset_completo () { //Esta función reiniciará todo el algoritmo.
       reinicia la lista de ID_vuelos_restantes;
       reinicia la matriz_de_solución;
       reinicia el costo de la solución;
       reinicia el contador de crews;
}

función siguiente_vuelo(vuelo_anterior){ //Esta función elige el siguiente vuelo.
//FC p1: Destino de vuelo_anterior = origen del vuelo_siguiente
       lista vuelos_FC_1 = obtén de ID_vuelos_restantes los que salgan de la ciudad
       destino(vuelo_anterior);
//FC p2: Restricciones de tiempo de espera mínimo y máximo
       lista vuelos_FC_2 = obtén de vuelos_FC_1 aquellos que: (hora_de_salida >=
       hora_de_llegada(vuelo_anterior) + tiempo_mínimo_de_espera) && (hora_de_salida <=
       hora_de_llegada(vuelo_anterior) + tiempo_máximo_de_espera);
//FC p3: Aquellos vuelos que, si se eligen, el crew no exceda sus horas //máximas de
trabajo, ni sus horas máximas de vuelo.
```

```
	lista vuelos_FC_3 = obtén de vuelos_FC_2 aquellos que, si se eligen, no excedan
	las horas máximas de trabajo ni de vuelo del crew, considerando su duración y el
	tiempo de espera;
//si no hay vuelos disponibles, que revise si está en base
	si vuelo_FC_3 no tiene elementos {
		si el destino vuelo_anterior los dejó en su base {
			fin_tripulación = true;
			sig_vuelo = 0; //null
		} si no es así {
			atascado = true;
		}
	} si hay por lo menos un elemento en vuelos_FC_3 {
		//Elección del vuelo siguiente mediante heurísticas
		si (horas_de_trabajo_restantes >= 3) && (horas_de_vuelo_restantes >= 2) {
			fin_tripulación = false;
			si número de elementos en vuelos_FC_3 > 1 {
				sig_vuelo = cualquiera de lista vuelos_FC_3;
			} si sólo es 1 elemento {
				sig_vuelo = primer elemento de vuelos_FC_3;
			}
		//Si no le quedan más de 3 y 2 horas restantes:
		} en otro caso {
			si hay algún vuelo de vuelta a base en vuelos_FC_3 {
				sig_vuelo = alguno que los regrese a base;
				fin_tripulación = true;
			} si no hay ningún vuelo de vuela {
				si ya se encuentran en su base {
					//Que allí se queden.
					fin_tripulación = true;
					sig_vuelo = 0;
				} si no están en su base {
					atascado = true;
				}
			}
		}
	}
	//Posteriormente se revisará si se puede hacer backup:
	si atascado es falso && sig_vuelo>0 && fin_tripulación es falso && hay más de un
	vuelo en vuelos_FC_3 {
		si en vuelos_FC_3 hay alguno que lo lleve a base {
			hay_retorno = true;
```

```
            } si no {
                  hay_retorno = false;
            }
            si hay_retorno es true {
                  reacomoda aleatoriamente vuelos_FC_3;
                  sig_vuelo = primer elemento de vuelos_FC_3;
                  quita el sig_vuelo de vuelos_FC_3;
                  llena el vector backup_vuelos_FC_3 con vuelos_FC_3;
            }
      }
      devuelve sig_vuelo, atascado, fin_tripulación, hay_retorno;
}
int main () { //Programa principal del algoritmo
      haz el parsing del archivo de texto y obtén la información de los vuelos,
      llenando los vectores: ID_vuelo, origin, destination, departure, arrival;
      convierte las horas de salida y llegada, originalmente de tipo char, a valores
      de tipo flotante, para que sean valores numéricos y no cadenas de caracteres;
      //Ya que originalmente ningún vuelo está cubierto:
      vector ID_vuelos_restantes = vector ID_vuelo;
      llena la lista vuelos_from_bases, filtrando los vuelos que parten de alguna de
      las bases;
      ordena de forma ascendiente la lista vuelos_from_bases, por hora de salida;
      llena el vector de duraciones mediante la resta de la hora de llegada y salida
      de cada vuelo;
      haz {
            primer_vuelo;
            mientras fin_tripulación sea falso {
                  si número de atascos > número máximo de atascos {
                        reinicia la construcción del crew;
                        número de resets de crew++;
                        número de atascos = 0;
                  }
                  si número de resets de crew > núm.máx. de crew resets {
                        reset_completo;
                        primer_vuelo;
                  }
                  vuelo_anterior = vuelos_crew[pos_solucion];
                  siguiente_vuelo(vuelo_anterior);
                  //Si hay retorno, que realice el backup
                  si hay_retorno es true {
                        haz backup completo del estado actual de la solución;
```

```
                }
                //Si está atascado, que realice el Graph Back Jumping
                si atascado es true {
                        num_atascos ++;
                        importa último backup realizado;
                        si backup_vuelos_FC_3 tiene por lo menos un elemento {
                                sig_vuelo = primer elemento de backup_vuelos_FC_3;

                                quita sig_vuelo de backup_vuelos_FC_3;
                                atascado = false;
                        } si backup_vuelos_FC_3 ya no tiene elementos{
                                realizar reset de tripulación;
                                num_crew_resets ++;
                        }
                }
                //Si ha elegido un vuelo válido…
                si atascado es falso && sig_vuelo > 0 {
                        vuelo_anterior = vuelos_crew[pos_solución];
                        pos_solución ++;
                        agrega a vuelos_crew el sig_vuelo;
                        incrementa horas_de_vuelo con la duración del vuelo;
                        incrementa horas_de_espera con el tiempo de espera;
                        horas_de_trabajo = horas_de_vuelo + horas_de_espera;
                }
        }
        //Fin tripulación ya es true:
        añade vuelos_crew a la matriz_solución;
        calcula costo_crew con las horas_de_trabajo y horas_de_espera;
        añade costo_crew a vector_de_costos;
} mientras vuelos_from_bases aún tenga elementos;
revisa cuántos vuelos quedaron sin cubrir;
calcula penalización por vuelos no cubiertos;
calcula penalización por crews que no volvieron a su base;
calcula la suma de costos de todos los crews;
calcula el costo total de la solución encontrada sumando costos y penalzs.;
exporta la solución en un archivo de texto, incluye los vuelos que cubre cada
tripulación, sus costos, las ID de los vuelos sin cubrir, las tripulaciones que
quedaron varadas, y el costo final de la solución;
}
```

3.4 Lenguaje de programación y ambiente de desarrollo

En esta sección se hablará sobre el lenguaje de programación elegido, así como el ambiente de desarrollo a utilizar para la parte de programación de ambos sistemas inteligentes. Se comienza por el lenguaje de programación, ya que la disponibilidad de ambientes de desarrollo depende del lenguaje a utilizar.

Lenguaje de programación C

Se ha elegido C para la realización de ambos proyectos. El lenguaje fue creado por Dennis Ritchie en 1972, quien se basó en los lenguajes previamente existentes B y BCPL para crear uno que incluyera características de ambos, con la meta principal de que este fuera capaz de funcionar de forma exitosa en distintas plataformas, ya que los lenguajes B y CPL habían sido intentos fallidos de tal meta [49]. C es un lenguaje considerado de nivel medio, es decir, cuenta con ciertas características de lenguajes de alto nivel, pero a su vez permite la manipulación directa de bits, utilización de punteros, y acceso directo a registros de memoria; características que son propias de los lenguajes de bajo nivel [50].

C, además, es un lenguaje con significativa importancia en la historia de la programación, ya que gran variedad de lenguajes modernos han sido basados en él, pues recogen alguna o varias características de este lenguaje, algunos ejemplos son: Java, C#, C++, Objective-C, PHP, Python, entre muchos otros [51].

La facilidad y portabilidad de este lenguaje se consideran como factores decisivos para su elección, ya que, habiendo varios lenguajes que se basan de alguna u otra manera en este, la programación realizada del algoritmo puede ser fácilmente extrapolable a otros lenguajes. Además, la posibilidad de que los sistemas inteligentes diseñados puedan ser ejecutados en computadoras que posean distintos sistemas operativos, ya sea UNIX, Linux, o Windows, realizando apenas unas cuantas modificaciones.

Ambiente de desarrollo (IDE) Code::Blocks

Existen distintos ambientes de desarrollo que permiten programar en C, algunos son exclusivos para un sistema operativo, mientras que otros son multiplataforma. Ya que una de las razones principales de haber elegido el lenguaje C es su portabilidad entre distintos sistemas operativos, se utilizará un IDE que también lo sea, pues aunque los códigos fuente podrían abrirse y compliarse desde cualquier otro IDE, resulta más práctico tener el proyecto completo

en un ambiente que esté disponible para distintos sistemas operativos; por ello es que se eligió el ambiente de desarrollo Code::Blocks, este es gratis, multiplataforma, y de código abierto [52].

3.5 Programación

Los códigos fuente correspondientes al sistema se encontrarán disponibles en un repositorio de GitHub mediante el enlace: <https://github.com/Ernest0o/IA1_FC_con_GBJ>. En esta sección se hablará acerca de los programas ya creados, es decir, de las funciones y bibliotecas diseñadas para este algoritmo.

El programa completo quedó dividido en 3 bibliotecas: CSP_VectorUtilities, CSP_FlightUtilities, y main; así como sus respectivos «headers», y una definición de estructura alojada en un «header» para su uso tanto en main como en CSP_FlightUtilities. Se explicarán las funciones que contiene cada archivo fuente, sus respectivos usos y funcionamientos:

«CSP_VectorUtilities.c»

Esta biblioteca contiene funciones relacionadas al manejo y manipulación de vectores (listas). Permite hacer manipulaciones entre sus datos como reacomodos, eliminación de elementos, cálculo de elementos, operaciones entre elementos, y otras varias. Las funciones que contiene son las siguientes:

double horachar_adecimal: esta función realiza la conversión de una hora formada por una cadena de caracteres hh:mm en formato de 24 h, a un valor equivalente de tipo flotante. Ejemplo: Si la hora es 13:20, el resultado será 13.3333.

void convierte_vector_horaschar_a_decimal: convierte un vector completo que contenga horas en formato char hh:mm, a un vector que contiene valores equivalentes de tipo flotante. Utiliza consecutivamente la función horachar_adecimal. Se utiliza para convertir los vectores de horas de llegada y salida a valores numéricos válidos para poder realizar operaciones con ellos.

void calcula_duraciones: genera un vector de duraciones mediante la resta de la hora de llegada con la hora de salida de cada uno de los vuelos.

int calcula_tamano: esta función calcula el número de elementos que hay en un vector numérico. Estos elementos tienen que ser consecutivos y ninguno puede ser 0, ya que en cuanto la función detecta un 0 en el vector, asume que este no es un valor válido y allí detiene el conteo. Se utiliza para muchas funciones del algoritmo.

int encuentra_posicion: tiene como objetivo encontrar la posición en que se encuentra un elemento dentro de una lista, es principalmente utilizada para encontrar la posición de un ID en el vector de ID_vuelo. Ejemplo: si id_vuelo = [5, 3, 1, 4], encuentra_posición(1) dará como resultado 2; (pos. 0, pos. 1, pos. 2).

void swap: tiene una función sencilla, intercambiar dos valores internos de un vector, es decir, si un vector tiene los elementos [4, 3, 6, 7], swap (1, 2) reacomodará el vector como [4, 6, 3, 7]. Es una función auxiliar utilizada por las funciones ordena_por_hora y reacomoda_vector_randomly.

void ordena_por_hora: ordena un vector de forma ascendente, utiliza el método de burbuja para llevar a cabo el ordenamiento. Se utiliza normalmente para reordenar el vector vuelos_from_bases, y así elegir el vuelo que más temprano salga.

void agrega_a_solucion: únicamente se utiliza para agregar la ID de un vuelo al vector de vuelos cubiertos por la tripulación en cuestión, una vez que este ha sido elegido.

void copia_vuelos_restantes: copia los contenidos de un vector al vector ID_vuelos_restantes. Sirve para el reset completo del programa y para actualizar la lista cuando sea necesario.

void copia_dos_vectores: similar a la función anterior, esta copia los contenidos de cualquier vector origen hacia un vector destino.

void reacomoda_vector_randomly: reacomoda los contenidos de un vector de forma aleatoria mediante un número aleatorio de funciones swap, sirve para la etapa de elección de vuelo siguiente.

void filtra_tempc_a_base: hace el filtraje de la lista vuelos_FC_3, dejando en una lista los vuelos que salen de vuelta hacia la base de la que partió la tripulación.

void crea_archivo_csv: una vez generada la solución, el algoritmo crea un archivo de texto de salida con extensión csv que muestra las tripulaciones, los vuelos cubiertos de cada una, costos, y penalizaciones. Esta es la función encargada de ello.

«CSP_FlightUtilities.c»

En esta biblioteca se encuentran funciones que son de utilidad para el filtraje de vuelos, algunos reajustes de los vectores de horas de llegada/salida y, principalmente, contiene la función

que elige el vuelo siguiente, la cual es una de las más largas e importantes para el programa. Sus funciones son las siguientes:

void filtra_vuelos_bases: es la función encargada del llenado de la lista vuelos_from_bases, filtrando la de ID_vuelos_restantes, dejando sólo aquellos que partan de alguna de las bases con las que cuente la aerolínea.

void horas_departure_reajuste/horas_arrival_reajuste: son funciones que reajustan el valor flotante de las horas de partida y llegada. Básicamente consisten en que, si los vuelos salen o llegan en un día siguiente, se les aumenta 24 h a su valor, es decir 01.50 se convierte en 25.5; esto facilita los cálculos de duraciones y comparaciones.

void quita_vuelo_de_restantes: esta función elimina el ID del vuelo elegido de la lista ID_vuelos_restantes, además de reacomodarla para recorrer los vuelos posteriores a este y ocupen el espacio que dejó el quitado, con el objetivo de que la lista siempre mantenga los valores continuos.

int random_number: genera un número aleatorio que esté entre un rango de valores mínimo y máximo. La función es utilizada tanto para la elección del vuelo siguiente como para el reacomodo de vector aleatoriamente.

Struct siguiente_vuelo: esta es una de las funciones más importantes del programa, se encarga de asignar el vuelo siguiente mediante FC y heurísticas, además de orientar al programa principal sobre cómo continuar mediante las variables hay_retorno, atascado, y fin_tripulación.

«main.c»

El programa principal del algoritmo, contiene un par de funciones auxiliares para el leído del archivo de texto, y la función principal del sistema.

const char* getfield/int get_file_vuelos: funciones que se encargan de leer el archivo de texto donde se encuentra alojado el horario de vuelos, para posteriormente pasar los datos a los vectores encargados de contener esa información durante toda la ejecución del programa. Estas funciones leen inicialmente el primer renglón del archivo de texto, que contiene el orden en que se encuentran los valores (columnas); de esta manera se orientan y posteriormente llenan los vectores correspondientes. Los datos del archivo de texto se encuentran separados por «,», como lo ilustra el siguiente ejemplo:

```
Flight_id,origin,destination,departure,arrival

1,IST,ANK,07:00,08:00

2,IST,IZM,06:00,07:00

3,IZM,ANK,10:05,11:20

4,IST,ANT,08:25,09:40

5,IZM,ANK,19:20,20:40

…
```

void primer_vuelo: esta se encarga de escoger el primer vuelo de una tripulación. Básicamente actualiza y reordena la lista vuelos_from_bases con aquellos que no han sido cubiertos y escoge el primero de dicha lista, es decir, el más temprano de todos los que no han sido cubiertos.

void hard_reset: reinicia la construcción de la solución por completo, reinicia todas las variables, vectores y matrices ocupadas para el algoritmo, con el fin de evitar conflictos y posteriormente comenzar con la construcción desde cero.

int main: función principal del algoritmo, básicamente es la directora de orquesta que utiliza todas las funciones antes mencionadas para realizar la construcción de una solución. Dentro de esta también se encuentra el código de la técnica de GBJ.

3.6 Aportes y mejoras al algoritmo

Durante la etapa de programación se realizaron distintas consideraciones que en un principio pudieron no haber sido planteadas. Estas modificaciones pueden ser consideradas como aportes o mejoras al funcionamiento del sistema, pues, aunque no forman parte de las técnicas en sí, le son de gran ayuda al programa en general. En otras palabras, se implementaron algunas heurísticas que mejoran la búsqueda de la solución, el tiempo de ejecución, y ayudan al algoritmo a orientarse bajo ciertas circunstancias que fueron descubiertas conforme se progresaba en la programación. A continuación se hará un recuento y descripción de ellas, algunas ya han sido mencionadas con anterioridad:

- Reacomodo de vuelos por orden temporal: aunque podría elegirse cualquier vuelo de la lista *vuelos_from_bases* sin ningún criterio, se considera importante el reacomodo de vuelos por orden cronológico, para elegir siempre el vuelo que más temprano salga de

estos como el inicial para la siguiente tripulación. De esta manera se les da prioridad y se evita que estos se queden sin ser cubiertos por tripulaciones que salgan después.

- En la etapa de elección del vuelo siguiente, se colocó una consideración que se basa en las horas de trabajo y de vuelo restantes de la tripulación en cuestión, la cual es: si a la tripulación le queda menos de 3 horas de trabajo o menos de 2 horas de vuelo, el siguiente vuelo que cubrirá será forzosamente uno que la lleve de vuelta a base. Esta heurística procura mirar un poco más hacia el futuro y considera que, si se tiene menos tiempo restante del indicado, la tripulación ya no tendrá tiempo de cubrir ningún otro vuelo posterior al que se está por elegir, así que se opta por cubrir uno que la lleve de regreso y con este terminar su jornada de trabajo.
- En la etapa de GBJ, es importante que siempre se elija como vuelo siguiente uno distinto a los escogidos anteriormente, dado el caso de que se vuelva al mismo estado más de una vez. Por ello es que se creó una especie de copia de seguridad de los vuelos disponibles en ese estado, de la cual, siempre que se elige uno, se elimina de esta lista para que no exista la posibilidad de volver a elegirlo. Puede ocurrir un caso en que se haya regresado todas las veces posibles a un estado, y ya no quede ningún vuelo distinto por explorar; en situaciones así, la IA lo detectará y reiniciará la construcción de la tripulación.
- Penalizaciones por anomalías: como previamente es dicho, no todas las restricciones pueden tratarse como estrictas. Por la manera en que funciona el algoritmo, pueden ocurrir casos en que una tripulación haya cubierto únicamente un vuelo, y ya no tenga ninguno disponible para continuar ni para regresar a su base. Se le da prioridad a cubrir la mayor cantidad de vuelos disponible, así que en vez de desechar la tripulación y dejar su único vuelo sin cubrir, se opta por considerar que «la tripulación pasará el resto del día en la ciudad que se quedó» y se añade una penalización llamada *penalizacion_por_varado,* que se considera como los viáticos del «crew» para poder pasar el resto del día en la ciudad donde hayan terminado.
- Detección de bucles (1): pueden presentarse situaciones donde el algoritmo regrese en el tiempo mediante GBJ una y otra vez, sin lograr terminar de formar la tripulación exitosamente. Tomando en cuenta esta situación, se definió un número máximo de GBJ que se pueden hacer, si se llega al máximo, la tripulación se comenzará a reconstruir desde el primer vuelo. A esto se le conoce como el «reset» de una tripulación.

- Detección de bucles (2): dicho lo anterior, de igual manera pueden presentarse situaciones donde el algoritmo se encuentre reiniciando la tripulación varias veces sin haber podido construirla, así que también se definió un número máximo de «resets» de tripulación que se pueden realizar, si se llega al máximo, se realiza un reinicio completo (*hard_reset*), es decir, se desecha todo y se comienza la búsqueda desde el primer vuelo de la primera tripulación.

Y algunos otros detalles de menor relevancia puntuales para condiciones específicas, que evitan que, si se llega a una situación concreta perjudicial, la inteligencia artificial sepa cómo continuar y no quedarse estancada.

4. Diseño del segundo sistema inteligente con técnica incompleta

En este capítulo se habla acerca del diseño y programación del segundo algoritmo, el cual utiliza una técnica de tipo incompleta para encontrar soluciones al problema. La técnica elegida fue Simulated Annealing, por poseer un balance adecuado entre exploración y explotación, características que le permitirán realizar inicialmente una exploración aceptable por el espacio de búsqueda, y una explotación una vez que haya elegido el lugar del espacio para intensificar.

4.1 Planteamiento del algoritmo

Las técnicas incompletas que se encargan de reparar soluciones, incluyendo SA, necesitan: una representación de solución, es decir, la manera en que se muestran/almacenan las variables dentro ella; una solución inicial, de esta es que partirá todo su funcionamiento; y un movimiento que modificará la solución actual para generar la solución futura. A continuación se planteará cada una de sus características:

Representación binaria:

Se propone una representación binaria para la solución, la cual contendrá un 1 en el elemento *j* del vector de solución si la tripulación/crew/pairing *j+1* pertenece a ella, y un 0 en caso contrario. Se tendrá una matriz de tripulaciones y un vector de solución, como lo muestran las figuras 4.1.1 y 4.1.2:

Número de crew	ID de vuelos cubiertos por cada crew				
T1	3	5	8	9	23
T2	14	18			
T3	29	30	33		
…	…				
Tj	A	B	C	D	E

Figura 4.1.1. Matriz de «pairings» generados. Fuente: elaboración propia.

	T1	T2	T3	T4	…	Tj	…	Tn-1	Tn
Solución	0	1	1	0	…	Xj	…	X_{n-1}	Xn

Figura 4.1.2. Vector de solución. Xj = {0, 1}. Fuente: elaboración propia.

De las figuras y lo mencionado en el párrafo anterior se puede observar uno de los que serán los factores clave del funcionamiento del algoritmo, el hecho de que se tengan que generar *n* tripulaciones para posteriormente trabajar con ellas en el vector de solución, es decir, se tendrán dos etapas principales del algoritmo: la etapa de la generación de «pairings», y la etapa del manejo de estas sobre el vector de solución.

Solución inicial aleatoria:

La solución inicial de la que partirá el algoritmo será generada mediante la asignación aleatoria de 1s dentro del vector de solución; al vector de solución, que originalmente está lleno de 0s, se le colocará una cierta cantidad de 1s en posiciones aleatorias.

Movimiento «bit-flip»:

El movimiento elegido para el algoritmo es llamado «bit-flip», el cual invertirá el valor del elemento en la posición deseada del vector de solución, como lo ilustra la figura 4.1.3:

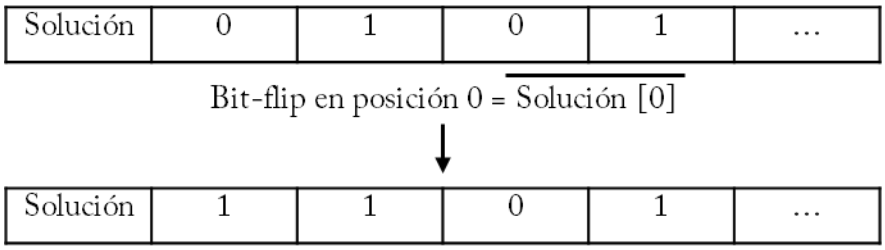


Figura 4.1.3. Ilustración del movimiento «bit-flip». — simboliza el operador lógico «not». Fuente: elaboración propia.

Una vez que se ha planteado la implementación de la técnica SA, se puede comenzar a plantear el funcionamiento del algoritmo por completo:

Igual que en el caso anterior, lo primero que se tiene que realizar es la obtención de los datos de los vuelos, provenientes de un horario alojado previamente en un archivo de texto, de este horario se deberán obtener:

- ID de los vuelos: para tener algún número identificador de cada vuelo.
- Número de vuelos del horario: obtenido del conteo de elementos en ID.
- Lugares de partida: uno asociado a cada ID de vuelo.

- Lugares de llegada: uno asociado a cada ID de vuelo.
- Horas de salida: una hora de salida asociada a cada ID.
- Horas de llegada: una hora de llegada asociada a cada ID.

Ya obtenida la información, el programa puede comenzar a trabajar con todos los datos recuperados.

Etapa 1 - Generación de tripulaciones:

La generación de «pairings» puede realizarse de distintas maneras. Podría, por ejemplo, hacerse de forma aleatoria asignando vuelos a las tripulaciones sin ningún tipo de consideraciones; sin embargo, realizarlo de esta manera provocaría que se generaran, en la mayoría de casos, tripulaciones infactibles, que sus horarios no respetaran varias o ninguna de las restricciones a nivel de tripulación, y que se perdiera mucho tiempo de forma innecesaria explorando por el espacio de lo infactible.

Por ello, la mejor alternativa es generar tripulaciones factibles desde un inicio, de esta manera el algoritmo se librará de penalizaciones por incumplimiento de restricciones a nivel de tripulación, y solamente lidiará con penalizaciones de vuelos que no fueron cubiertos, o «deadheadings», pues, en este sistema, a diferencia del anterior, podrá ocurrir el caso en que uno o varios vuelos aparezcan más de una vez entre los «crews» elegidos para la solución en cuestión.

Para aprovechar lo previamente hecho, se utilizará una versión simplificada del primer algoritmo diseñado para la etapa de generación de tripulaciones. Este consistirá en un sencillo FC que generará una tripulación a partir de un vuelo inicial, sin tomar en cuenta vuelos previamente cubiertos ni ninguna otra restricción; se propone el nombre *genera_crew* para la función. Esta se utilizará para generar cierta cantidad de tripulaciones por cada vuelo que parta de alguna de las bases, y así generar la *matriz_de_crews*, con la que el algoritmo trabajará en la siguiente etapa.

Etapa 2 - Simulated Annealing:

Paso 1: se realizará la creación de la *solución_inicial*, la cual, como se menciona previamente, será mediante la asignación de 1s en posiciones aleatorias dentro del vector; de igual manera se comenzará con una *temperatura* inicial alta. Posteriormente se calculará su costo, sumando los

costos de cada tripulación que forme parte de la solución, con las penalizaciones por vuelos sin cubrir y por «deadheadings».

Paso 2: se realizará el movimiento «bit-flip» en el primer elemento del vector *solución_inicial*, lo que dará lugar a una *solución_futura***.** Hecho el movimiento, se calculará el *costo_solución_futura,* y se comparará el costo de esta con el *costo_solución_inicial.* Aquí podrán ocurrir dos situaciones:

Paso 3: si la *solución_futura* tiene un costo menor o igual al de la *solución_inicial*, ya que la función objetivo es minimizar, la *solución_futura* se convertirá en la nueva *solución_inicial*, y se volverá al paso 2, ahora tomando a la *solución_futura* como la solución inicial a la que se le realizarán los movimientos. Además, se disminuirá la *temperatura* en un porcentaje.

Paso 4: si la *solución_futura* tiene un costo mayor al de la *solución_inicial*, es cuando se tomará en cuenta la temperatura y la probabilidad. Se calculará la *probabilidad* mediante la operación $e^{\frac{-\Delta f}{T}}$, donde $\Delta f = costo\_solución\_futura - costo\_solución\_inicial$, y T es la *temperatura*. Después, se generará un número aleatorio entre 0 y 1, y se podrán presentar dos situaciones más:

Paso 5: si la *probabilidad* es mayor o igual al número aleatorio, la *solución_futura* se convertirá en la nueva *solución_inicial*, y se volverá al paso 2.

Paso 6: si la *probabilidad* resultó menor al número aleatorio, la *solución_futura* será descartada y la *solución_inicial* se mantendrá intacta. Se volverá al paso 2, pero esta vez se realizará el movimiento en la posición siguiente del vector, no en la que se realizó con anterioridad. Además, la *temperatura* disminuirá en un porcentaje.

Durante toda la ejecución del programa se tendrá una variable *mejor_solución*, con la cual toda *solución_futura* se comparará, y se sustituirá por esta en caso de ser mejor, el objetivo será tener siempre en memoria la mejor solución encontrada hasta el momento.

La etapa 2 se mantendrá iterando, es decir, creando soluciones futuras, un *número_máximo_de_iteraciones* establecido en los parámetros del programa; este número será uno de los que se varíen para la etapa de experimentación.

Una vez que se ha alcanzado el *número_máximo_de_iteraciones*, la búsqueda finaliza y se mostrará la *mejor_solución_encontrada*, pues no se garantiza que la solución de la última iteración sea la mejor de toda la ejecución del programa. El porqué es sencillo, es posible que SA haya podido

haber encontrado un excelente óptimo local en alguna etapa intermedia de exploración, y haber seguido explorando, y este óptimo ser mejor que el encontrado ya que el sistema intensificó en una zona fija del espacio. Por ello es indispensable el hecho de recordar la mejor solución hallada durante todo el algoritmo.

4.2 Diagrama de flujo

En esta sección se muestra la representación en diagrama de flujo del funcionamiento del sistema. Al igual que en el caso anterior, el diagrama es una versión simplificada del algoritmo, que no toma en cuenta cada una de las funciones complementarias necesarias para toda la ejecución, únicamente muestra los procedimientos más relevantes. La figura 4.2 se encuentra dividida en 2 partes, la parte 4.2a corresponde al inicio del programa y la generación de tripulaciones, mientras que la parte 4.2b corresponde a la etapa de SA y fin del programa:

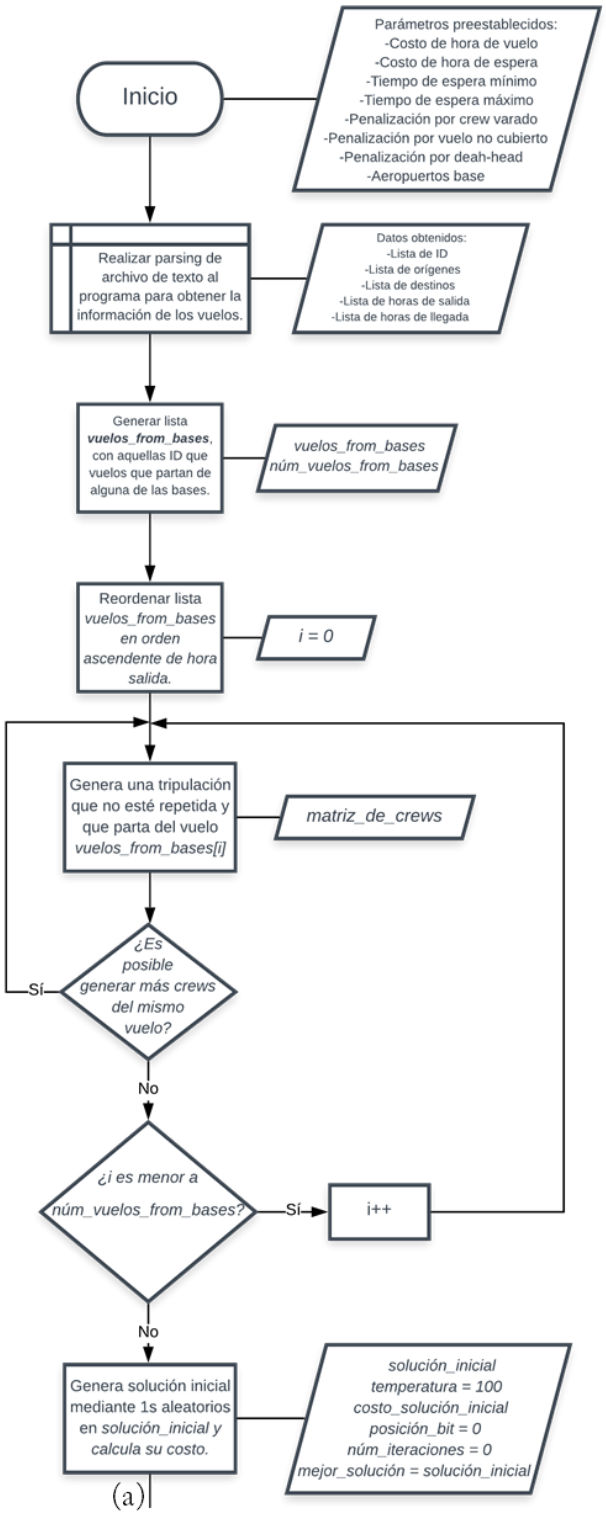


Figura 4.2a. Inicio del segundo sistema y etapa de generación de «pairings». Fuente: elaboración propia.

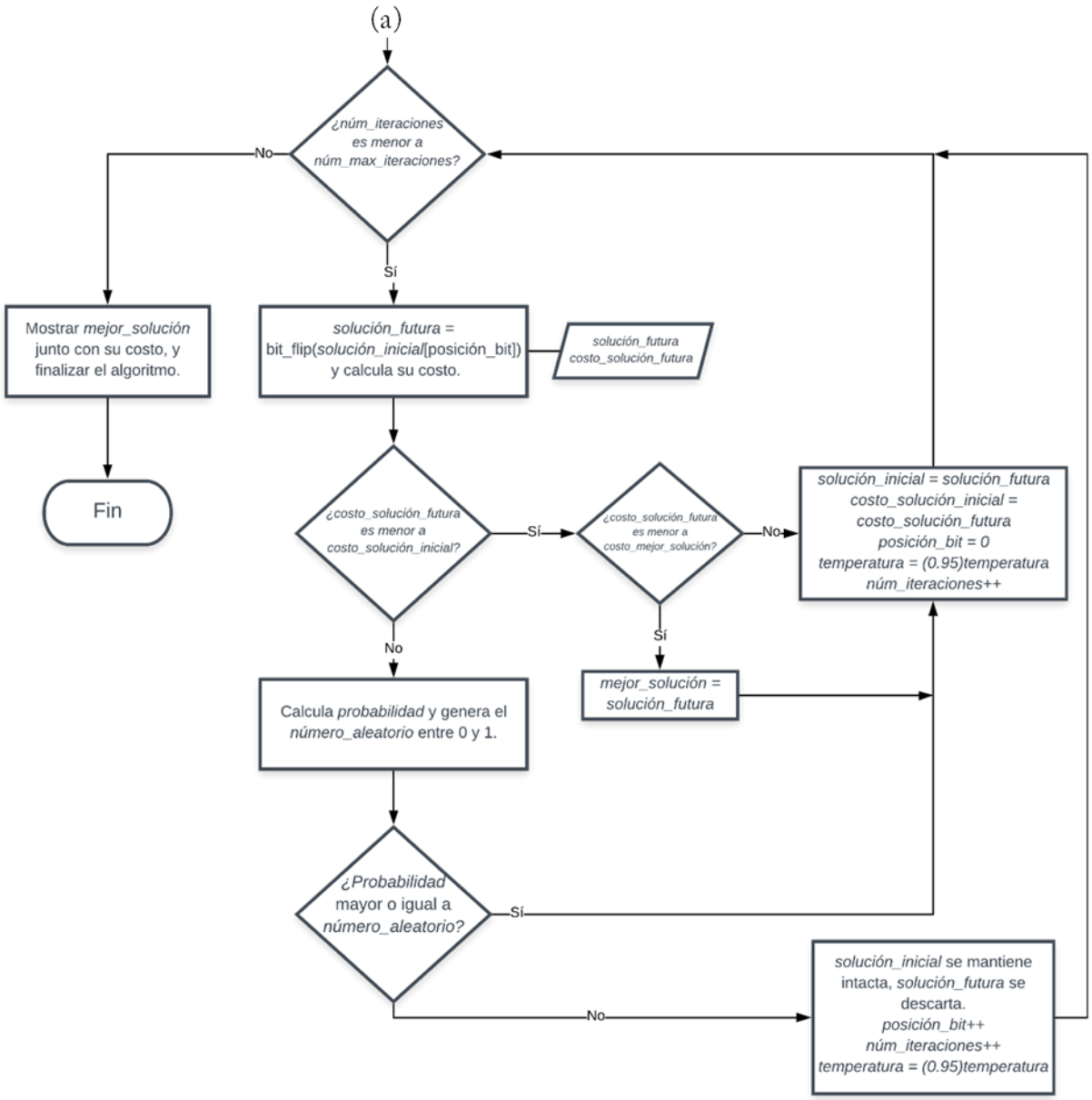


Figura 4.2b. Etapa de Simulated Annealing y fin del segundo sistema. Fuente: elaboración propia.

4.3 Representación en pseudocódigo

La representación en pseudocódigo es un acercamiento más profundo al funcionamiento del sistema completo, servirá como guía para la programación directa, ya que proporciona una idea más clara sobre los procedimientos que se tienen que realizar durante la ejecución. El pseudocódigo del segundo sistema queda de la siguiente manera:

```
función primer_vuelo (start_flight) { //El primer vuelo de un crew.
	vuelo_inicial = start_flight;
	reinicia los valores de vuelos_crew, horas_de_trabajo, horas_de_vuelo,
	horas_de_espera;
	horas_de_vuelo = duración de vuelo_inicial;
	horas_de_trabajo = horas_de_vuelo + horas_de_espera;
	agrega a vuelos_crew el vuelo_inicial;
	quita el vuelo_inicial de ID_vuelos_restantes;
	pos_solución = 0; //Número de vuelos en el vector vuelos_crew
}

función siguiente_vuelo (vuelo_anterior){ //Elige el siguiente vuelo a partir del
anterior mediante FC.
//FC p1: Destino de vuelo_anterior = origen del vuelo_siguiente
	lista vuelos_FC_1 = obtén de ID_vuelos_restantes los que salgan de la ciudad
	destino(vuelo_anterior);
//FC p2: Restricciones de tiempo de espera mínimo y máximo
	lista vuelos_FC_2 = obtén de vuelos_FC_1 aquellos que: (hora_de_salida >=
	hora_de_llegada(vuelo_anterior) + tiempo_mínimo_de_espera) &&
	(hora_de_salida <= hora_de_llegada(vuelo_anterior) +
	tiempo_máximo_de_espera);
//FC p3: Aquellos vuelos que, si se eligen, el crew no exceda sus horas //máximas
de trabajo, ni sus horas máximas de vuelo.
	lista vuelos_FC_3 = obtén de vuelos_FC_2 aquellos que, si se eligen, no
	excedan las horas máximas de trabajo ni de vuelo del crew, considerando su
	duración y el tiempo de espera;
//si no hay vuelos disponibles, que revise si está en base
	si vuelo_FC_3 no tiene elementos {
		si el destino vuelo_anterior los dejó en su base {
			fin_tripulación = true;
			sig_vuelo = 0; //null
		} si no es así {
			si sólo han cubierto un vuelo { //Si sólo han cubierto un vuelo
			y no pueden elegir ningún otro, es imposible generar el pairing
			(no se aceptan pairings de un solo vuelo)
```

```
                        imposible_generar_crew = true;
                }
                atascado = true;
        }
    } si hay por lo menos un elemento en vuelos_FC_3 {
        //Elección del vuelo siguiente mediante heurísticas
        si (horas_de_trabajo_restantes >= 3) && (horas_de_vuelo_restantes >=
        2) {
                fin_tripulación = false;
                si número de elementos en vuelos_FC_3 > 1 {
                        sig_vuelo = cualquiera de lista vuelos_FC_3;
                } si sólo es 1 elemento {
                        sig_vuelo = primer elemento de vuelos_FC_3;
                }
        //Si no le quedan más de 3 y 2 horas restantes:
        } en otro caso {
                si hay algún vuelo de vuelta a base en vuelos_FC_3 {
                        sig_vuelo = alguno que los regrese a base;
                        fin_tripulación = true;
                } si no hay ningún vuelo de vuela {
                        si ya se encuentran en su base {
                                //Que allí se queden.
                                fin_tripulación = true;
                                sig_vuelo = 0;
                        } si no están en su base {
                                atascado = true;
                        }
                }
        }
    }
    devuelve sig_vuelo, atascado, fin_tripulación, hay_retorno,
    imposible_generar_crew;
}

función genera_crew (start_flight) { //Genera un crew a partir de un start_flight.
    primer_vuelo(start_flight);
    mientras fin_tripulación sea falso && imposible_generar_crew también sea
    falso, haz {
        siguiente_vuelo(vuelo_anterior);
```

```
            si atascado es true {
                  que reinicie la construcción del crew;
            }
            //Si ha elegido un vuelo válido…
            si atascado es falso && sig_vuelo > 0 {
                  vuelo_anterior = vuelos_crew[pos_solución];
                  pos_solución ++;
                  agrega a vuelos_crew el sig_vuelo;
                  incrementa horas_de_vuelo con la duración del vuelo;
                  incrementa horas_de_espera con el tiempo de espera;
                  horas_de_trabajo = horas_de_vuelo + horas_de_espera;
            }
      }
      //fin_tripulación ya es true:
      si imposible_generar_crew es falso {
            vector crew_generado = vector vuelos_crew;
            calcula costo_crew con las horas_de_trabajo y horas_de_espera;
      }
      devuelve imposible_generar_crew;
}

función genera_set_de_crews (vuelo_inicial) { //Función que genera un set de crews
que parten de un vuelo_inicial
      haz {
            crew_noposible = genera_crew (vuelo_inicial);//Si es true, significa
            que no se ha podido generar el pairing por algún motivo
            mientras el crew_generado sea igual a uno de los creados anteriormente
            && crew_noposible sea falso && num_reconstruciones <=
            num_max_reconstrucciones, haz{
                  crew_noposible = genera_crew (vuelo_inicial);
                  num_reconstrucciones++; //número de veces que se ha
                  reconstruido un crew por estar repetido
            }
            si num_reconstrucciones <= num_max_reconstrucciones && crew_noposible
            es falso {
                  agrega el crew_generado al set de crews generados;
                  agrega su costo al vecor_costos_crew;
                  num_crews_por_vuelo++;
            }
      } mientras num_reconstrucciones <= max_reconstrucciones &&  crew_noposible
      sea falso && num_crews_por_vuelo < num_max_crews_por_vuelo;
```

```
}

función bit_flip (elemento_booleano) {
	si elemento_booleano es verdadero {
		elemento_booleano = false;
	} si elemento_booleano es falso {
		elemento_booleano = true;
	}
}

int main () { //Programa principal del algoritmo
	haz el parsing del archivo de texto y obtén la información de los
	vuelos, llenando los vectores: ID_vuelo, origin, destination, departure,
	arrival;
	convierte las horas de salida y llegada, originalmente de tipo char, a
	valores de tipo flotante, para que sean valores numéricos y no cadenas de
	caracteres;
	llena la lista vuelos_from_bases, filtrando los vuelos que parten de alguna
	de las bases;
	calcula el número de elementos en vuelos_from_bases y almacénalo en la
	variable num_vuelos_from_bases;
	ordena de forma ascendiente la lista vuelos_from_bases, por hora de salida;
	llena el vector de duraciones mediante la resta de la hora de llegada y
	salida de cada vuelo;
	//--Comienza la generación de pairings factibles --//
	for (int i = 0; i<num_vuelos_from_bases; i++){
		genera_set_de_crews (vuelos_from_bases[i]);
		//-Ya que se ha generado el set de crews-//
		añade los crews generados a la matriz_de_crews;
	}
	//-Comienza el Simulated Annealing-//
	calcula el total de crews generados, para saber el tamaño que tendrá el
	vector de solución;
	posición_bit = 0;
	temperatura = 100; //Valor inicial de temperatura
	coloca 1s aleatorios en el vector solución_inicial;
	calcula el costo_solución_inicial mediante la suma de los costos de los
	pairings, más las penalizaciones por vuelos sin cubrir y vuelos deadhead;
	//Al ser la primera solución, automáticamente es la mejor:
	vector mejor_solución = solución_inicial;
	costo_mejor_solución = costo_solución_inicial;
```

```
    //Comienza la búsqueda de soluciones:
    for (iter = 0; iter<num_max_iteraciones; iter++){
        solución_futura = bit_flip (solución_inicial[posición_bit]);
        calcula el costo_solución_futura;
        posición_bit ++;
        si costo_solución_futura < costo_solución_inicial {
            si costo_solución_futura < costo_mejor_solución {
            //Actualización de la mejor_solución hasta el momento:
                mejor_solución = solución_futura;
                costo_mejor_solución = costo_solución_futura;
            }
            //Aceptación de la solución generada
            vector solución_inicial = vector solución_futura;
            costo_solución_inicial = costo_solución_futura;
            posición_bit = 0;
        } si el costo_solución_futura > costo_solución_inicial {
            probabilidad = e^-[(costo_solución_futura –
            costo_solución_inicial)/temperatura];
            genera número_random;
            si probabilidad >= número_random {
                //Aceptación de la solución generada por prob.
                vector solución_inicial = vector solución_futura;
                costo_solución_inicial = costo_solución_futura;
                posición_bit = 0;
            } //En caso contrario, no se acepta
        }
        temperatura = coeficiente_temperatura*temperatura;
        //coeficiente_temperatura siempre menor a 1. Ej: 0.92
    }
    //Una vez que se ha alcanzado el máximo número de iteraciones:
    exporta la mejor_solución en un archivo de texto, incluye los vuelos que
    cubre cada tripulación, sus costos, las ID de los vuelos sin cubrir, los
    vuelos deadhead, y el costo total de la mejor_solución encontrada;
}
```

4.4 Programación

Para mantener uniformidad con el primer proyecto, este se realizará de la misma manera que el anterior; se programará en lenguaje C utilizando el ambiente de desarrollo Code::Blocks. El proyecto se encontrará disponible en el repositorio GitHub del enlace:

<https://github.com/Ernest0o/IA2_SA>. En esta sección se comentará sobre las funciones creadas para el segundo algoritmo.

El proyecto fue programado modularmente en cuatro bibliotecas; una biblioteca más que en el proyecto anterior, la cual contiene funciones que solamente el algoritmo con SA utiliza. Los nombres de los archivos fuente son: CSP_VectorUtilities, CSP_FlightUtilities, CSP_IA2Utilities, y main. También se crearon sus respectivos «headers», y un «header» extra para la definición de la estructura utilizada en las funciones *siguiente_vuelo* y *genera_crew*. Las funciones que posee cada biblioteca son algunas exactamente iguales a las del primer proyecto, y algunas adicionales.

«CSP_VectorUtilities.c»

Este archivo fuente contiene funciones para el manejo y manipulación de vectores, operaciones entre elementos, reacomodos, conteos, entre otras varias. Las funciones que contiene son las siguientes:

double horachar_adecimal: esta función realiza la conversión de una hora formada por una cadena de caracteres hh:mm en formato de 24 h, a un valor equivalente de tipo flotante. Ejemplo: Si la hora es 13:20, el resultado será 13.3333.

void convierte_vector_horaschar_a_decimal: convierte un vector completo que contenga horas en formato char hh:mm, a un vector que contiene valores equivalentes de tipo flotante. Utiliza consecutivamente a la función horachar_adecimal. Se utiliza para convertir los vectores de horas de llegada y salida a valores numéricos válidos para poder realizar operaciones con ellos.

void calcula_duraciones: genera un vector de duraciones mediante la resta de la hora de llegada con la hora de salida de cada uno de los vuelos.

int calcula_tamano: esta función calcula el número de elementos que hay en un vector numérico. Estos elementos tienen que ser consecutivos y ninguno puede ser 0, ya que en cuanto la función detecta un 0 en el vector, asume que este no es un valor válido y allí detiene el conteo. Se utiliza para muchas funciones del algoritmo.

int calcula_renglones: es una función que calcula el número de renglones, es decir, el número de elementos horizontales de una matriz.

int encuentra_posicion: tiene como objetivo encontrar la posición en que se encuentra un elemento dentro de una lista, es principalmente utilizada para encontrar la posición de un ID en el vector de ID_vuelo. Ejemplo: id_vuelo = [5, 3, 1, 4], encuentra_posición(1) dará como resultado 2; (pos. 0, pos. 1, pos. 2).

void swap: tiene una función sencilla, intercambiar dos valores internos de un vector, es decir, si un vector tiene los elementos [4, 3, 6, 7], swap (1, 2) reacomodará el vector como [4, 6, 3, 7]. Es una función auxiliar utilizada por las funciones ordena_por_hora y reacomoda_vector_randomly.

void ordena_por_hora: ordena un vector de forma ascendente, utiliza el método de burbuja para llevar a cabo el ordenamiento. Se utiliza normalmente para reordenar el vector vuelos_from_bases.

void agrega_a_solucion: únicamente se utiliza para agregar la ID de un vuelo al vector de vuelos cubiertos por la tripulación en cuestión, una vez que este ha sido elegido.

void copia_vuelos_restantes: copia los contenidos de un vector al vector ID_vuelos_restantes. Sirve para el reset completo del programa y para actualizar la lista cuando sea necesario.

void copia_dos_vectores: similar a la función anterior, esta copia los contenidos de cualquier vector origen hacia un vector destino.

void reacomoda_vector_randomly: reacomoda los contenidos de un vector de forma aleatoria mediante un número aleatorio de funciones swap entre elementos en posiciones también aleatorias, sirve para la etapa de elección de vuelo siguiente y reacomodo los «pairings» generados.

void filtra_tempc_a_base: hace el filtraje de la lista vuelos_FC_3, dejando en una lista los vuelos que salen de vuelta hacia la base de la que partió la tripulación.

bool compara_dos_vectores: es una función que se encarga de comparar uno a uno los elementos de dos vectores para ver si estos son iguales. Devuelve un 1 en caso de que los vectores sean iguales y un 0 en caso contrario. Es utilizada por la función *revisa_si_esta_en_matriz*.

bool revisa_si_esta_en_matriz: revisa si un vector ingresado se encuentra ya los vectores de una matriz existente. Utiliza la función *compara_dos_vectores* el número de veces que sea necesario,

según el tamaño de la matriz. Se utiliza para corroborar que no se generen tripulaciones repetidas.

int suma_elementos_de_vector: es una función sencilla, que se encarga de devolver la suma de los elementos que están dentro de un vector. Ej: v = {1, 3, 5, 9} ; Suma = 18.

«CSP_FlightUtilities.c»

Contiene funciones que son necesarias para la elección de vuelos que se adecúen a la tripulación que se esté generando, ya sea mediante filtrajes, reacomodos, y la función más importante: la que elige el vuelo siguiente de la tripulación. Las funciones que contiene son las siguientes:

void filtra_vuelos_bases: es la función encargada del llenado de la lista vuelos_from_bases, filtrando la de ID_vuelos_restantes, dejando sólo aquellos que partan de alguna de las bases con las que cuente la aerolínea.

void horas_departure_reajuste/horas_arrival_reajuste: son funciones que reajustan el valor flotante de las horas de partida y llegada. Básicamente consisten en que, si los vuelos salen o llegan al día siguiente, se les aumenta 24 h a su valor, es decir 01.50 se convierte en 25.5, esto facilita los cálculos.

int random_number: genera un número aleatorio que esté entre un rango de valores mínimo y máximo. La función es utilizada tanto para la elección del vuelo siguiente como para el reacomodo de vector aleatoriamente.

Struct siguiente_vuelo: se encarga de asignar el vuelo siguiente mediante FC y heurísticas, además de orientar a la función *genera_crew* sobre cómo continuar mediante las variables hay_retorno, atascado, fin_tripulación y una variable extra llamada imposible_generar_crew, que indica si no es posible generar un pairing factible desde un vuelo de partida en concreto.

«CSP_IA2Utilities.c»

Esta biblioteca posee funciones de utilidad para la segunda IA en particular, las cuales tienen que ver con el funcionamiento directo del algoritmo SA. Las funciones que se encuentran dentro de esta biblioteca son:

void coloca_1s_aleatorios: como su nombre lo indica, esta función coloca una cantidad de 1s en posiciones aleatorias dentro de un vector, cuidando que la posición no sea la misma para que el número de 1s se respete. Es utilizada para generar la solución inicial de la que parte el algoritmo.

void solucion_bool_a_decimal: se encarga de pasar una solución que se encuentra en formato binario, a un vector que contenga los números de tripulaciones por los que está compuesta la misma, y así sea más fácil su manejo. Ej: solución_bool = {1, 0, 0, 1, 0, 1}; solución_decimal = {1, 4, 6}.

double calcula_costo_solucion: función que se encarga de calcular el costo de una solución. Hace la suma de los costos de los «crews» que conforman la solución, y les añade las penalizaciones por «deadheading» y por vuelos no cubiertos.

void backup_best_solucion: esta función actualiza la mejor solución encontrada por el algoritmo, así como su costo, cuando el programa principal detecta que ha encontrado una mejor solución que la que se había encontrado hasta ese momento.

void copia_dos_vectores_bool: función similar a *copia_dos_vectores*, con la diferencia de que esta trabaja con vectores booleanos.

void bit_flip: se encarga de invertir el valor binario que se encuentra en la posición indicada del vector deseado. Su funcionamiento es simplemente la operación lógica «not» del elemento en posición deseada del vector de solución booleano.

bool eleccion_por_simulated_annealing: es la función que decide si una solución peor a la actual se escogerá como la siguiente, dependiendo de la probabilidad y del número aleatorio generado. Devolverá un 1 si ha decidido que la solución se acepta, y un 0 en caso contrario.

void recuento_y_creacion_csv: muestra en pantalla la información de la mejor solución encontrada una vez que se ha alcanzado el máximo de iteraciones. Asimismo, crea el archivo de salida «.csv» que contiene toda la información de tal solución: tripulaciones, vuelos asignados a cada una, costos, y penalizaciones.

«main.c»

Biblioteca principal del sistema, contiene funciones para la obtención de información del horario, generación de tripulaciones, y la función principal del sistema; todas en conjunto

utilizan las funciones de las otras bibliotecas para llevar a cabo la ejecución completa del algoritmo. Las funciones que se encuentran en main son las siguientes:

const char* getfield/int get_file_vuelos: funciones que se encargan de leer el archivo de texto donde se encuentra alojado el horario de vuelos, para posteriormente pasar los datos a los vectores encargados de contener esa información durante toda la ejecución del programa. Estas funciones leen inicialmente el primer renglón del archivo de texto, que contiene el orden en que se encuentran los valores (columnas); de esta manera se orientan y posteriormente llenan los vectores correspondientes. Los datos del archivo de texto se encuentran separados por «,», como lo ilustra el siguiente ejemplo:

```
Flight_id,origin,destination,departure,arrival

1,IST,ANK,07:00,08:00

2,IST,IZM,06:00,07:00

3,IZM,ANK,10:05,11:20

4,IST,ANT,08:25,09:40

5,IZM,ANK,19:20,20:40

…
```

void primer_vuelo: esta se encarga de colocar el vuelo inicial en el vector de vuelos cubiertos por la tripulación en cuestión, dado que el primer vuelo ya es proporcionado específicamente, *primer_vuelo* ya no lo elige, solamente introduce su ID al vector de vuelos cubiertos, para después continuar.

void hard_reset: reinicia la construcción de la tripulación, reinicia todas las variables, vectores y matrices ocupadas para su creación, con el fin de evitar conflictos y posteriormente comenzar con la construcción desde cero.

bool genera_crew: es la función encargada de crear una tripulación a partir de un vuelo inicial, utiliza a las funciones *primer_vuelo* y *siguiente_vuelo* hasta generar un «crew» completo. En caso de que no sea posible la construcción, se devolverá un 1, porque la variable devuelta es *imposible_generar_crew*, proveniente de la función *siguiente_vuelo*.

void genera_set_de_crews: genera un número de «pairings» factibles y no repetidos a partir de un vuelo inicial. El número máximo de «pairings» que genera está restringido por un parámetro llamado *max_num_crews_por_vuelo*, o por la imposibilidad de crear «pairings» nuevos, lo que ocurra primero.

int main: función principal de todo el algoritmo. La directora de orquesta que lleva a cabo las dos etapas principales del sistema: la generación de tripulaciones, y la manipulación de ellas mediante SA.

4.5 Aportes y mejoras al algoritmo

De la misma manera que en el sistema anterior, conforme se progresó en la etapa de programación, se fueron descubriendo distintas situaciones que llevaron a tomar en cuenta varias consideraciones con el objetivo de mejorar el funcionamiento del algoritmo. En esta sección se enlistan y describen las heurísticas y mejoras implementadas para la segunda inteligencia artificial:

Para la etapa de generación de «pairings»:

- Reducción del espacio de búsqueda (1): el espacio de búsqueda está directamente ligado a la cantidad de tripulaciones generada. Recordando, es el producto de los dominios de todas las variables. Puesto que el dominio para todas las variables en este caso es 0 o 1, es decir, 2, el espacio de búsqueda queda como 2^n, donde n es el número de «crews» generados. Aumentar una variable (crear una tripulación más), significa duplicar el tamaño del espacio de búsqueda, por ello es que se colocó un máximo de tripulaciones por vuelo, con el fin de acotar en lo posible el espacio de búsqueda y no caer en una «explosión combinatoria»[3].
- Reducción del espacio de búsqueda (2): la creación de sólo tripulaciones factibles es una gran ventaja para el algoritmo, le evita estar explorando por zonas del espacio de búsqueda que incumplan la mayoría de restricciones, dejando que navege únicamente en zonas donde sólo pueden existir «deadheadings» y vuelos sin cubrir.
- Reducción del espacio de búsqueda (3): aparte de la creación de tripulaciones factibles, el algoritmo también se asegura de generar tripulaciones diferentes, ninguna se repetirá.

[3] Una explosión combinatoria se produce cuando el «trabajo» requerido para solucionar un problema se ve incrementado de forma agresiva (como si explotara), a medida que el tamaño del problema aumenta [53].

De esta manera se evitan situaciones donde dos tripulaciones que formen una solución sean exactamente iguales, reduciendo directamente el número de «deadheadings».

- Es posible que haya casos donde no sea posible generar el número máximo de tripulaciones por vuelo sin que estas sean repetidas. Tomando en cuenta esta consideración, se colocó un número máximo de reconstrucciones, si pasado este número no se ha podido generar una tripulación que no haya sido creada previamente, no se crearán más tripulaciones a partir de ese vuelo inicial y se pasará al siguiente.
- También es posible que no haya manera de crear ninguna tripulación factible a partir de un vuelo inicial, es decir, dicho vuelo tiene que ser cubierto por alguna otra tripulación en sus vuelos intermedios, pues no hay manera de que este sea el primero de ningún «pairing». El algoritmo es capaz de detectar si esta situación se presenta, y no se quedará atascado intentando generar tripulaciones para vuelos iniciales que no son posibles.

Para la etapa de Simulated Annealing:

- Aunque la etapa de generación de tripulaciones se realiza de forma ordenada, es decir, los «pairings» están acomodados por «sets» que parten de un mismo vuelo, para la etapa de SA todos se reacomodan de forma aleatoria. El objetivo del reacomodo es reducir el número de «deadheads» por movimiento, ya que como el movimiento avanza de posición consecutivamente, el hecho de que se tengan dos 1s seguidos, si no se reacomodan las tripulaciones, resulta en una alta probabilidad de que sean 2 tripulaciones que partan de un mismo vuelo y que haya por lo menos un «deadhead»; al reacomodar los «crews» aleatoriamente se reduce esta posibilidad.
- El número de 1s que se colocan para generar la solución inicial es calculado mediante la operación: $número\_de\_unos = número\_de\_vuelos/6$. Se considera que una solución con una cantidad menor de 1s que la indicada, es bastante pobre en calidad, mucho más lo es una solución que no contenga ningún 1, por eso se opta por esa cantidad inicial de 1s.
- Sin retorno al pasado: puede ocurrir que el movimiento en la solución actual dé lugar a una solución que ya haya sido generada anteriormente, y esta pueda aceptarse por causas de la probabilidad y la alta temperatura. La inteligencia artificial es capaz de saber si la

solución generada por el movimiento es igual a alguna generada en el pasado, para así no elegirla y continuar con el movimiento siguiente.

- Penalizaciones por anomalías: debido a que los «pairings» son generados de manera independiente entre ellos, existen varios que cubren algún o varios vuelos iguales a los otros. La posibilidad de «deadheading» es más que real en este segundo sistema, así que las soluciones que presentan este tipo de conflictos son penalizadas mediante el parámetro *penalización_por_deadhead*, con el objetivo de que este fenómeno tenga un peso directo sobre la calidad de la solución.

Entre algunas otras consideraciones menores para situaciones específicas, que orientan al programa para saber cómo continuar y no quedarse atascado en ningún momento.

5. Experimentación

En el presente capítulo se relata la etapa de experimentos realizados en ambos algoritmos. Se describe el protocolo de pruebas llevado a cabo, que incluye: los horarios elegidos, los parámetros configurados, la forma de ejecución; así como la sección respectiva de variación de parámetros, donde se realizan experimentos con distintos valores en los parámetros de las restricciones y de ejecución en los algoritmos. Los resultados de la etapa de experimentación están alojados y comentados en el capítulo correspondiente; en este, a grandes rasgos, sólo se comenta la manera en que las pruebas fueron realizadas.

5.1 Protocolo de pruebas

Todos los experimentos, así como el desarrollo de ambos proyectos, fueron realizados en una computadora con sistema operativo Windows 10 y las siguientes características de hardware:

Procesador: AMD FX 6300; 6 núcleos a 3.5 GHz.
Memoria: RAM DDR3 «single-channel» de 8 GB.

Como es de suponerse, para realizar experimentos con los sistemas, se debe contar con un horario de vuelos, y así estos puedan llevar a cabo la lectura del archivo del horario y realicen la planificación de tripulaciones correspondiente.

Los horarios elegidos para la etapa de experimentación fueron extraídos del artículo «Robust Crew Pairing for Managing Extra Flights», el cual proporciona 3 horarios diferentes para jornadas de un solo día; un horario contiene 38 vuelos, otro 58, y uno con 96. Los horarios se encuentran en las tablas 5.1.1, 5.1.2 y 5.1.3 respectivamente:

ID	Origin	Destination	Departure	Arrival	ID	Origin	Destination	Departure	Arrival
1	IST	ANK	07:00	08:00	20	IST	ANK	15:00	16:00
2	IST	IZM	06:00	07:00	21	IST	IZM	17:00	18:00
3	IZM	ANK	10:05	11:20	22	IZM	IST	19:20	20:20
4	IST	ANT	08:25	09:40	23	IST	ANK	17:00	18:00
5	IZM	ANK	19:20	20:40	24	IZM	IST	22:00	23:00
6	IZM	IST	09:00	10:00	25	IST	IZM	20:00	21:00
7	ANT	IST	11:00	12:10	26	IST	ANK	19:00	20:00

8	IST	ANT	14:25	15:50	27	IST	ANK	22:00	23:00
9	IST	IZM	09:00	10:00	28	IST	ANK	23:45	00:45
10	IST	IZM	11:00	12:00	29	ANK	IZM	07:45	09:05
11	ANT	IST	16:50	18:05	30	ANK	IZM	17:00	18:20
12	IZM	IST	11:00	12:00	31	ANK	IST	08:00	09:00
13	IST	ANK	11:00	12:00	32	ANK	IST	11:00	12:00
14	IST	ANT	19:00	20:15	33	ANK	IST	14:00	15:00
15	IST	IZM	13:00	14:00	34	ANK	IST	17:00	18:00
16	IZM	IST	13:00	14:00	35	ANK	IST	13:00	14:00
17	IST	ANK	13:00	14:00	36	ANK	IST	21:00	22:00
18	ANT	IST	21:15	22:30	37	ANK	IST	20:00	21:00
19	IZM	IST	15:00	16:00	38	ANK	IST	22:00	23:00

Tabla 5.1.1. Horario de 38 vuelos. Fuente: «Robust Crew Pairing for Managing Extra Flights» [40].

ID	Origin	Destination	Departure	Arrival	ID	Origin	Destination	Departure	Arrival
1	IST	ADA	07:00	08:35	30	ANT	IST	10:40	11:55
2	IST	ADA	11:15	12:50	31	ANT	IST	13:40	14:55
3	IST	ADA	14:15	15:50	32	ANT	IST	16:45	18:00
4	ADA	IST	09:35	11:10	33	IST	BOD	14:30	15:40
5	ADA	IST	13:50	15:25	34	IST	BOD	19:30	20:40
6	ADA	IST	16:50	18:25	35	BOD	IST	16:45	17:55
7	IST	ANK	07:00	08:00	36	BOD	IST	21:40	22:50
8	IST	ANK	09:00	10:00	37	IST	DAL	17:20	18:40
9	IST	ANK	10:00	11:00	38	IST	DAL	19:35	20:50
10	IST	ANK	13:00	14:00	39	DAL	IST	19:40	21:00
11	IST	ANK	15:00	16:00	40	DAL	IST	21:50	23:05
12	IST	ANK	17:00	18:00	41	IST	IZM	07:00	08:00
13	IST	ANK	17:30	18:30	42	IST	IZM	08:00	09:00
14	IST	ANK	18:00	19:00	43	IST	IZM	08:30	09:30
15	IST	ANK	19:00	20:00	44	IST	IZM	11:00	12:00
16	IST	ANK	20:00	21:00	45	IST	IZM	13:00	14:00
17	ANK	IST	09:00	10:00	46	IST	IZM	15:00	16:00
18	ANK	IST	11:00	12:00	47	IST	IZM	16:00	17:00
19	ANK	IST	12:00	13:00	48	IST	IZM	20:00	21:00

20	ANK	IST	15:00	16:00	49	IST	IZM	23:45	00:45
21	ANK	IST	17:00	18:00	50	IZM	IST	10:00	11:00
22	ANK	IST	19:00	20:00	51	IZM	IST	10:30	11:30
23	ANK	IST	19:30	20:30	52	IZM	IST	11:00	12:00
24	ANK	IST	20:00	21:00	53	IZM	IST	13:00	14:00
25	ANK	IST	21:00	22:00	54	IZM	IST	15:00	16:00
26	ANK	IST	22:00	23:00	55	IZM	IST	17:00	18:00
27	IST	ANT	07:20	08:35	56	IZM	IST	19:20	20:20
28	IST	ANT	11:25	12:40	57	IZM	IST	22:00	23:00
29	IST	ANT	14:25	15:40	58	IZM	IST	01:45	02:45

Tabla 5.1.2. Horario de 58 vuelos. Fuente: «Robust Crew Pairing for Managing Extra Flights» [40].

ID	Origin	Destination	Departure	Arrival	ID	Origin	Destination	Departure	Arrival
1	IST	ANK	04:00	05:05	49	IST	IZM	16:00	17:05
2	IST	ANK	05:10	06:15	50	IZM	IST	18:05	19:10
3	ANK	IST	04:15	05:20	51	IST	IZM	17:00	18:05
4	ANK	IST	05:30	06:35	52	IZM	IST	19:05	20:10
5	IST	ANK	06:40	07:45	53	IST	IZM	21:45	22:50
6	ANK	IST	06:00	07:05	54	IZM	IST	23:50	00:55
7	ANK	IST	07:00	08:05	55	ANK	IZM	05:45	07:00
8	ANK	IST	07:30	08:35	56	IZM	ANK	08:00	09:15
9	IST	ANK	07:00	08:05	57	ANK	IZM	15:00	16:15
10	ANK	IST	08:00	09:05	58	IZM	ANK	17:15	18:30
11	IST	ANK	09:00	10:05	59	ANK	IZM	20:50	22:05
12	IST	ANK	10:00	11:05	60	IZM	ANK	23:10	00:25
13	ANK	IST	09:00	10:05	61	ANK	ANT	04:15	05:15
14	IST	ANK	11:00	12:05	62	ANT	ANK	06:15	07:15
15	ANK	IST	11:00	12:05	63	ANK	ANT	19:00	20:00
16	IST	ANK	13:00	14:05	64	ANT	ANK	21:00	22:00
17	ANK	IST	12:00	13:05	65	IST	ANT	06:25	07:40
18	IST	ANK	14:00	15:05	66	ANT	IST	08:40	09:55
19	ANK	IST	13:00	14:05	67	IST	ANT	09:30	10:45
20	ANK	IST	14:00	15:05	68	ANT	IST	11:45	13:00
21	ANK	IST	15:00	16:05	69	IST	ANT	12:45	14:00

22	IST	ANK	15:00	16:05	70	ANT	IST	15:00	16:15
23	ANK	IST	16:00	17:05	71	IST	ANT	15:30	16:45
24	IST	ANK	16:00	17:05	72	ANT	IST	17:55	19:10
25	IST	ANK	16:15	17:20	73	IST	ANT	17:00	18:15
26	ANK	IST	17:00	18:05	74	ANT	IST	19:15	20:30
27	IST	ANK	17:00	18:05	75	IST	ANT	18:30	19:45
28	IST	ANK	18:00	19:05	76	ANT	IST	20:45	22:00
29	IST	ANK	19:00	20:05	77	IST	ANT	21:55	23:10
30	ANK	IST	19:00	20:05	78	ANT	IST	00:15	01:30
31	ANK	IST	20:00	21:05	79	IST	ADA	06:20	07:50
32	IST	ANK	20:00	21:05	80	ADA	IST	08:50	10:20
33	IST	IZM	05:00	06:05	81	IST	ADA	15:00	16:30
34	IZM	IST	07:05	08:10	82	ADA	IST	17:30	19:00
35	IST	IZM	06:00	07:05	83	IST	ADA	17:20	18:50
36	IZM	IST	08:05	09:10	84	ADA	IST	19:55	21:35
37	IST	IZM	06:40	07:45	85	IST	ADA	12:15	13:45
38	IZM	IST	08:45	09:50	86	ADA	IST	14:45	16:25
39	IST	IZM	07:00	08:05	87	IST	ADA	14:00	15:30
40	IZM	IST	09:05	10:10	88	ADA	IST	16:30	18:00
41	IST	IZM	09:00	10:05	89	IST	ADA	19:30	21:00
42	IZM	IST	11:05	12:10	90	ADA	IST	22:00	23:30
43	IST	IZM	11:00	12:05	91	IST	ADA	09:15	10:45
44	IZM	IST	13:10	14:15	92	ADA	IST	11:45	13:15
45	IST	IZM	13:00	14:05	93	IST	ADA	21:35	23:05
46	IZM	IST	15:05	16:10	94	ADA	IST	01:30	02:00
47	IST	IZM	14:00	15:05	95	ANK	ADA	17:30	18:30
48	IZM	IST	16:10	17:15	96	ADA	ANK	19:30	20:30

Tabla 5.1.3. Horario de 96 vuelos. Fuente: «Robust Crew Pairing for Managing Extra Flights» [40].

Estos horarios fueron copiados a archivos de texto con extensión «.csv», con el primer renglón detallando el orden en que se encuentran los valores, y estos separados por «,».

Los valores de los parámetros y restricciones asignados por defecto, es decir, para la primera serie de pruebas, son los establecidos en la tabla 5.1.4:

Nombre del parámetro	Descripción	Valor asignado
$T_{emín}$	Tiempo mínimo entre la llegada de un vuelo y la salida del siguiente para una misma tripulación.	*0.5 h (30 min)*
$T_{emáx}$	Tiempo máximo entre la llegada de un vuelo y la salida del siguiente para una misma tripulación.	*4 h*
$T_{máxtrabajo}$	Duración máxima de la jornada de trabajo de una tripulación.	*12 h*
$T_{vmáx}$	Horas máximas de vuelo en una jornada.	*8 h*
costo_hora_vuelo	Costo de hora de vuelo por cada tripulación.	*1000 USD*
costo_hora_espera	Costo de hora de espera por cada tripulación.	*0.75(costo_hora_vuelo)*
penalización_por_varado	Penalización por cada tripulación que no termine en su base.	*7(costo_hora_vuelo)*
penalización_por_headhead	Penalización por cada «deadhead» en la solución.	*penalización_por_varado*
pen_por_vuelo_sin_cubrir	Penalización por cada vuelo que quede sin ser cubierto en la solución.	*1.7(penalización_por_varado)*
B1, B2	Aeropuertos base de la aerolínea.	*IST, ANK*

Tabla 5.1.4. Parámetros iniciales para la primera serie de pruebas. Fuente: elaboración propia.

5.2 Experimentos con el primer sistema inteligente

El primer algoritmo está programado para ejecutarse a manera de «script» desde la terminal de la computadora. Se le ingresan dos parámetros: el nombre del archivo de horario, y la semilla utilizada para generar los números aleatorios, esta segunda es totalmente opcional, pues si no se ingresa por el usuario, se utiliza una aleatoria; el hecho de poder ingresarla manualmente es por si se desea replicar algún resultado obtenido previamente. La sintaxis es la siguiente:

```
IA1_FC_con_GBJ.exe nombre_archivo_a_leer.csv (opcional)semilla
```

Una vez que ha terminado su ejecución, el programa mostrará en pantalla un recuento de la solución obtenida, donde aparecen los vuelos que cubrirá cada tripulación y los costos de cada una; las ID de los vuelos que no fueron cubiertos y el costo de tal penalización; el número de tripulaciones que no pudieron volver a base, el costo de esa penalización; y el costo total de la solución encontrada. La figura 5.2.1 es una captura de pantalla de ejemplo para 38 vuelos:

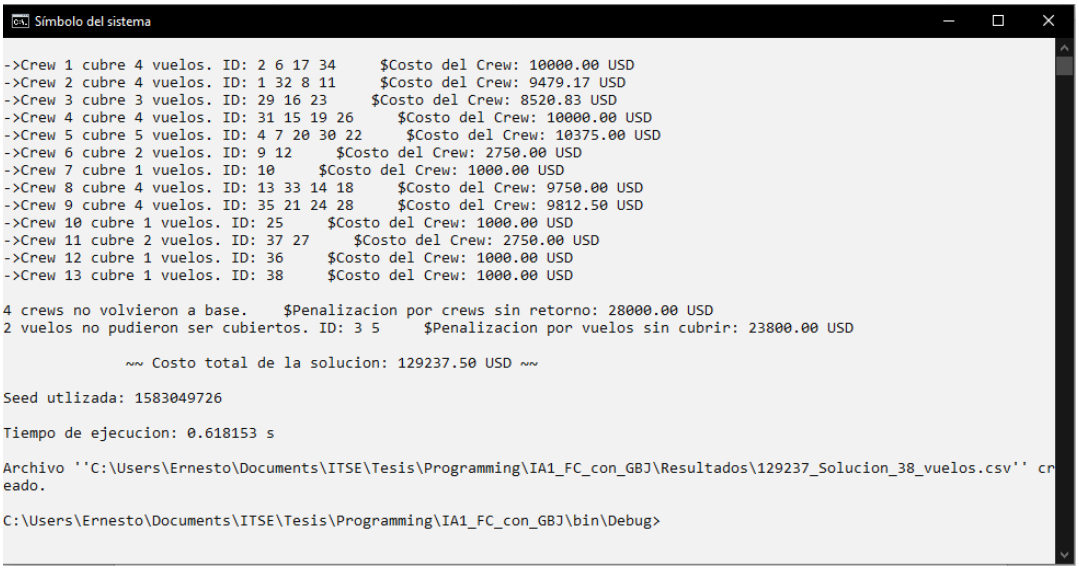


Figura 5.2.1. Ejemplo de ejecución del primer sistema inteligente. Fuente: elaboración propia.

Además de la visualización en pantalla, el sistema también crea un archivo de salida «.csv» que contiene la información de la solución encontrada. El nombre del archivo tiene la siguiente estructura: *CostoTotal*_Solucion_NombreArchivoLeido.csv; el costo de la solución se incluye en el nombre del archivo con el fin de que estos se ordenen automáticamente en el explorador de archivos.

De esta manera es que se realizan las ejecuciones necesarias para obtener 50 archivos de salida distintos por cada horario de vuelos, es decir, un total de 150 archivos de texto, de los cuales se obtendrán estadísticas y comparaciones en el capítulo dedicado a los resultados.

5.3 Experimentos con el segundo sistema inteligente

El segundo algoritmo, al igual que el primero, está programado para poderse ejecutar como un «script» desde la terminal. A este se le ingresan hasta 3 parámetros, los cuales son: el nombre del archivo del horario; el número máximo de iteraciones que, si no se ingresa, se toma el establecido dentro de los parámetros del programa; y la semilla utilizada para generar los números aleatorios, también opcional, pues si no se ingresa manualmente, se utiliza una aleatoria. La sintaxis es la siguiente:

```
IA2_SA.exe nombre_archivo_a_leer.csv (opcional)max_iteraciones (opcional)semilla
```

Una vez que se ha alcanzado el número máximo de iteraciones, el programa mostrará en pantalla la mejor solución obtenida, donde aparecerá la combinación de «pairings» que la conforma, los vuelos que cubre cada una y sus costos; los ID de los vuelos que no fueron cubiertos y el costo de tal penalización; número de «deadheadings», los ID de los vuelos que lo tienen, y el costo de esa penalización; así como el costo total de la solución encontrada. La figura 5.3.1 es una captura de pantalla que muestra un ejemplo de salida para la instancia de 38 vuelos:

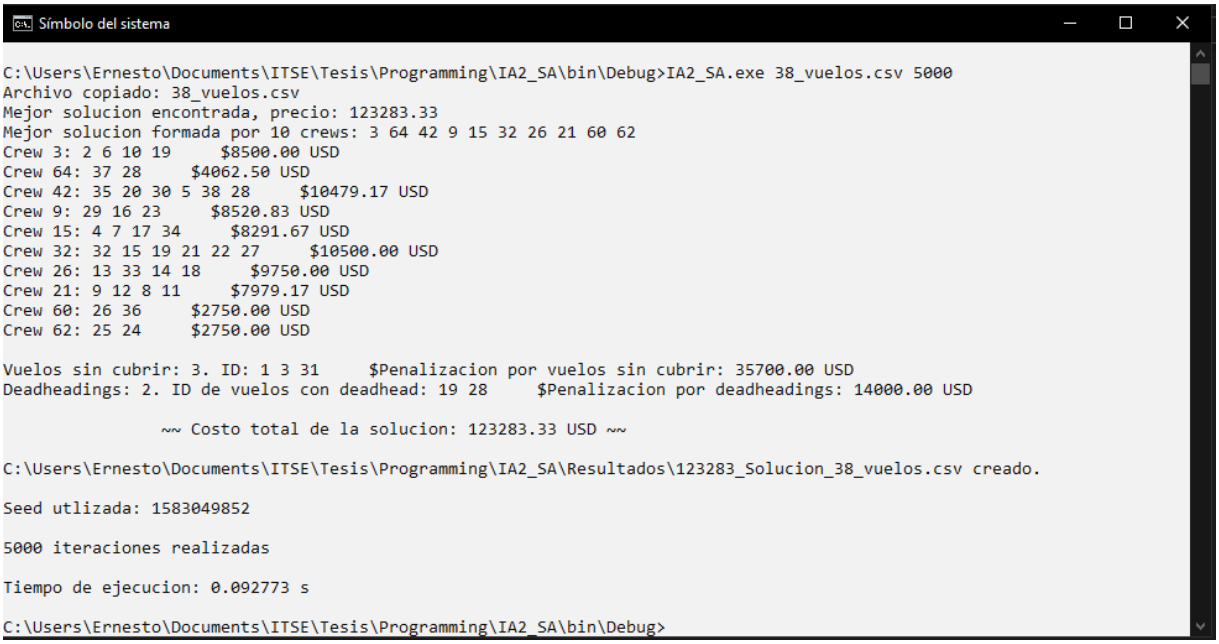


Figura 5.3.1. Ejemplo de ejecución del segundo sistema inteligente. Fuente: elaboración propia.

Este sistema también genera un archivo «.csv» de salida, que contiene la información de la mejor solución encontrada. El nombre del archivo, al igual que en el otro caso, se compone por: *CostoTotal*_Solucion_NombreArchivoLeido.csv.

En este caso, parte importante del funcionamiento del algoritmo radica en la cantidad de iteraciones, así que las pruebas también se dividirán en número de iteraciones. Se obtendrán 50 archivos de texto para 2500, 5000 y 10000 iteraciones por cada horario de vuelos, dando un total de 450 archivos de salida, los cuales serán analizados y comparados en la sección correspondiente. Además, se realizarán algunas pruebas extra, utilizando la semilla de los mejores resultados en cada máximo de iteraciones, con un número de iteraciones mayor, para de observar si el hecho de incrementar las iteraciones ayuda a mejorar el costo de la solución encontrada.

Finalmente, los valores de los parámetros propios del segundo sistema que fueron configurados como «por defecto», se encuentran en la tabla 5.3.1:

Nombre del parámetro	Descripción	Valor asignado
máx_núm_crews_por_vuelo	Número de «pairings» máximo que se pueden generar a partir de un vuelo inicial. Aumentarlo incrementa el número total de «crews» generado, por ende también el número de variables.	*4*
coeficiente_temperatura	El coeficiente que multiplica a la temperatura cada vez que se disminuye, por ende siempre debe ser menor a 1.	*0.9*
temperatura_inicial	Valor de la temperatura inicial, se multiplica por el *costo_hora_vuelo* para mantener consistencia con el resultado de la resta en Δf.	*100(costo_hora_vuelo)*

Tabla 5.3.1. Parámetros iniciales del segundo sistema. Fuente: elaboración propia.

5.4 Experimentos con variación de parámetros

Además de las pruebas hechas con los parámetros que se asignaron por defecto, también se harán pruebas mediante la variación de parámetros, ya sea modificando los valores de ciertas restricciones, o modificando parámetros propios del algoritmo, con el fin de observar cambios en el funcionamiento y en las soluciones encontradas. Cada sistema tendrá su propia serie de pruebas según sean sus parámetros modificables, por lo que las pruebas están enfocadas principalmente a hacer comparativas individuales de cada algoritmo, en vista de que varias de ellas no son directamente equiparables entre ambos.

5.4.1 Variación de parámetros en el primer sistema

Este sistema, debido a que utiliza técnicas de tipo completa, no posee parámetros propios de su funcionamiento que sean modificables para repercutir directamente en la calidad de la solución encontrada ni en su funcionamiento, pues recuérdese que las técnicas completas navegan por todo el espacio de búsqueda. Por ello, la variación se puede realizar mediante los valores de los demás parámetros.

No todos ellos afectan directamente al funcionamiento del algoritmo. Por ejemplo, las penalizaciones por vuelos sin cubrir y por tripulaciones sin retorno no modifican de ninguna

manera la búsqueda de la solución, solamente se utilizan para saber el costo total una vez que esta ha sido completada, así que no tendría ningún caso modificarlas para este sistema. Por otro lado, las restricciones temporales sí repercuten en la manera en que funciona el algoritmo, así como en las soluciones que este encuentra, por lo que serán estas las que se varíen de tal manera que estén «más apretadas» para dificultarle el trabajo al algoritmo, y «más libres» para facilitárselo. Se proponen entonces dos etapas de variación, la primera con restricciones temporales más justas, y la segunda con restricciones temporales más libres.

Aprieto[4] de restricciones temporales

Se decidió, para la primera subetapa, modificar los parámetros de las restricciones temporales de tiempo mínimo y máximo de espera, de la manera como se muestra en la tabla 5.4.1:

Nombre del parámetro	Descripción	Valor asignado
$T_{emín}$	Tiempo mínimo entre la llegada de un vuelo y la salida del siguiente para una misma tripulación.	*1 h*
$T_{emáx}$	Tiempo máximo entre la llegada de un vuelo y la salida del siguiente para una misma tripulación.	*3 h*

Tabla 5.4.1. Primera modificación de parámetros en el primer sistema. Fuente: elaboración propia.

Para la segunda subetapa, se modificará únicamente el largo de la jornada de trabajo ($T_{máxtrabajo}$), reduciéndolo de 12 a 10 h, para observar si el algoritmo puede entregar soluciones de calidad con una jornada de trabajo más reducida.

[4] El «aprieto» de las restricciones consiste en aumentar el tiempo mínimo de espera que una tripulación necesita entre un vuelo y otro, así como disminuir el tiempo máximo que pueden pasar esperando por su siguiente vuelo, para forzarlas a estar trabajando el mayor tiempo posible.

Ensanche[5] de restricciones temporales

En esta etapa se modificará únicamente el largo de la jornada de trabajo ($T_{máxtrabajo}$), extendiéndolo de 12 a 24 h, con el fin de que el único valor que restrinja el número de vuelos que una tripulación cubre, sea el número máximo de horas de vuelo por jornada. Recuérdese que los demás valores son regresados a su valor por defecto.[6]

5.4.2 Variación de parámetros en el segundo sistema

El segundo programa utiliza una técnica incompleta, la cual posee parámetros propios de su funcionamiento que afectan a la manera en que esta encuentra soluciones. El aumento y disminución del *máx_núm_crews_por_vuelo* aumenta y disminuye respectivamente el tamaño del espacio de búsqueda, y el aumento del *coeficiente_temperatura* hace que se incremente la exploración que el sistema realiza en un inicio.

Los valores de las restricciones no repercuten de forma directa al algoritmo en sí, sino que afectan a la etapa de generación de «pairings», por lo que en este caso no se les modificarán sus valores. Sin embargo, caso contrario al primer sistema, los valores de las penalizaciones por vuelos sin cubrir y por «deadheadings» afectan directamente al algoritmo, pues tienen impacto en la manera en que se comparan las soluciones futuras y presentes dentro de SA.

Se proponen entonces 3 etapas de variación, las cuales variarán los parámetros y penalizaciones con relevancia anteriormente mencionados. Todas las pruebas se realizarán con 10,000 iteraciones.

Modificación del «máx_núm_crews_por_vuelo»

Se aumentará y disminuirá este valor con el fin de observar el comportamiento en la calidad de las soluciones encontradas por el algoritmo, para las 3 instancias. Los valores asignados se encuentran en la tabla 5.4.2:

[5] El «ensanche» en las restricciones consiste en aumentar la jornada de trabajo, para que el algoritmo tenga mayor libertad en cuanto a la elección de vuelos factibles posibles.

[6] Nota: si no se muestran los valores de ciertos parámetros en una etapa de modificación, es porque estos fueron configurados/reconfigurados a su valor por defecto.

Subetapa	Nombre del parámetro	Valor asignado
Aumento	*máx_núm_crews_por_vuelo*	*8*
Disminución	*máx_núm_crews_por_vuelo*	*2*

Tabla 5.4.2. Variación del número máximo de tripulaciones por vuelo inicial, para el segundo sistema. Fuente: elaboración propia.

Modificación del «coeficiente_temperatura»

Se incrementará el valor del *coeficiente_temperatura* a 0.965, para así observar si el hecho de aumentar la exploración inicial conlleva al algoritmo a encontrar mejores soluciones. Se reutilizarán algunas semillas de los experimentos por defecto, para que la generación de tripulaciones sea la misma y sólo varíe la etapa de SA. No se considera relevante el hecho de disminuir el coeficiente para decrementar la exploración, pues se pierde parte fundamental de la técnica utilizada.

Modificación de penalizaciones

Debido a que las penalizaciones por vuelos no cubiertos y por «deadheading» repercuten directamente en las decisiones que toma el algoritmo, estas se variarán para observar qué efectos producen en las soluciones halladas. Los valores asignados se muestran en la tabla 5.4.3:

Subetapa	Nombre del parámetro	Valor asignado
1	*pen_por_deadheading*	*2(hora_de_vuelo)*
	pen_por_vuelo_sin_cubrir	*12(hora_de_vuelo)*
2	*pen_por_deadheading*	*12(costo_hora_vuelo)*
	pen_por_vuelo_sin_cubrir	*12(costo_hora_vuelo)*

Tabla 5.4.3. Variación de valores de penalizaciones, para el segundo sistema. Fuente: elaboración propia.

El hecho de no colocar la penalización por «deadheading» con un costo mayor al de vuelos cubiertos, es porque se considera que nunca será más costoso que una tripulación viaje como pasajera a que un vuelo quede sin cubrir; por eso sólo se incrementa, por mucho, al mismo costo.

6. Resultados

En el presente capítulo se analizarán y comentarán los resultados obtenidos en la etapa de experimentación, tanto para la primera serie de pruebas con los parámetros por defecto, como para las pruebas realizadas en la segunda etapa de experimentación mediante la variación de parámetros. Además de los resultados obtenidos de forma individual para cada algoritmo, también se harán comparativas de los resultados obtenidos entre ambos.

6.1 Análisis de resultados con parámetros iniciales

La primera serie de pruebas fue realizada con los 3 horarios de prueba en ambos sistemas inteligentes, configurando los valores de las restricciones y de los parámetros del funcionamiento a los bautizados «por defecto». Del primer sistema se obtuvieron 150 archivos de salida, y del segundo, 450. Los resultados obtenidos se encuentran detallados en las dos secciones siguientes.

6.1.1 Resultados obtenidos con el primer sistema

Debido a que la primera IA utiliza técnicas completas, no hay ningún parámetro propio del algoritmo que se tuviese que configurar. Se dejó que este entregara una solución por ejecución, realizando el número de ejecuciones necesario para obtener 50 archivos de salida diferentes en cada horario de prueba.

Horario de 38 vuelos

La primera observación que hay que comentar, es que fue con el horario de 38 vuelos cuando más número de ejecuciones se tuvo que realizar para obtener los 50 archivos de salida. En otras palabras, el espacio de búsqueda del horario de 38 vuelos no es tan grande como para poder entregar 50 soluciones distintas de forma fácil, entre más soluciones se tienen creadas, más complicado fue encontrar diferentes. Se realizaron 163 ejecuciones para obtener las 50 soluciones distintas.

- **Resultados promedio:**

La tabla 6.1.1 posee los resultados promedio obtenidos por el algoritmo, los cuales muestran los detalles de las soluciones y del tiempo de ejecución para la instancia de 38 vuelos: La columna de ID de vuelos más conflictivos, acomodada en orden descendiente, muestra las ID de los vuelos que aparecían más frecuentemente en las ID de vuelos sin cubrir; el vuelo 3 estuvo presente en la mayoría.

Número de «crews» generados	Número de «crews» que no volvieron a base	Número de vuelos no cubiertos	ID de vuelos más conflictivos	Costo de solución [USD]	Tiempo de ejecución [ms]
12 – 13	3 – 4	2	3, 5, 6	123,356.00	81.37336

Tabla 6.1.1. Resultados promedio del primer programa, para la instancia de 38 vuelos con parámetros iniciales. Fuente: elaboración propia.

- **Mejor solución encontrada:**

La mejor solución encontrada, de las 163 ejecuciones realizadas para el horario de 38 vuelos, tuvo un costo de $96,170.83, un tiempo de ejecución de 16.355 ms, y está formada por las tripulaciones de la tabla 6.1.2:

Crew 1: 2, 6, 10, 16, 20, 35	Costo: $10500.00
Crew 2: 1, 32, 8, 11	Costo: $9479.17
Crew 3: 29, 12, 15, 19, 23	Costo: $9020.83
Crew 4: 31, 13, 33, 26	Costo: $10000.00
Crew 5: 4, 7, 17, 30, 22	Costo: $10375.00
Crew 6: 9	Costo: $1000.00
Crew 7: 35, 21, 5, 38, 28	Costo: $10145.83
Crew 8: 14, 18	Costo: $3250.00
Crew 9: 25, 24	Costo: $2750.00
Crew 10: 37, 27	Costo: $2750.00
Crew 11: 36	Costo: $1000.00
2 crews sin retorno: 6, 11	Penalización: $14000.00
1 vuelo sin cubir. ID: 3	Penalización: $11900.00
Costo total	$96,170.83

Tabla 6.1.2. Mejor solución encontrada por el primer algoritmo, para el horario de 38 vuelos con parámetros iniciales. Fuente: elaboración propia.

Como puede observarse, los resultados promedio muestran que la mayoría de soluciones encontradas tienen unos cuantos vuelos sin cubrir, así como unas cuantas tripulaciones que tuvieron que pasar el resto del día en una ciudad distinta a la de su base. La mejor solución

hallada es 27,000 dólares más barata en comparación con el promedio, pero aun siendo la mejor, viola las dos restricciones blandas, por lo que sufre ambas penalizaciones. Si se desea replicar este resultado, se obtuvo con la semilla 1572825131. De las 50 soluciones distintas encontradas, se concluye que, al parecer, con los valores de las restricciones configurados por defecto, no existen soluciones para este horario que satisfagan al 100 % todas las restricciones, o que al menos estas no son fáciles de hallar, y este sistema inteligente no fue capaz de hacerlo.

Horario de 58 vuelos

Para este caso, encontrar las 50 soluciones diferentes no tuvo tanta dificultad como en el primer horario. Se realizaron 78 ejecuciones para obtener los 50 archivos de salida, y los resultados fueron los siguientes:

- **Resultados promedio:**

La tabla 6.1.3 contiene los valores promedio de los resultados en los 50 archivos, para el horario de 58 vuelos:

Número de «crews» generados	Número de «crews» que no volvieron a base	Número de vuelos no cubiertos	ID de vuelos más conflictivos	Costo de solución [USD]	Tiempo de ejecución [ms]
18	3	1	53, 50, 52	156,210.09	84.85064

Tabla 6.1.3. Resultados promedio del primer programa, para la instancia de 58 vuelos con parámetros iniciales. Fuente: elaboración propia.

- **Mejor solución encontrada:**

La mejor solución encontrada, de las 78 ejecuciones realizadas para la instancia de 58 vuelos, tuvo un costo de $118,958.34, un tiempo de ejecución de 25.9763 ms, y está conformada por las tripulaciones de la tabla 6.1.4:

Crew 1: 1, 4, 46, 55	Costo: $9541.67
Crew 2: 7, 19, 33, 35	Costo: $9270.83
Crew 3: 41, 51, 29, 32	Costo: $9375.00
Crew 4: 27, 30, 10, 21	Costo: $9125.00
Crew 5: 42, 52, 45, 54, 12, 22	Costo: $10500.00

Crew 6: 43, 50, 3, 6	Costo: $8729.17
Crew 7: 8, 18, 11, 24	Costo: $10000.00
Crew 8: 17, 2, 5, 13	Costo: $8416.67
Crew 9: 9, 20, 37, 39	Costo: $9416.67
Crew 10: 44, 53, 47, 56	Costo: $8000.00
Crew 11: 28, 31, 14, 23	Costo: $7937.50
Crew 12: 15, 25, 49, 58	Costo: $6812.50
Crew 13: 34, 36	Costo: $3083.33
Crew 14: 38, 40	Costo: $3250.00
Crew 15: 16, 26	Costo: $2750.00
Crew 16: 48, 57	Costo: $2750.00
Todos los crews volvieron a su base	Penalización: $0.00
Todos los vuelos cubiertos	Penalización: $0.00
Costo total	$118,958.34

Tabla 6.1.4. Mejor solución encontrada por el primer algoritmo, para el horario de 58 vuelos con parámetros iniciales. Fuente: elaboración propia.

Como se observa, los valores promedio de las soluciones encontradas también poseen tripulaciones que no volvieron a su base, así como algunos vuelos sin cubrir. No obstante, la mejor solución encontrada satisface al 100 % todas ellas, por lo que se concluye que, para los valores de las restricciones por defecto y el horario de 58 vuelos, sí existen soluciones que pueden satisfacerlas por completo; de estas se hallaron dos, la mostrada fue la más barata de ambas. Si se desea replicar el resultado, este fue obtenido con la semilla 1570002534.

Horario de 96 vuelos

Este horario fue el que entregó más variedad de resultados. Se ejecutó 51 veces y exportó los 50 archivos de salida para su respectivo análisis. Los resultados obtenidos fueron los siguientes:

- **Resultados promedio:**

La tabla 6.1.5 muestra los valores promedio obtenidos en las ejecuciones realizadas, para la instancia de 96 vuelos:

Número de «crews» generados	Número de «crews» que no volvieron a base	Número de vuelos no cubiertos	ID de vuelos más conflictivos	Costo de solución [USD]	Tiempo de ejecución [s]
30	6	4	34, 72, 62	277,593.18	1.1918981

Tabla 6.1.5. Resultados promedio del primer programa, para la instancia de 96 vuelos con parámetros iniciales. Fuente: elaboración propia.

- **Mejor solución encontrada:**

La mejor solución encontrada de las 51 ejecuciones realizadas tuvo un costo de $221,112.52, un tiempo de ejecución de 0.9824 s, y quedó conformada por las tripulaciones de la tabla 6.1.6:

Crew 1: 1, 8, 14, 6, 41, 42	Costo: $7750.00
Crew 2: 3, 91, 92, 22	Costo: $10166.67
Crew 3: 61, 62, 10, 16	Costo: $8416.67
Crew 4: 33, 56, 19	Costo: $7666.67
Crew 5: 2, 7, 11, 17, 18, 23	Costo: $10562.50
Crew 6: 4, 67, 68, 25	Costo: $10041.67
Crew 7: 55, 36, 43, 44, 24	Costo: $9895.83
Crew 8: 35, 40, 47, 48	Costo: $9520.83
Crew 9: 79, 80, 69, 70	Costo: $8812.50
Crew 10: 65, 66, 45, 46	Costo: $8479.17
Crew 11: 5, 13, 87, 88	Costo: $9791.67
Crew 12: 37, 38, 85, 86	Costo: $8645.83
Crew 13: 9, 15, 81, 82	Costo: $10291.67
Crew 14: 39	Costo: $1083.33
Crew 15: 12, 20, 83, 84	Costo: $10020.83
Crew 16: 21, 73, 32	Costo: $5416.67
Crew 17: 57, 58, 59, 60	Costo: $8312.50
Crew 18: 71, 72, 77, 78	Costo: $8750.00
Crew 19: 49, 50, 53, 54	Costo: $7770.83
Crew 20: 26, 29	Costo: $2854.17
Crew 21: 27, 30, 93, 94	Costo: $7791.67

Crew 22: 51, 52	Costo: $2916.67
Crew 23: 95, 96	Costo: $2750.00
Crew 24: 28, 31	Costo: $2854.17
Crew 25: 75, 76	Costo: $3250.00
Crew 26: 63, 64	Costo: $2750.00
Crew 27: 89, 90	Costo: $3750.00
1 crew sin retorno: 14	Penalización: $7000.00
2 vuelos sin cubrir. ID: 34, 74	Penalización: $23800.00
Costo total	$221,112.5

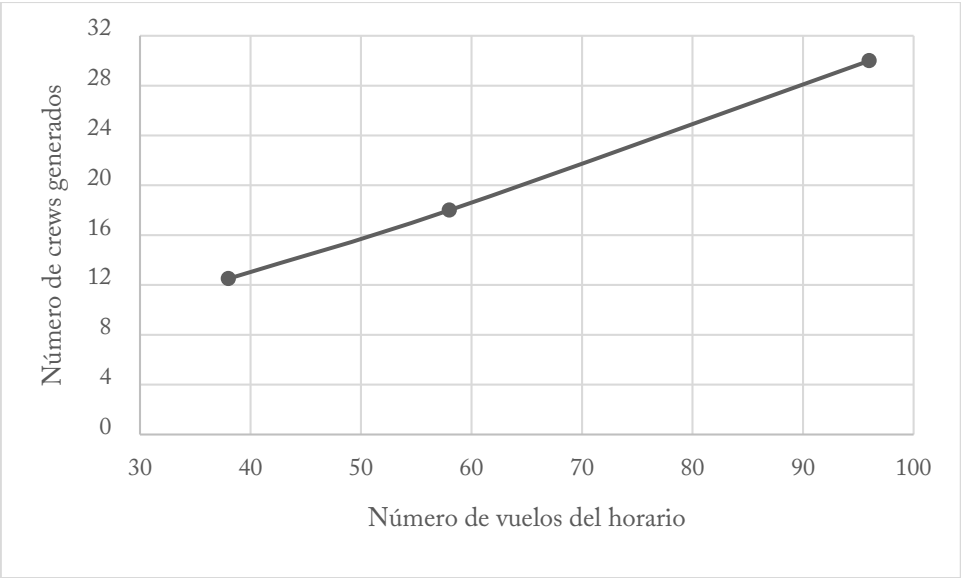
Tabla 6.1.6. Mejor solución encontrada por el primer algoritmo, para el horario de 96 vuelos con parámetros iniciales. Fuente: elaboración propia.

De los resultados obtenidos es de notarse que, al igual que en los casos anteriores, los valores promedio indican que las soluciones encontradas poseen algunos vuelos sin cubrir, así como tripulaciones que no pudieron regresar a sus respectivas bases. La mejor solución encontrada, a pesar de ser 56,000 dólares más barata que el promedio, sufre las penalizaciones por el incumplimiento de ambas restricciones. No se considera, sin embargo, que no existan soluciones que satisfagan al 100 % las restricciones, el no haber encontrado una solución de este tipo seguramente se deba al tamaño del espacio de búsqueda; es probable que, si se siguieran buscando soluciones nuevas, en algún momento el sistema encontraría una de este tipo, pero no se continuó con la búsqueda con el fin de mantener uniformidad en la cantidad de archivos para todos los horarios.

Efectos del incremento en el número de vuelos

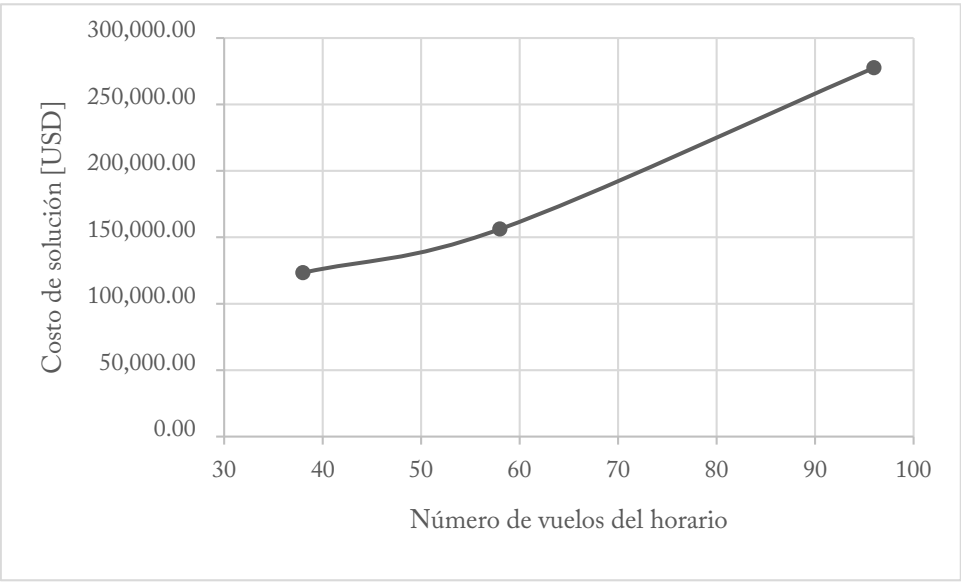
Con la información promedio obtenida en los apartados correspondientes a cada horario, se pueden realizar comparativas para observar el comportamiento del sistema y de las soluciones, conforme se le van añadiendo vuelos al horario en cuestión.

Si se grafica el número de tripulaciones generadas contra número de vuelos, se observa un comportamiento bastante lineal. La gráfica indica que el número de tripulaciones necesarias crece proporcionalmente al número de vuelos que se tengan que cubrir. Véase la gráfica 6.1.1:



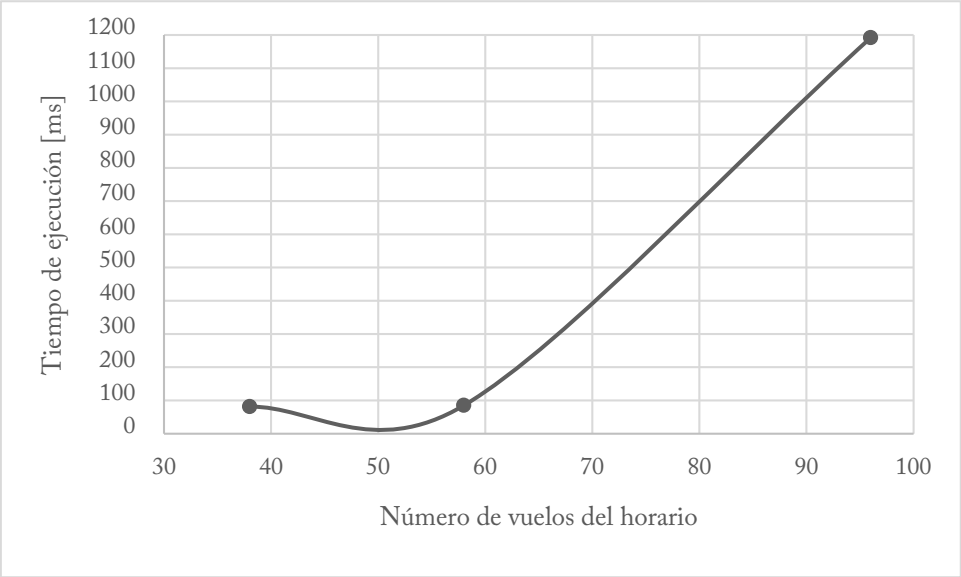
Gráfica 6.1.1. Número de «crews» promedio contra número de vuelos, para el primer sistema inteligente con parámetros iniciales. Fuente: elaboración propia.

Lo mismo ocurre si se grafica el costo promedio de la solución obtenida contra el número de vuelos, se observa una tendencia bastante lineal al incremento del costo conforme al aumento en los vuelos del horario. Véase la gráfica 6.1.2:



Gráfica 6.1.2. Costo promedio de solución contra número de vuelos, para el primer sistema inteligente con parámetros iniciales. Fuente: elaboración propia.

El resultado más interesante es la comparativa del tiempo de ejecución contra el número de vuelos. En este caso no se observa un comportamiento lineal, sino uno de tipo exponencial, pues el tiempo de ejecución promedio para la instancia de 96 vuelos se aleja por mucho del tiempo de los otros dos horarios. Véase la gráfica 6.1.3:



Gráfica 6.1.3. Tiempo de ejecución promedio en milisegundos contra número de vuelos, para el primer sistema inteligente con parámetros iniciales. Fuente: elaboración propia.

Con la gráfica 6.1.3 queda comprobado experimentalmente el fenómeno conocido como «explosión combinatoria»; pues se observa cómo, a medida que aumentan los vuelos en el horario del problema, el tiempo necesario para resolverlo se incrementa de forma exponencial, debido al crecimiento no lineal en la cantidad de posibles combinaciones.

6.1.2 Resultados obtenidos con el segundo sistema

El segundo sistema utiliza una técnica incompleta, la cual posee parámetros propios de su funcionamiento que tuvieron que ser configurados por defecto y se muestran anteriormente en la tabla 5.3.1. Adicionalmente, se decidió no mantener constante el número de iteraciones, y se hicieron experimentos con los parámetros iniciales, pero con distintos valores del máximo de iteraciones, así que los resultados obtenidos están divididos de esta manera.

Horario de 38 vuelos

Debido a la aceptación de «deadheadings» y la variada generación de tripulaciones, el algoritmo tuvo facilidad en entregar los 50 archivos de salida con soluciones distintas, teniendo que realizar, en el peor caso, 51 ejecuciones para obtenerlos. Los resultados fueron los siguientes:

- **Resultados promedio:**

La tabla 6.1.7 contiene los resultados promedio obtenidos para el horario de 38 vuelos, divididos por número de iteraciones:

Número de iteraciones	Número de «crews» generados	Número de «deadheads»	Número de vuelos no cubiertos	Costo de solución [USD]	Tiempo de ejecución [ms]
2500	11	4 – 5	3 – 4	151,826.91	41.8637
5000	11	4	3	145,541.25	55.90448
10000	11	3 – 4	3	136,922.00	95.9078

Tabla 6.1.7. Resultados promedio del segundo sistema, para la instancia de 38 vuelos con parámetros iniciales. Fuente: elaboración propia.

- **Mejores soluciones encontradas:**

La mejor solución encontrada, con 2500 iteraciones, tuvo un costo de $91,833.33 y un tiempo de ejecución de 47.355 ms. Se puede replicar con la semilla 1574043296. Quedó conformada por las tripulaciones de la tabla 6.1.8:

38 vuelos / 2500 iteraciones	
Crew 1: 26, 36	Costo: $2750.00
Crew 2: 31, 10, 16, 23	Costo: $8500.00
Crew 3: 1, 32, 8, 11	Costo: $9479.17
Crew 4: 29, 3, 33, 23	Costo: $8833.33
Crew 5: 4, 7, 17, 34	Costo: $8291.67
Crew 6: 9, 12, 20, 30, 22	Costo: $9833.33
Crew 7: 13, 35, 21, 5, 38	Costo: $10333.33

Crew 8: 37, 27	Costo: $2750.00
Crew 9: 14, 18	Costo: $3250.00
Crew 10: 2, 6, 15, 19	Costo: $8500.00
Crew 11: 25, 24	Costo: $2750.00
Crew 12: 38, 28	Costo: $2562.50
2 «deadheads». ID: 23, 38	Penalización: $14000.00
Todos los vuelos cubiertos	Penalización: $0.00
Costo total	$91,833.33

Tabla 6.1.8. Mejor solución encontrada por el segundo algoritmo en 2500 iteraciones, para el horario de 38 vuelos con parámetros iniciales. Fuente: elaboración propia.

La mejor solución encontrada, con 5000 iteraciones, tuvo un tiempo de ejecución de 75.433 ms. Se puede replicar con la semilla 1574044394. Conformada por las tripulaciones de la tabla 6.1.9:

38 vuelos / 5000 iteraciones	
Crew 1: 2, 6, 15, 19	Costo: $8500.00
Crew 2: 37, 27	Costo: $2750.00
Crew 3: 1, 32, 17, 34	Costo: $9250.00
Crew 4: 29, 3, 33, 23	Costo: $8833.33
Crew 5: 31, 13, 35, 23	Costo: $8500.00
Crew 6: 4, 7, 8, 11	Costo: $8520.83
Crew 7: 9, 12, 20, 30, 22	Costo: $9833.33
Crew 8: 10, 16, 21, 5, 38	Costo: $10333.33
Crew 9: 33, 14, 18, 28	Costo: $9187.50
Crew 10: 26, 36	Costo: $2750.00
Crew 11: 25, 24	Costo: $2750.00
2 «deadheads». ID: 23, 33	Penalización: $14000.00
Todos los vuelos cubiertos	Penalización: $0.00
Costo total	$95,208.33

Tabla 6.1.9. Mejor solución encontrada por el segundo algoritmo en 5000 iteraciones, para el horario de 38 vuelos con parámetros iniciales. Fuente: elaboración propia.

La mejor solución encontrada, con 10000 iteraciones, tuvo un costo de $91,833.33 y un tiempo de ejecución de 134.768 ms. Se puede replicar con la semilla 1574045389. Quedó conformada por las tripulaciones de la tabla 6.1.10:

38 vuelos / 10000 iteraciones	
Crew 1: 2, 6, 15, 19	Costo: $8500.00
Crew 2: 1, 32, 17, 34	Costo: $9250.00
Crew 3: 29, 3, 35, 23	Costo: $8833.33
Crew 4: 31, 10, 16, 20, 34, 26	Costo: $10500.00
Crew 5: 4, 7, 8, 11	Costo: $8520.83
Crew 6: 9, 12, 20, 30, 22	Costo: $9833.33
Crew 7: 37, 27	Costo: $2750.00
Crew 8: 13, 33, 21, 5, 38	Costo: $10333.33
Crew 9: 14, 18	Costo: $3250.00
Crew 10: 25, 24	Costo: $2750.00
Crew 11: 36, 28	Costo: $3312.50
2 «deadeads». ID: 20, 34	Penalización: $14000.00
Todos los vuelos cubiertos	Penalización: $0.00
Costo total	$91,833.33

Tabla 6.1.10. Mejor solución encontrada por el segundo algoritmo en 10000 iteraciones, para el horario de 38 vuelos con parámetros iniciales. Fuente: elaboración propia.

Como puede observarse, los valores promedio para la instancia de 38 vuelos incluyen, para las 3 iteraciones, algunos vuelos sin cubrirse, así como la presencia de algunos vuelos «deadhead». Las mejores soluciones cubrieron todos los vuelos a cambio de un par de «deadheads». En cuanto a la mejor semilla de 2500 iteraciones, se ejecutó con 5000 iteraciones y mejoró su costo en 1000 dólares, pero no fue la mejor encontrada. Las mejores semillas de 2500 y 5000 iteraciones fueron pasadas a ejecutarse con 10000, sin que el algoritmo fuera capaz de mejorarlas. También es de notarse que la mejor solución de las ejecuciones con 5000 iteraciones no fue tan buena como la obtenida con 2500 y 10000, las cuales, a pesar de tener tripulaciones distintas, tuvieron el mismo costo.

Horario de 58 vuelos

Al igual que en el caso anterior, la obtención de los 50 archivos de salida fue de manera fácil. Se realizaron exactamente 50 ejecuciones para cada número de iteraciones, todas con un costo diferente. Los resultados fueron los siguientes:

- **Resultados promedio:**

La tabla 6.1.11 contiene los resultados promedio obtenidos para el horario de 58 vuelos, los cuales están divididos por número de iteraciones:

Número de iteraciones	Número de «crews» generados	Número de «deadheads»	Número de vuelos no cubiertos	Costo de solución [USD]	Tiempo de ejecución [ms]
2500	16 – 17	11	4	262,015.50	54.24616
5000	16 – 17	10	3 – 4	247,195.58	87.23344
10000	16	9	3	237,169.08	150.93932

Tabla 6.1.11. Resultados promedio del segundo sistema, para la instancia de 58 vuelos con parámetros iniciales. Fuente: elaboración propia.

- **Mejores soluciones encontradas:**

La mejor solución encontrada, en la serie de ejecuciones con 2500 iteraciones, tuvo un costo de $218,500, un tiempo de ejecución de 49.8 ms, semilla 1573192996, y quedó conformada por las tripulaciones de la tabla 6.1.12:

58 vuelos / 2500 iteraciones	
Crew 1: 11, 4, 11, 21	Costo: $9541.67
Crew 2: 7, 19, 33, 35	Costo: $9270.83
Crew 3: 41, 50, 29, 32	Costo: $9375.00
Crew 4: 10, 20, 16, 26	Costo: $8500.00
Crew 5: 27, 30, 3, 6	Costo: $9729.17
Crew 6: 42, 51, 46, 55	Costo: $8500.00

Crew 7: 43, 52, 45, 54, 12, 22	Costo: $10125.00
Crew 8: 8, 18, 46, 56	Costo: $9500.00
Crew 9: 17, 2, 5, 15	Costo: $9541.67
Crew 10: 44, 53, 14, 23	Costo: $8125.00
Crew 11: 28, 31, 13, 25	Costo: $9062.50
Crew 12: 45, 55, 34, 36	Costo: $8458.33
Crew 13: 37, 40, 49, 58	Costo: $8208.33
Crew 14: 48, 57, 49, 58	Costo: $6062.50
5 «deadheads». ID: 45, 46, 49, 55, 58	Penalización: $35000.00
5 vuelos sin cubrir. ID: 9, 24, 38, 39, 47	Penalización: $59500.00
Costo total	$218,500.00

Tabla 6.1.12. Mejor solución encontrada por el segundo algoritmo en 2500 iteraciones, para el horario de 58 vuelos con parámetros iniciales. Fuente: elaboración propia.

La mejor solución encontrada, en la serie de ejecuciones con 5000 iteraciones, tuvo un costo de $208,995.83, un tiempo de ejecución de 89.85 ms, semilla 1573193208, y quedó conformada por las tripulaciones de la tabla 6.1.13:

58 vuelos / 5000 iteraciones	
Crew 1: 1, 4, 46, 55	Costo: $9541.67
Crew 2: 8, 18, 45, 54, 12, 22	Costo: $9750.00
Crew 3: 44, 53, 37, 39	Costo: $8666.67
Crew 4: 45, 54, 34, 36	Costo: $8458.33
Crew 5: 27, 30, 29, 32	Costo: $9250.00
Crew 6: 42, 52, 3, 6	Costo: $9104.17
Crew 7: 17, 2, 5, 13	Costo: $8416.67
Crew 8: 9, 19, 11, 21, 15, 25	Costo: $10500.00
Crew 9: 28, 31, 47, 56	Costo: $7812.50
Crew 10: 10, 20, 48, 57	Costo: $8500.00
Crew 11: 41, 50, 33, 35	Costo: $9270.83
Crew 12: 38, 40, 49, 58	Costo: $6500.00
Crew 13: 14, 23, 49, 58	Costo: $$7562.50

Crew 14: 16, 26, 49, 58	Costo: $6062.50
6 «deadheads». ID: 45, 49, 54, 58	Penalización: $42000.00
4 vuelos sin cubrir. ID: 7, 24, 43, 51	Penalización: $47600.00
Costo total	$208,995.83

Tabla 6.1.13. Mejor solución encontrada por el segundo algoritmo en 5000 iteraciones, para el horario de 58 vuelos con parámetros iniciales. Fuente: elaboración propia.

Y finalmente, la mejor solución encontrada, en la serie de ejecuciones con 10000 iteraciones, tuvo un costo de $199,525, un tiempo de ejecución de 154.78 ms, semilla 1573193468, y quedó conformada por las tripulaciones de la tabla 6.1.14:

58 vuelos / 10000 iteraciones	
Crew 1: 1, 4, 29, 32	Costo: $9666.67
Crew 2: 7, 19, 46, 55	Costo: $9250.00
Crew 3: 41, 51, 33, 35	Costo: $9270.83
Crew 4: 27, 30, 3, 6	Costo: $9729.17
Crew 5: 42, 52, 3, 6	Costo: $9104.17
Crew 6: 8, 18, 47, 56	Costo: $9500.00
Crew 7: 17, 2, 5, 13	Costo: $8416.67
Crew 8: 9, 20, 37, 39	Costo: $9416.67
Crew 9: 44, 53, 12, 24	Costo: $8500.00
Crew 10: 38, 40, 49, 58	Costo: $6500.00
Crew 11: 28, 31, 14, 23	Costo: $7937.50
Crew 12: 10, 21, 15, 26	Costo: $8500.00
Crew 13: 10, 21, 34, 36	Costo: $8458.33
Crew 14: 45, 54, 48, 57	Costo: $8500.00
Crew 15: 43, 50, 11, 22	Costo: $9625.00
Crew 16: 21, 16	Costo: $3500.00
Crew 17: 49, 58	Costo: $2750.00
7 «deadheads». ID: 3, 6, 10, 21, 49, 58	Penalización: $49000.00
1 vuelo sin cubrir. ID: 25	Penalización: $11900.00

Costo total	$199,525.00

Tabla 6.1.14. Mejor solución encontrada por el segundo algoritmo en 10000 iteraciones, para el horario de 58 vuelos con parámetros iniciales. Fuente: elaboración propia.

De los resultados obtenidos puede notarse que, también para este caso, los valores promedio entregados por el sistema incluyen soluciones que no cubren todos los vuelos, así como la presencia de «deadheadings». Las mejores soluciones, a pesar de ser aproximadamente $40,000 más baratas que el promedio, sufren penalizaciones por vuelos no cubiertos y por vuelos «deadhead». Las mejores semillas de cada máximo de iteraciones pasaron a probarse con uno mayor, con resultados que no lograron sustituir a ninguno de los mejores respectivos.

Horario de 96 vuelos

La gran cantidad de vuelos de este horario repercute en la variedad de soluciones. Se realizaron exactamente 50 ejecuciones para cada número de iteraciones. Los resultados obtenidos fueron los siguientes:

- **Resultados promedio:**

La tabla 6.1.15 contiene los resultados obtenidos para la instancia de 96 vuelos, con parámetros iniciales y divididos por número de iteraciones:

Número de iteraciones	Número de «crews» generados	Número de «deadheads»	Número de vuelos no cubiertos	Costo de solución [USD]	Tiempo de ejecución [ms]
2500	25	25 - 26	20	626,304.00	102.2536
5000	26 – 27	21	11	496,495.75	183.81
10000	26 – 27	18	8	441,895.83	337.54

Tabla 6.1.15. Resultados promedio del segundo sistema, para la instancia de 96 vuelos con parámetros iniciales. Fuente: elaboración propia.

- **Mejores soluciones encontradas:**

La mejor solución encontrada, de las 50 soluciones con 2500 iteraciones, tuvo un tiempo de ejecución de 83.93 ms, semilla 1573201970. Conformada por las tripulaciones de la tabla 6.1.16:

96 vuelos / 2500 iteraciones	
Crew 1: 1, 55, 56, 17	Costo: $7979.17
Crew 2: 3, 5, 15, 91, 92, 18	Costo: $9958.33
Crew 3: 33, 34, 12, 21	Costo: $9395.83
Crew 4: 2, 10, 85, 86	Costo: $9770.83
Crew 5: 4, 67, 68, 18	Costo: $8354.17
Crew 6: 43, 46, 75, 76	Costo: $9416.67
Crew 7: 6, 11, 20, 24	Costo: $9395.83
Crew 8: 35, 40, 16, 21	Costo: $8645.83
Crew 9: 79, 80, 18, 23	Costo: $9354.17
Crew 10: 65, 66, 69, 70	Costo: $8625.00
Crew 11: 37, 42, 47, 48	Costo: $9020.83
Crew 12: 7, 41, 44, 22	Costo: $7895.83
Crew 13: 39, 38, 45, 46	Costo: $7958.33
Crew 14: 8, 14, 23, 28	Costo: $9770.83
Crew 15: 11, 20, 51, 52	Costo: $9458.33
Crew 16: 11, 17, 81, 82	Costo: $8791.67
Crew 17: 14, 19, 83, 57, 58, 31	Costo: $9375.00
Crew 18: 85, 88 ,89, 90	Costo: $9937.50
Crew 19: 87 ,88, 93, 94	Costo: $10250.00
Crew 20: 22, 95, 84	Costo: $5875.00
Crew 21: 71, 74, 77, 78	Costo: $8750.00
Crew 22: 26, 32	Costo: $3604.17

Crew 23: 27, 30, 53, 54	Costo: $7020.83
13 «deadheads». ID: 11, 14, 17, 18, 20, 21, 22, 23, 46, 85, 88	Penalización: $91000.00
16 vuelos sin cubrir. ID: 9, 13, 25, 29, 36, 49, 50, 59, 60, 61, 62, 63, 64, 72, 73, 96	Penalización: $190400.00
Costo total	$408,004.17

Tabla 6.1.16. Mejor solución encontrada por el segundo algoritmo en 2500 iteraciones, para el horario de 96 vuelos con parámetros iniciales. Fuente: elaboración propia.

La mejor solución encontrada, de las 50 soluciones con 5000 iteraciones, tuvo un tiempo de ejecución de 172.82 ms, semilla 1573206304. Conformada por los «crews» de la tabla 6.1.17:

96 vuelos / 5000 iteraciones	
Crew 1: 1, 55, 56, 19	Costo: $8729.17
Crew 2: 3, 5, 15, 16	Costo: $8458.33
Crew 3: 61, 62, 13, 16	Costo: $8416.67
Crew 4: 33, 34, 12, 20	Costo: $8645.83
Crew 5: 2, 10, 85, 86	Costo: $9770.83
Crew 6: 4, 41, 42, 24	Costo: $9770.83
Crew 7: 6, 11, 20, 25	Costo: $9583.33
Crew 8: 79, 80, 16, 21	Costo: $8604.17
Crew 9: 65, 66, 43, 46	Costo: $8479.17
Crew 10: 37, 38, 69, 70	Costo: $8354.17
Crew 11: 19, 27, 63, 64	Costo: $7791.67
Crew 12: 7, 67, 68, 47, 27	Costo: $9750.00
Crew 13: 39, 40, 45, 48	Costo: $8770.83

Crew 14: 22, 31, 77, 78	Costo: $9041.67
Crew 15: 13, 14, 57, 58	Costo: $8291.67
Crew 16: 91, 92, 81, 82	Costo: $8812.50
Crew 17: 12, 17, 71, 72	Costo: $8041.67
Crew 18: 17, 18, 26, 49, 50, 32	Costo: $8437.50
Crew 19: 45, 46, 83, 90	Costo: $9166.67
Crew 20: 87, 88, 75, 76	Costo: $7375.00
Crew 21: 23, 29, 59, 60	Costo: $7479.17
Crew 22: 51, 52, 93, 94	Costo: $7791.67
Crew 23: 73, 74, 53, 54	Costo: $7104.17
Crew 24: 83, 84	Costo: $3979.17
11 «deadheads». ID: 12, 13, 16, 17, 19, 20, 27, 45, 46, 83	Penalización: $77000.00
10 vuelos sin cubrir. ID: 8, 9, 28, 30, 35, 36, 44, 89, 95, 96	Penalización: $119000.00
Costo total	$396,645.83

Tabla 6.1.17. Mejor solución encontrada por el segundo algoritmo en 5000 iteraciones, para el horario de 96 vuelos con parámetros iniciales. Fuente: elaboración propia.

La mejor solución encontrada, de las 50 soluciones con 10000 iteraciones, tuvo un costo de 374,112.50, un tiempo de ejecución de 334.908 ms, semilla 1573206514, y quedó conformada por las tripulaciones de la tabla 6.1.18:

96 vuelos / 10000 iteraciones	
Crew 1: 1, 6, 41, 55, 56, 15	Costo: $7770.83
Crew 2: 3, 91, 92, 39, 38, 14	Costo: $7708.33
Crew 3: 61, 62, 10, 12, 19, 22	Costo: $10458.33
Crew 4: 33, 34, 14, 19	Costo: $7895.83

Crew 5: 49, 50, 77, 78	Costo: $8291.67
Crew 6: 4, 11, 20, 25	Costo: $9958.33
Crew 7: 37, 40, 87, 88	Costo: $9791.67
Crew 8: 65, 66, 16, 21	Costo: $8416.67
Crew 9: 7, 11, 15, 24	Costo: $8645.83
Crew 10: 79, 80, 45, 46	Costo: $8666.67
Crew 11: 27, 30, 93, 94	Costo: $7791.67
Crew 12: 5, 13, 47, 48	Costo: $9020.83
Crew 13: 51, 52, 53, 54	Costo: $7020.83
Crew 14: 10, 14, 57, 58	Costo: $9041.67
Crew 15: 41, 42, 71, 74	Costo: $9791.67
Crew 16: 67, 68, 87, 82	Costo: $8500.00
Crew 17: 15, 69, 70, 73, 83, 96	Costo: $8958.33
Crew 18: 43, 44, 81, 84	Costo: $9270.83
Crew 19: 17, 18, 63, 64	Costo: $8541.67
Crew 20: 85, 86, 75, 76	Costo: $8729.17
Crew 21: 18, 23, 89, 90	Costo: $8416.67
Crew 22: 26, 29, 59, 60	Costo: $6729.17
Crew 23: 26, 32	Costo: $3604.17
Crew 24: 28, 31, 77, 78	Costo: $6791.67
13 «deadheads». ID: 10, 11, 14, 15, 18, 19, 26, 41, 77, 78, 87	Penalización: $91000.00
7 vuelos sin cubrir. ID: 2, 8, 9, 35, 36, 72, 95	Penalización: $83300.00
Costo total	$374,112.50

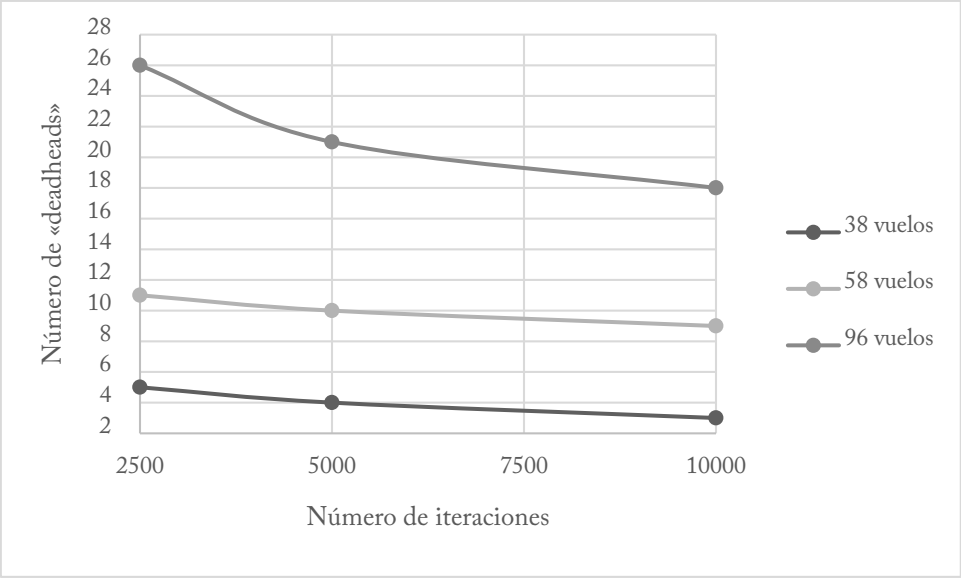
Tabla 6.1.18. Mejor solución encontrada por el segundo algoritmo en 10000 iteraciones, para el horario de 96 vuelos con parámetros iniciales. Fuente: elaboración propia.

De los resultados obtenidos, puede observarse que los valores promedio para todas las iteraciones incluyen soluciones con vuelos sin cubrir y con presencia de «deadheadings». Las mejores soluciones, a pesar de ser mucho más baratas que el promedio, no satisfacen todas las condiciones y sufren grandes penalizaciones por ambas restricciones blandas. Las mejores semillas fueron pasadas a ejecutarse con más iteraciones, sin entregar algún resultado que mejorara en costo a comparación de los mejores entregados por cada número de iteraciones.

Efectos del incremento en el número de iteraciones

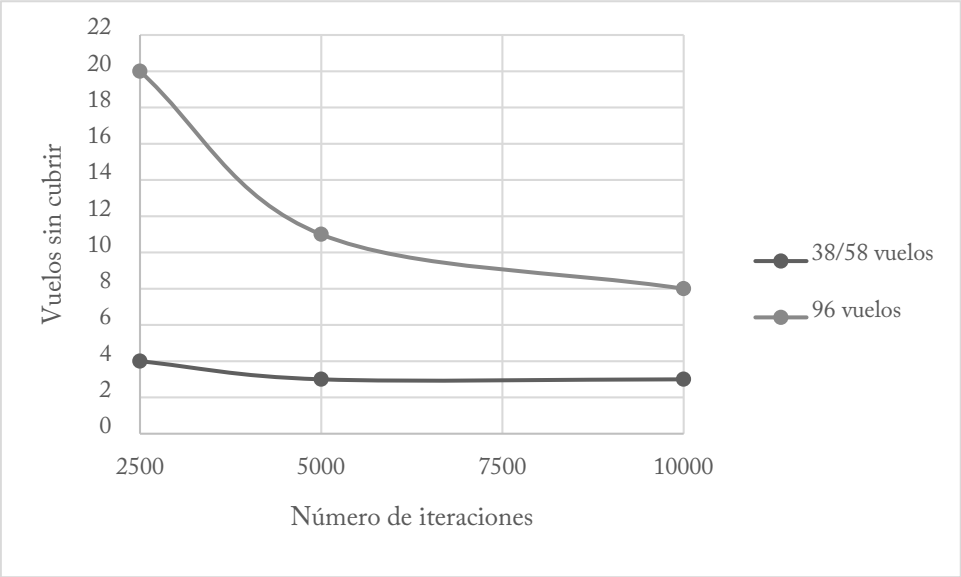
Con los resultados promedio obtenidos previamente, se pueden realizar distintas observaciones acerca del efecto que tiene el incrementar el número de iteraciones con respecto al desempeño del algoritmo, y de la calidad de las soluciones encontradas.

Si se grafica el número de «deadheads» promedio en cada solución contra el número de iteraciones, se observa cómo estos disminuyen de manera sutil conforme se duplica el número de iteraciones, por lo que no es un comportamiento perfectamente lineal. Obsérvese la gráfica 6.1.4:



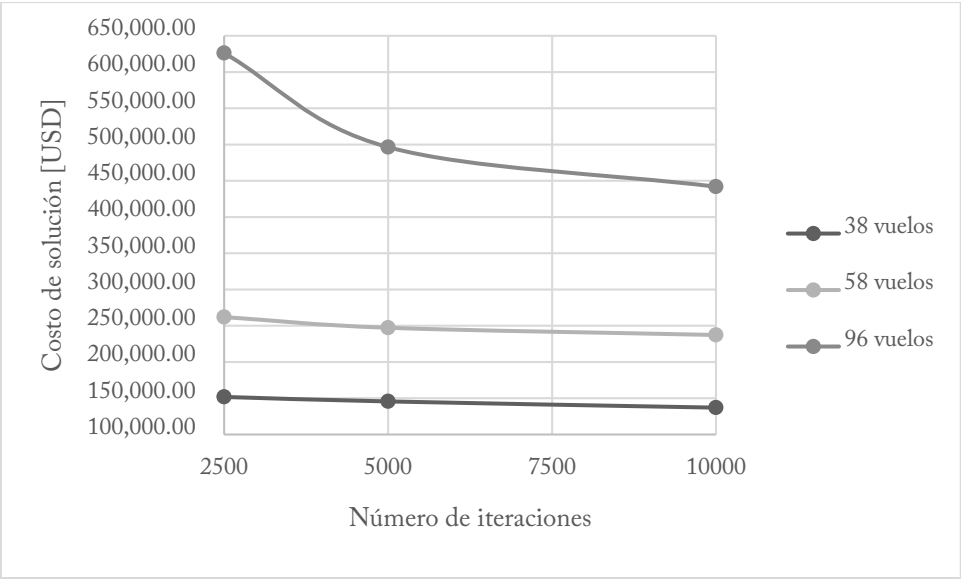
Gráfica 6.1.4. Número de «deadheads» promedio contra número de iteraciones, para el segundo sistema inteligente con parámetros iniciales. Fuente: elaboración propia.

Si se observa el comportamiento de los vuelos sin cubrir contra el número de iteraciones, para el caso de 38 y 58 vuelos, los vuelos sin cubrir disminuyen muy ligeramente al duplicar las iteraciones; mientras que para el horario de 96 vuelos se nota que el aumentar las iteraciones le ayuda mucho al algoritmo a disminuir la cantidad de vuelos sin cubrir. Obsérvese la gráfica 6.1.5:



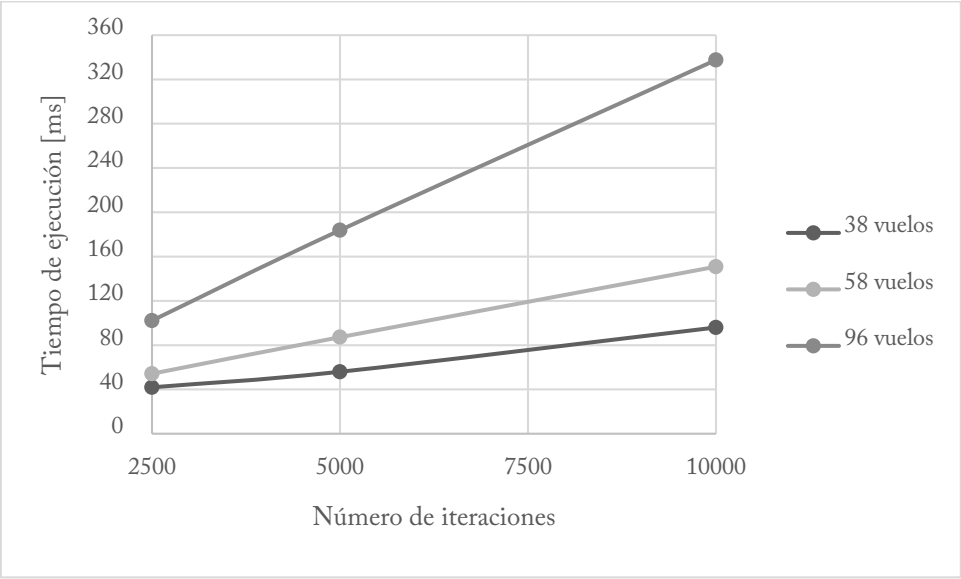
Gráfica 6.1.5. Número promedio de vuelos sin cubrir contra número de iteraciones, para el segundo sistema inteligente con parámetros iniciales. Fuente: elaboración propia.

Asimismo, el costo promedio de la solución disminuye conforme las iteraciones se aumentan. Lo hace de manera sutil para las instancias de 38 y 58, mientras que para la de 96 vuelos la disminución del costo es más notable. Véase la gráfica 6.1.6:



Gráfica 6.1.6. Costo promedio de solución contra número de iteraciones, para el segundo sistema inteligente con parámetros iniciales. Fuente: elaboración propia.

Finalmente, el comportamiento del tiempo de ejecución en relación al número de iteraciones es el más lineal de todos, como se observa en la gráfica 6.1.7. Las 3 instancias son prácticamente rectas; sin embargo, puede notarse que la recta del horario de 96 vuelos tiene una pendiente mayor a la de los otros dos:

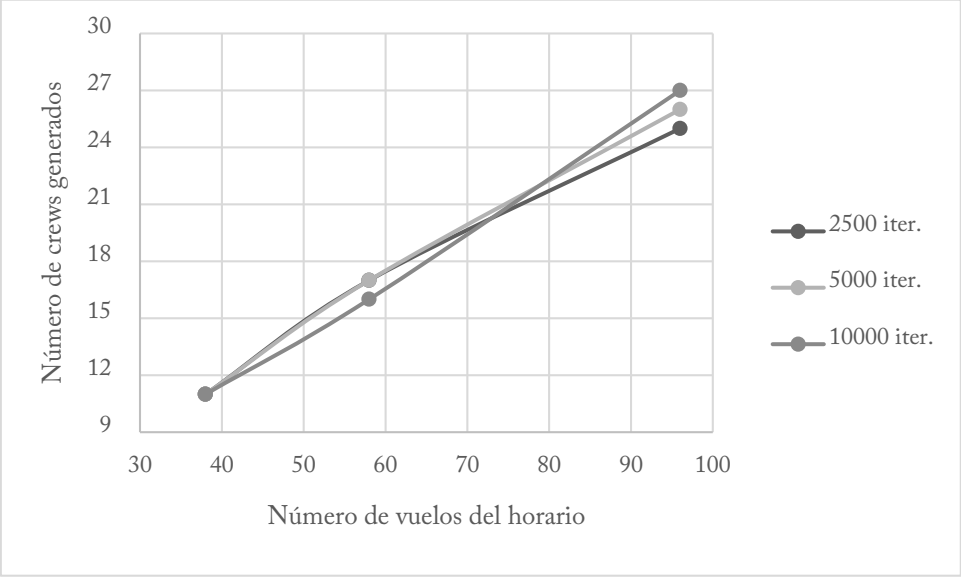


Gráfica 6.1.7. Tiempo promedio de ejecución contra número de iteraciones, para el segundo sistema inteligente con parámetros iniciales. Fuente: elaboración propia.

Efectos del incremento en el número de vuelos

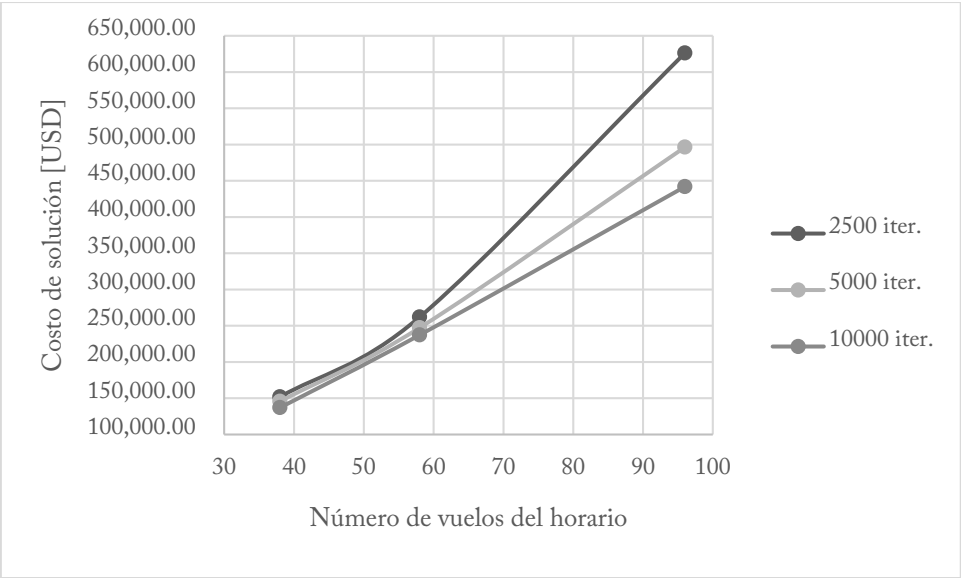
Además del análisis de resultados con relación al número de iteraciones, también se puede observar el comportamiento y desempeño del algoritmo conforme aumenta el número de vuelos del horario que tiene que resolver.

Si se grafica el número de tripulaciones generadas contra número de vuelos, el comportamiento tiene una tendencia lineal. El número de «crews» crece linealmente con relación al número de vuelos del horario. Además de notarse que casi no hay variación entre las distintas iteraciones. Véase la gráfica 6.1.8:



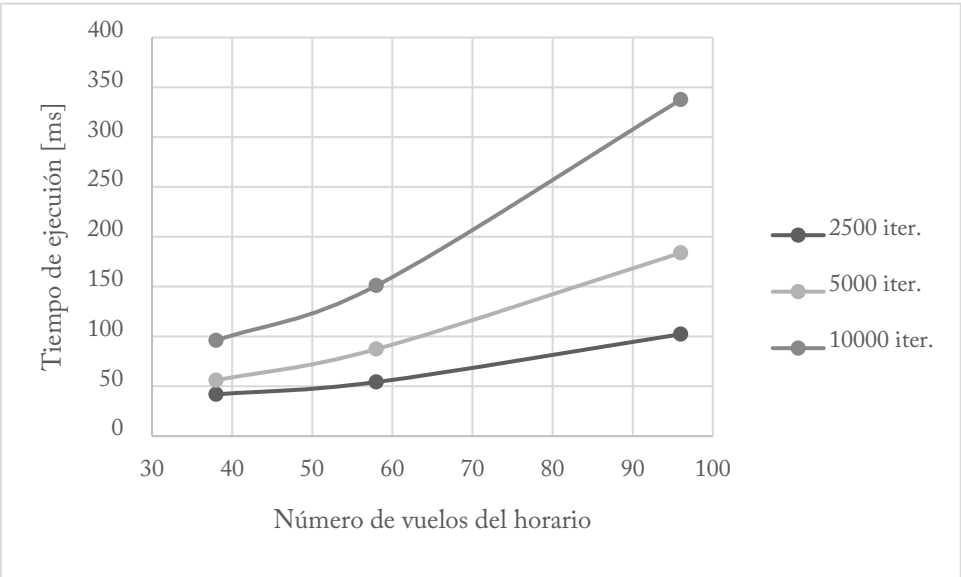
Gráfica 6.1.8. Número de «crews» promedio contra número de vuelos, para el segundo sistema inteligente con parámetros iniciales. Fuente: elaboración propia.

Ahora bien, si se realiza la relación entre costo de la solución generada contra número de vuelos, también se nota un comportamiento que tiende a ser lineal. Aunque el aumento en el número de iteraciones disminuye el costo, la tendencia lineal se mantiene para los 3 casos de iteraciones. Véase la gráfica 6.1.9:



Gráfica 6.1.9. Costo promedio de solución contra número de vuelos, para el segundo sistema inteligente con parámetros iniciales. Fuente: elaboración propia.

Para finalizar, el tiempo de ejecución en relación al número de vuelos. Es de notarse que el tiempo de ejecución tiene un incremento de tipo exponencial muy sutil que se asemeja a ser lineal. El número de iteraciones aumenta el tiempo de ejecución promedio, pero la similitud en las gráficas se mantiene en todos los casos. Véase la gráfica 6.1.10:



Gráfica 6.1.10. Tiempo promedio de ejecución en milisegundos contra número de vuelos, para el segundo sistema inteligente con parámetros iniciales. Fuente: elaboración propia.

6.2 Análisis de resultados con variación de parámetros

La segunda serie de pruebas se enfocó en modificar ciertos valores de parámetros o restricciones, los cuales afectaran de alguna manera al algoritmo, ya sea en su funcionamiento, o en la calidad de las soluciones que este encontraba. Los experimentos fueron hechos principalmente para comparaciones individuales de cada algoritmo, pues la mayoría de las pruebas entregaron resultados que no son de utilidad para comparativas entre ambos sistemas.

6.2.1 Resultados obtenidos con el primer sistema

La modificación de parámetros en el primer sistema se basó en reducir y aumentar las restricciones temporales, para así observar si este era capaz de entregar soluciones teniendo unas restricciones de tiempo más estrictas, y ver si las soluciones mejoraban otorgándole mayor libertad con restricciones temporales menos estrechas. Las etapas de esta sección y los resultados obtenidos se encuentran comentados en las tablas 6.2.1 y 6.2.3:

Ajuste de restricciones temporales

Etapa	Instancia	Observaciones
1. Ajuste: aumento de tiempo mínimo de espera y disminución del tiempo máximo de espera.	38 vuelos	No hubo resultados para esta instancia, el algoritmo no fue capaz de entregar ninguna solución. Se estuvo reiniciando una y otra vez por más de 5 minutos en una sola ejecución, sin poder terminar una construcción. Se concluye que, para esta instancia no existen soluciones que sean capaces de satisfacer los nuevos valores configurados para ese par de restricciones.
	58 vuelos	Para este horario, el algoritmo sí fue capaz de entregar soluciones, aunque ninguna que las satisficiera al 100 %. Lo más interesante de las pruebas es que el tiempo de ejecución se disparó, al algoritmo le costaba mucho más trabajo entregar una solución con estas restricciones temporales. El tiempo de ejecución promedio pasó de 84.85064 ms a 4.782 s.
	96 vuelos	Al igual que en la instancia anterior, el sistema fue capaz de entregar soluciones, aunque tampoco ninguna que cumpliera con todas las restricciones, igual que lo fue para el caso de las restricciones por defecto. El tiempo de ejecución se incrementó de 1.1918981 s, a 14.896 s.

2. Ajuste: reducción de horas máximas de trabajo.	38 vuelos	El algoritmo fue capaz de encontrar soluciones para esta instancia, pero ninguna que cumpliera por completo todas las restricciones. Se encontró un nuevo mínimo de precio para esta instancia. Esta solución se podría considerar como la mejor global encontrada por el primer sistema, ya que trabajar 10 horas respeta claramente el máximo de 12. La solución mencionada se encuentra en la tabla 6.2.2.
	58/96 vuelos	Con un máximo de 10 horas de trabajo, el primer sistema no tuvo la capacidad de organizar estos horarios. Seguramente se deba a que para estas instancias no existan soluciones que satisfagan ese máximo de horas de trabajo reducido.

Tabla 6.2.1. Resultados de los experimentos con el ajuste de restricciones temporales, en el primer sistema inteligente. Fuente: elaboración propia.

Etapa 1: de los resultados obtenidos en la etapa 1 se concluye que, el primer sistema inteligente es capaz de lidiar, aunque con gran dificultad, con restricciones de tiempo más apretadas, siempre y cuando existan soluciones que cumplan estas restricciones en el horario que se le introduzca. Al tener menor libertad en cuestiones de tiempo, las alternativas que puede tomar también se ven reducidas, y esto conlleva a un aumento drástico en el tiempo de ejecución.

Etapa 2: de los resultados obtenidos para etapa 2 se concluye que, el algoritmo tiene la posibilidad de encontrar soluciones para horarios con jornadas de trabajo más reducidas, con apenas incremento en el tiempo de ejecución; por supuesto, siempre y cuando que para los vuelos del horario existan soluciones que las satisfagan.

Adicionalmente, también se observa la complejidad natural del espacio de búsqueda en los problemas de optimización, ya que el sistema, por como está programado, procura utilizar al máximo las horas de trabajo de las tripulaciones, cuando pueden existir soluciones mejores dentro del espacio que no necesariamente utilicen todo el tiempo disponible de cada «crew», como se observó en el nuevo mínimo de la instancia de 38 vuelos, ubicado en la tabla 6.2.2.

Crew 1: 2, 3, 35	Costo: $6812.50
Crew 2: 1, 32, 15, 19	Costo: $7750.00
Crew 3: 29, 12, 17	Costo: $5520.83

Crew 4: 31, 10, 16, 23	Costo: $8500.00
Crew 5: 4, 7, 20, 34	Costo: $8291.67
Crew 6: 9	Costo: $1000.00
Crew 7: 13, 33, 21, 22	Costo: $8000.00
Crew 8: 8, 11, 26, 36	Costo: $6854.17
Crew 9: 30, 5, 38, 28	Costo: $6979.17
Crew 10: 14, 18	Costo: $3250.00
Crew 11: 25, 24	Costo: $2750.00
Crew 12: 37, 27	Costo: $2750.00
1 «crew» sin retorno: 6	Penalización: $7000.00
1 vuelo sin cubrir. ID: 6	Penalización: $11900.00
Costo total	$87,358.34
Semilla	1573519496

Tabla 6.2.2. Mejor solución encontrada por el primer algoritmo, para el horario de 38 vuelos con reducción en la jornada de trabajo. Fuente: elaboración propia.

Ensanche de restricciones temporales

Para el incremento en los valores, solamente se aumentó el valor del número máximo de horas de trabajo, para así observar si el algoritmo encontraba mejores soluciones por el hecho de tener mayor libertad, en el sentido poder ausentar a los «crews» hasta 24 h de su base. Los resultados se encuentran detallados en la tabla 6.2.3:

Etapa	Instancia	Observaciones
1. Ajuste: aumento del largo de la jornada de trabajo	38 vuelos	Como era de esperarse, no hubo ninguna dificultad en la construcción de soluciones para el horario de 38 vuelos. El tiempo de ejecución promedio se redujo ligeramente de 81 a 79.0432 ms, aunque se considera que la reducción es más producto de variaciones en las pruebas mismas. Las soluciones encontradas presentaban más variedad en cuanto a vuelos cubiertos debido a la extensión de horas de trabajo, y se encontraron varios resultados que satisfacen por completo todas las restricciones; el más económico hallado está alojado en la tabla 6.2.4.

	58 vuelos	Para las ejecuciones del horario de 58 vuelos, se vio una tendencia del algoritmo a cubrir un número de vuelos mayor en las soluciones. El número de vuelos sin cubrir promedio mejoró de 1 a 0.65, con la mayoría de resultados cubriendo todos los vuelos, con pocas o ninguna tripulación sin retorno. El tiempo promedio de ejecución se mantuvo prácticamente igual, 84.2054 ms. La mejor solución encontrada, no distinta a la obtenida previamente, cubrió todos los vuelos y todos los «crews» regresaron a sus bases.
	96 vuelos	Para el caso de la instancia de 96 vuelos, el sistema mejoró el promedio de vuelos sin cubrir, disminuyéndolo de 4 a 3. El tiempo de ejecución varió ligeramente de 1.19 segundos a 1.19533. La mejor solución encontrada tuvo un costo de 246,987 dólares, resultando en 2 vuelos sin cubrir y sólo 1 tripulación que pasó el resto del día en otra ciudad. Esta solución no es considerada como la mejor encontrada por el primer sistema para esta estancia, pues la duración de la jornada es incrementada al doble y se sigue sin cumplir por completo las condiciones.

Tabla 6.2.3. Resultados de los experimentos con el ensanche de restricciones temporales, en el primer sistema inteligente. Fuente: elaboración propia.

De los resultados obtenidos con el incremento de la jornada de trabajo se concluye que, el primer sistema inteligente no tiene dificultad alguna en la construcción de soluciones con restricciones temporales menos estrechas, aunque tampoco recibe una gran ventaja en cuanto a tiempo de ejecución, seguramente debido a que sólo se modificó uno de los valores, quizás se aumentara más de uno, como el tiempo máximo de horas de vuelo, repercutiría en una mejora más notoria en el tiempo de ejecución.

Por otro lado, gracias a que la única limitante es la cantidad de horas de vuelo, las soluciones cubren más vuelos por tripulación, entregando soluciones con valores promedio mejores que las de los parámetros iniciales, pero que no son directamente comparables. Además, se observó cómo el hecho de aumentar el tiempo máximo de trabajo le sirvió al algoritmo para encontrar soluciones completas en la instancia de 38 vuelos.

Crew 1: 2, 6, 17, 30, 5, 38	Costo: $14416.67
Crew 2: 1, 32, 8, 11, 26, 36	Costo: $12916.67

Crew 3: 29, 3, 33, 14, 18, 28	Costo: $14520.83
Crew 4: 31, 10, 16, 23, 37, 27	Costo: $12750.00
Crew 5: 4, 7, 15, 19, 21, 22	Costo: $10541.67
Crew 6: 9, 12, 20, 34, 25, 24	Costo: $12000.00
Crew 7: 13, 35	Costo: $2750.00
Todos los crews volvieron a su base	Penalización: $0.00
Todos los vuelos cubiertos	Penalización: $0.00
Costo total	$79,895.84
Semilla	1573638853

Tabla 6.2.4. Solución completa encontrada por el primer algoritmo, para el horario de 38 vuelos con ampliación de jornada de trabajo. Fuente: elaboración propia.

6.2.2 Resultados obtenidos con el segundo sistema

La modificación de parámetros para el segundo sistema inteligente consistió en variar los aquellos propios del algoritmo y no de las restricciones, para afectar únicamente a la etapa de Simulated Annealing y no a la de generación de «pairings». El objetivo fue observar el comportamiento del algoritmo y de las soluciones halladas. Los resultados y las etapas fueron los siguientes:

Modificación del «máx_núm_crews_por_vuelo»

Se incrementó y disminuyó el valor del número máximo de «crews» que pueden generarse a partir de un vuelo inicial, para observar qué comportamientos tenía el sistema al tener un número mayor de «pairings» generados, y por ende un espacio de búsqueda mayor; y uno menor al disminuirlos. Los resultados promedio se encuentran en las tablas 6.2.6 y 6.2.8, mientras que el análisis de estos se encuentra en la tabla 6.2.5:

Etapa	Instancia	Observaciones
1. Ajuste: incremento del espacio de búsqueda. Ver resultados	38 vuelos	Puede notarse que el número de «deadheads» promedio disminuye en comparación a los resultados de parámetros por defecto, así como el costo de la solución, pero a cambio, hay un incremento de aproximadamente 60 ms en el tiempo promedio de ejecución. La mejor solución encontrada tuvo un costo de

promedio en tabla 6.2.6.		$93,233.33, contando con un solo «deadhead» y un vuelo sin cubrir. Se puede replicar con la semilla 1574047316.
	58 vuelos	Los resultados obtenidos fueron mejores en comparación a los de los parámetros por defecto, pues se disminuye el número de «deadheads» promedio, así como el número de vuelos sin cubrir. Adicionalmente, se halló una nueva mejor solución para la instancia de 58 vuelos. Con un costo de $194,145.83, tiempo de ejecución de 210.887 ms y semilla 1573524848, quedó conformada como lo muestra la tabla 6.2.7.
	96 vuelos	A diferencia de los horarios anteriores, para la instancia de 96 vuelos no se encontraron resultados que mejoraran con respecto a los anteriores con parámetros por defecto; de hecho, la mayoría de estos empeoran. Nótese además que el tiempo de ejecución no se ve incrementado de manera significativa.
2. Ajuste: reducción del espacio de búsqueda. Ver resultados promedio en tabla 6.2.8.	38/58 vuelos	De los resultados promedio obtenidos, es de notarse que estas instancias se ven perjudicadas con la disminución de «pairings» en la mayoría de resultados. El único valor que disminuye es, por obvias razones, el tiempo de ejecución, aunque tampoco en gran cantidad. Finalmente, no se encontró ninguna solución que fuera mejor que las halladas previamente para ninguno de estos dos horarios.
	96 vuelos	De manera interesante, los resultados para la instancia de 96 vuelos se vieron beneficiados con la disminución de la cantidad de tripulaciones. Los resultados promedio mejoraron de manera significativa en comparación a los obtenidos anteriormente, y se obtuvo una nueva mejor solución para este horario de vuelos. La disminución del espacio de búsqueda ayuda al algoritmo a realizar una explotación más intensa, y aprovechar de mejor manera el número de iteraciones, entregando, en consecuencia, mejores resultados. La mejor solución para este horario es la que se encuentra en la tabla 6.2.9.

Tabla 6.2.5. Resultados de los experimentos con variación del espacio de búsqueda, en el segundo sistema inteligente. Fuente: elaboración propia.

Etapa 1 - Incremento del espacio de búsqueda:

Los resultados promedio obtenidos para los 3 horarios se encuentran en la tabla 6.2.6:

Número de vuelos	Número de «crews» generados	Número de «deadheads»	Número de vuelos no cubiertos	Costo de solución [USD]	Tiempo de ejecución [ms]
38	10 – 11	4	3	122,795.83	156.4736
58	16	7	2	232,664.99	233.257
96	26	18	9 – 10	471,586.01	350.691

Tabla 6.2.6. Resultados promedio de las 3 instancias, para el segundo sistema con *máx_núm_crews_por_vuelo* incrementado. Fuente: elaboración propia.

De los experimentos basados en la ampliación del tamaño de búsqueda se concluye que, para las instancias de 38 y 58 vuelos, el sistema se vio beneficiado con una mayor cantidad de «pairings», permitiéndole realizar más combinaciones entre ellos y así entregar resultados que satisfacen en mejor manera las condiciones puestas, al punto de incluso haber encontrado una mejor solución para el horario de 58 vuelos. Por supuesto, algo se tuvo que dar a cambio, y esto fue el tiempo de ejecución, el cual se vio incrementado. Para el caso de los 96 vuelos, los resultados no fueron favorables; se pueden deducir distintas teorías acerca del porqué, la más clara es que el aumento del espacio de búsqueda y el limitado número de iteraciones, hagan que no se logre una explotación lo suficientemente intensa para poder entregar resultados de mejor calidad.

Crew 1: 1, 4, 33, 35	Costo: $9562.50
Crew 2: 41, 51, 11, 21	Costo: $9250.00
Crew 3: 27, 30, 3, 6	Costo: $9729.17
Crew 4: 42, 50, 29, 32	Costo: $8625.00
Crew 5: 7, 18, 10, 21	Costo: $9250.00
Crew 6: 43, 52, 46, 53, 47, 56	Costo: $10375.00
Crew 7: 8, 19, 12, 23	Costo: $9625.00
Crew 8: 17, 2, 5, 13	Costo: $8416.67
Crew 9: 9, 20, 15, 25	Costo: $10000.00
Crew 10: 44, 54, 37, 39	Costo: $8666.67

Crew 11: 28, 31, 13, 22	Costo: $7562.50
Crew 12: 10, 21, 16, 26	Costo: $8500.00
Crew 13: 45, 55, 34, 36	Costo: $8458.33
Crew 14: 14, 24, 49, 58	Costo: $7562.50
Crew 15: 38, 40, 49, 58	Costo: $6500.00
Crew 16: 48, 57, 49, 58	Costo: $6062.50
8 «deadheads». ID: 10, 13, 21, 49, 58	Penalización: $56000.00
Todos los vuelos cubiertos	Penalización: $0.00
Costo total	$194,145.83

Tabla 6.2.7. Mejor solución encontrada por el segundo algoritmo, para el horario de 58 vuelos con incremento del espacio de búsqueda y 10000 iteraciones. Fuente: elaboración propia.

Etapa 2 – Reducción del espacio de búsqueda:

Los resultados promedio obtenidos para los 3 horarios de vuelo se encuentran alojados en la tabla 6.2.8:

Número de vuelos	Número de «crews» generados	Número de «deadheads»	Número de vuelos no cubiertos	Costo de solución [USD]	Tiempo de ejecución [ms]
38	11	4	3	124,684.33	98.356
58	16	7	2	237,778.3	145.229
96	25 – 26	9	8	368,017.79	303.764

Tabla 6.2.8. Resultados promedio de las 3 instancias, para el segundo sistema con *máx_núm_crews_por_vuelo* disminuido. Fuente: elaboración propia.

De los experimentos realizados con disminución del espacio de búsqueda se concluye que el reducir su tamaño puede ofrecer ciertas ventajas o desventajas. Si bien es cierto que se disminuye el tiempo de ejecución, hay que recordar que las técnicas incompletas por sí mismas ya son rápidas, y que al final puede resultar perjudicial para el algoritmo el hecho de tener menos opciones por las cuales optar en su búsqueda, como ocurrió para las instancias de 38 y 58 vuelos. Por otro lado, si el espacio de búsqueda es grande de por sí debido a otros factores, como la

cantidad de vuelos, existe la posibilidad de que un acotamiento extra en el espacio le resulte de utilidad, y le ayude a intensificar más por el hecho de reducir las posibilidades por las que se puede «desviar», tal como ocurrió para la instancia de 96 vuelos.

Crew 1: 1, 55, 56, 17	Costo: $7979.17
Crew 2: 89, 90	Costo: $3750.00
Crew 3: 61, 62, 10, 12, 17, 22	Costo: $10458.33
Crew 4: 33, 36, 12, 19	Costo: $7895.83
Crew 5: 4, 41, 42, 18	Costo: $8270.83
Crew 6: 35, 40, 85, 86	Costo: $9145.83
Crew 7: 43, 44, 28, 31	Costo: $8645.83
Crew 8: 65, 66, 16, 21	Costo: $8416.67
Crew 9: 7, 11, 15, 45, 46, 27	Costo: $9937.50
Crew 10: 9, 13, 47, 48	Costo: $8770.83
Crew 11: 39, 38, 69, 70	Costo: $8104.17
Crew 12: 79, 80, 87, 88	Costo: $10250.00
Crew 13: 91, 92, 81, 82	Costo: $8812.50
Crew 14: 67, 68, 51, 52	Costo: $9166.67
Crew 15: 14, 20, 83, 84	Costo: $9270.83
Crew 16: 57, 58, 63, 64	Costo: $6375.00
Crew 17: 71, 72, 77, 78	Costo: $8750.00
Crew 18: 23, 32	Costo: $4354.17
Crew 19: 24, 30, 93, 94	Costo: $8541.67
Crew 20: 49, 50, 53, 54	Costo: $7770.83
Crew 21: 26, 29, 59, 60	Costo: $6729.17
Crew 22: 73, 74, 77, 78	Costo: $7625.00
Crew 23: 95, 96	Costo: $2750.00
Crew 24: 75, 76	Costo: $3250.00
4 «deadheads». ID: 12, 17, 77, 78	Penalización: $28000.00
8 vuelos sin cubrir. ID: 2, 3, 5, 6, 8, 25, 34, 37	Penalización: $95200.00
Costo total	$308,220.83
Semilla	1574136919

Tabla 6.2.9. Mejor solución encontrada por el segundo algoritmo, para el horario de 96 vuelos con reducción del espacio de búsqueda y 10000 iteraciones. Fuente: elaboración propia.

Modificación del «coeficiente_temperatura»

Se incrementó el valor del coeficiente, con el objetivo de incrementar la exploración inicial que el algoritmo realiza antes de que explote en una zona fija del espacio de búsqueda. Se realizaron algunas ejecuciones con semillas obtenidas previamente, para que la generación de tripulaciones fuera la misma y sólo variara la etapa de SA. Los resultados promedio son los de la tabla 6.2.10:

Número de vuelos	Número de «crews» generados	Número de «deadheads»	Número de vuelos no cubiertos	Costo de solución [USD]	Tiempo de ejecución [ms]
38	11 – 12	5	2 – 3	132,232.84	102.255
58	16 – 17	8	1 – 2	229,132.17	157.996
96	26	11	10	447,867.85	326.437

Tabla 6.2.10. Resultados promedio de las 3 instancias, para el segundo sistema con *coeficiente_temperatura* incrementado. Fuente: elaboración propia.

De los resultados obtenidos se concluye que, el hecho de incrementar la exploración inicial no tuvo una relevancia significativa en los resultados obtenidos, pues las variaciones que se presentan en los resultados promedio no son de gran magnitud. Las semillas reutilizadas no entregaron ningún resultado que fuera mejor al previo, de hecho, estos empeoraron en costo. Tampoco se encontró ninguna solución para ninguna instancia que fuera mejor que las obtenidas previamente. Lo cierto es que, disminuir la exploración incluso más que su valor por defecto traería efectos negativos más resaltables a los resultados promedio, ya que se le estaría extrayendo parte importante del funcionamiento de Simulated Annealing.

Modificación de penalizaciones

La modificación de penalizaciones se realizó en dos subetapas: la primera de ellas consistió en disminuir el valor de penalización por «deadhead», y la segunda en colocar el mismo valor para las penalizaciones de «deadhead» y de vuelo sin cubrir. Las pruebas se realizaron para observar los valores promedio de este par de fenómenos en las soluciones. Los resultados fueron los siguientes:

Subetapa 1 – Disminución de penalización por «deadhead»:

Los resultados promedio obtenidos para las tres estancias están en la tabla 6.2.11. Nótese que ya no se incluye el costo promedio de la solución, debido a que los costos se ven afectados por el cambio en las penalizaciones y ya no son comparables con el resto, además de no ser de interés para esta modificación:

Número de vuelos	Número de «crews» generados	Número de «deadheads»	Número de vuelos no cubiertos	Tiempo de ejecución [ms]
38	12	9	0	100.065
58	18	18	0	152.559
96	32	45	0	352.656

Tabla 6.2.11. Resultados promedio de las 3 instancias, para el segundo sistema con *pen_por_deadhead* disminuido. Fuente: elaboración propia.

Como puede observarse en la tabla 6.2.11, los resultados promedio cambian mucho en comparación a los obtenidos anteriormente. Estos cubren todos los vuelos la gran mayoría de veces, a cambio de un aumento drástico en el número de «deadheads», y de un incremento gradual en el número de tripulaciones; lo cual concuerda con la modificación de las penalizaciones. Tal como fueron modificadas, el sistema inteligente «prefiere» tener 6 «deadheads» en una solución, antes que dejar un vuelo sin cubrir, por ello es el incremento drástico en la cantidad de estos por solución, sobre todo para la instancia de 96 vuelos.

Subetapa 2 – Igualación de penalizaciones:

Los resultados promedio obtenidos en esta subetapa se encuentran en la tabla 6.2.12. Al igual que en el caso anterior, no se incluye el costo de la solución debido a la modificación de las penalizaciones:

Número de vuelos	Número de «crews» generados	Número de «deadheads»	Número de vuelos no cubiertos	Tiempo de ejecución [ms]
38	10	1	3	99.856
58	12 – 13	2	10	161.43
96	22 – 23	7	17	352.029

Tabla 6.2.12. Resultados promedio de las 3 instancias, para el segundo sistema con *pen_por_deadhead* y *pen_por_vuelo_sin_cubrir* igualados. Fuente: elaboración propia.

En este caso, se puede observar una disminución importante en el número de «deadheads» promedio, a cambio de un incremento en el número de vuelos que quedan descubiertos. Aunque pudiera pensarse que el hecho de igualar las penalizaciones conllevaría a un equilibrio entre vuelos sin cubrir y «deadheadings», se nota que el sistema entrega soluciones con más vuelos sin cubrir, no porque los prefiera, ya que para el algoritmo es igual de perjudicial tener un vuelo sin cubrir que un «deadhead»; sino porque le es más fácil encontrar soluciones con vuelos sin cubrir y un número de tripulaciones menor, por ello es la reducción notable en el número de «crews» generados.

De los experimentos realizados mediante la variación de penalizaciones se concluye que, el hecho de utilizar una técnica incompleta en el segundo sistema y el propio funcionamiento de la misma, permite hacer inclinaciones hacia el tipo de resultados que se desean obtener, es decir, si se desea priorizar el número de vuelos cubiertos a cambio de más «deadheads», es posible gracias a los valores de las penalizaciones; lo mismo ocurre si estos se intentan minimizar a cambio de un mayor número de vuelos sin cubrir. Esto es una gran ventaja para situaciones donde no se lograsen encontrar soluciones que satisfagan por completo ninguna de las restricciones blandas.

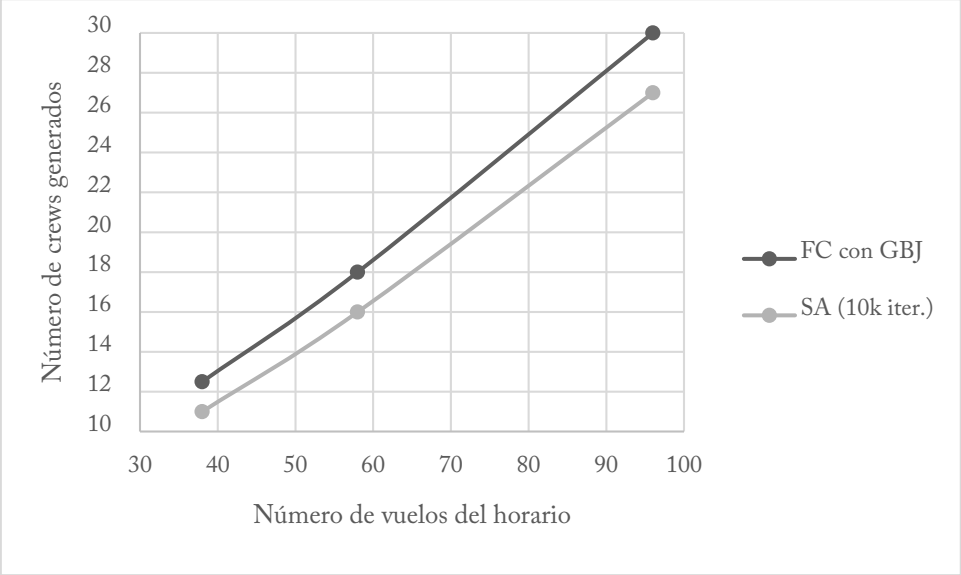
6.3 Comparativas entre ambos sistemas

En esta última sección se confrontarán los resultados obtenidos, así como el funcionamiento de ambos sistemas inteligentes, con el objetivo de observar cómo estos se comportaron bajo las distintas circunstancias que fueron sometidos, y ver en qué momento fue mejor o peor uno sobre

el otro. Las comparativas serán hechas con los resultados que sean equiparables, ya que no todos los resultados lo son.

6.3.1 Personal requerido

El personal requerido para cubrir los vuelos programados consiste en el número de «crews» promedio, los cuales son generados en relación al número de vuelos. La gráfica 6.3.1 ilustra la cantidad de personal requerido para ambos algoritmos:



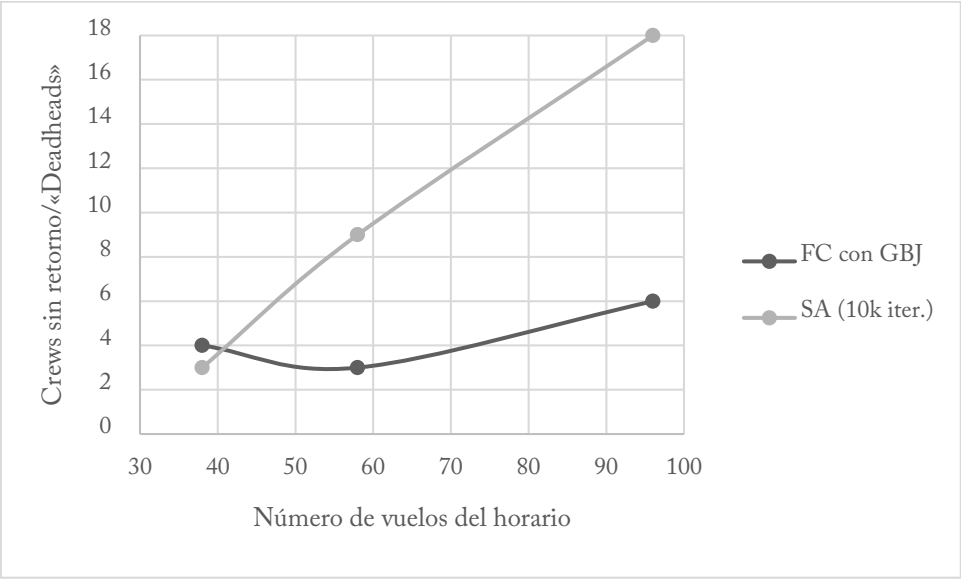
Gráfica 6.3.1. Tripulaciones promedio contra número de vuelos, para ambos sistemas inteligentes con parámetros iniciales. Fuente: elaboración propia.

Como se observa en la gráfica 6.3.1, ambos sistemas poseen un incremento lineal en el número de «crews» generado con relación al número de vuelos. Por otro lado, el segundo sistema tuvo unos números promedio de tripulaciones menores (2, 2 y 3, respectivamente) a los del primer sistema. Un número de tripulaciones menor es mejor siempre y cuando no crezca el número de vuelos sin cubrir, lo cual no ocurrió en el caso del segundo sistema.

6.3.2 Incumplimiento de restricciones blandas

Para este caso, ya que ambos sistemas poseen fenómenos (penalizaciones) diferentes, se comparará el número de tripulaciones sin retorno contra el número de «deadheadings». Se les colocó el mismo valor de penalización, pues se considera que el hecho de que una tripulación pase el resto del día en otra ciudad, es equivalente a que si esta volviera como pasajera en un

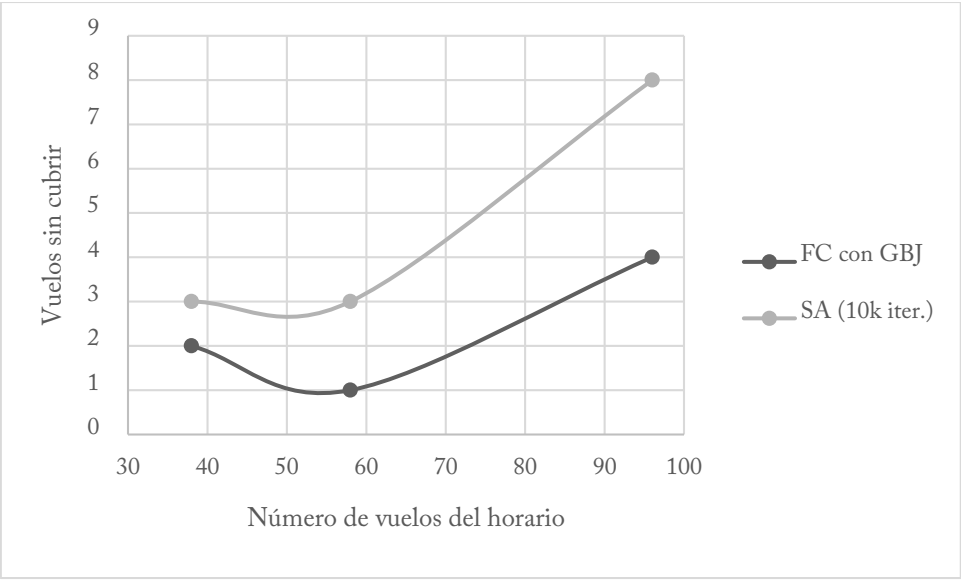
vuelo a la base de donde partió. Por otro lado, la penalización que ambos sistemas comparten es la de vuelos sin cubrir. Los resultados pueden observarse en las gráficas 6.3.2.1 y 6.3.2.2:



Gráfica 6.3.2.1. Tripulaciones sin retorno y «deadheads» contra número de vuelos, para ambos sistemas inteligentes con parámetros iniciales. Fuente: elaboración propia.

Se puede notar de la gráfica 6.3.2.1 que el segundo sistema tiene un incremento lineal en cuanto a «deadheadings» conforme aumenta el número de vuelos. Sin embargo, el primer sistema tiene un comportamiento que no parece seguir ningún patrón, ya que disminuye para 58 vuelos, pero aumenta para 96; lo más probable es que se deba a que utiliza técnicas completas, y esto conlleve a que el número de tripulaciones sin retorno dependa de la «complejidad» del horario; caso contrario en la segunda técnica, donde se observa que depende del número de vuelos. En este apartado, la primera IA tiene un mejor desempeño que la segunda, pues el número de «crews» sin retorno es menor en dos instancias.

El número de vuelos promedio sin cubrir para ambos sistemas, en relación al número de vuelos, se encuentra en la gráfica 6.3.2.2:

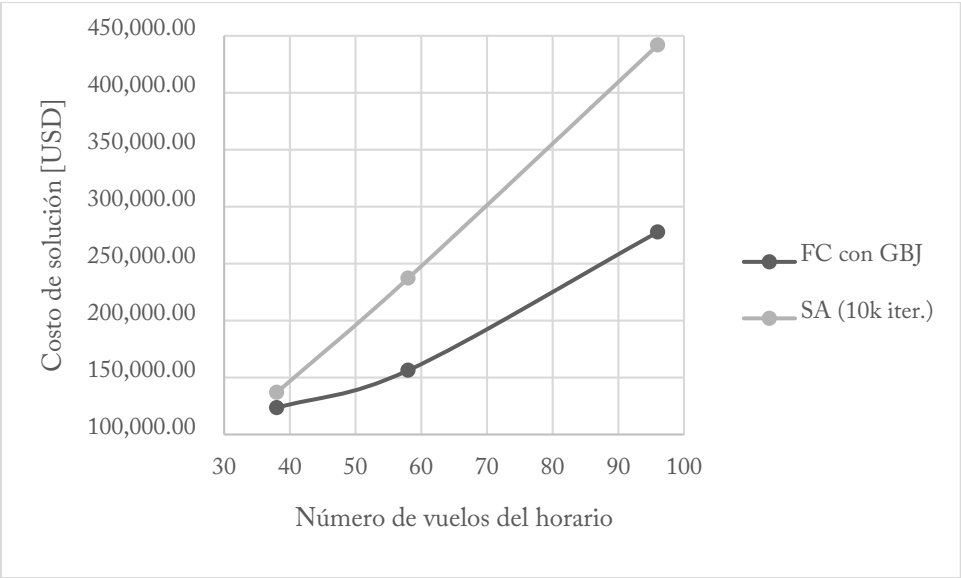


Gráfica 6.3.2.2. Número promedio de vuelos sin cubrir contra número de vuelos, para ambos sistemas inteligentes con parámetros iniciales. Fuente: elaboración propia.

Como puede observarse en la gráfica 6.3.2.2, ambos sistemas tienen un comportamiento bastante similar, a pesar de que la segunda técnica tenga más vuelos sin cubrir que la primera. Por otro lado, pese a que su comportamiento es parecido, este no tiene ninguna tendencia lógica, por lo que puede deducirse que el número de vuelos sin cubrir no depende del sistema ni de la cantidad de vuelos de la instancia, sino de la «complejidad» del horario a resolver. Por ejemplo, el hecho de que los vuelos sin cubrir disminuyan en el horario de 58 vuelos, se debe a que para este los sistemas sí fueron capaces de encontrar soluciones completas, mas no para las instancias de 38 y 96.

6.3.3 Costo de soluciones

En lo que respecta al costo promedio de las soluciones halladas contra el número de vuelos, se encuentran en la gráfica 6.3.3:

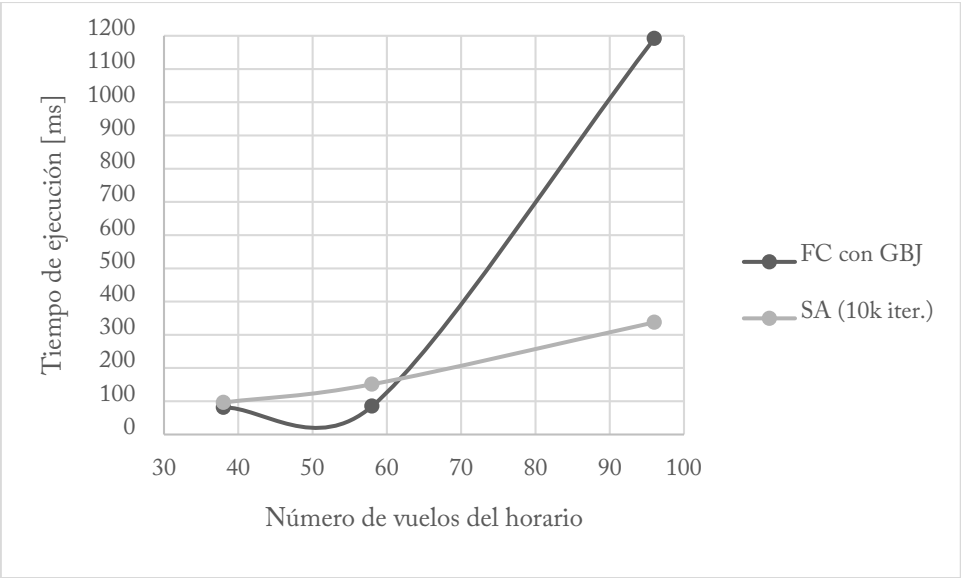


Gráfica 6.3.3. Costo de solución promedio contra número de vuelos, para ambos sistemas inteligentes con parámetros iniciales. Fuente: elaboración propia.

Lo primero que se puede notar de la gráfica 6.3.3 es que ambos sistemas tienen un comportamiento suficientemente lineal en el incremento del precio promedio conforme al aumento de los vuelos. Sin embargo, es de observarse que el segundo sistema tiene una pendiente mayor a la del primero, es decir, el precio aumenta linealmente en mayor cantidad conforme lo hacen los vuelos. En conclusión, el primer sistema, al utilizar técnicas de tipo completa, entrega mejores soluciones en cuanto a precio comparadas con las entregadas por el segundo sistema, lo cual confirma la teoría de que las técnicas completas son mejores que las incompletas, siempre y cuando puedan obtener las soluciones en un tiempo razonable.

6.3.4 Tiempo de ejecución

La gráfica 6.3.4 muestra los resultados obtenidos en cuanto a tiempo de ejecución contra número de vuelos, para ambos sistemas inteligentes:



Gráfica 6.3.4. Tiempo promedio de ejecución en milisegundos contra número de vuelos, para ambos sistemas inteligentes con parámetros iniciales. Fuente: elaboración propia.

Como se comentó previamente, en el primer sistema se observa el fenómeno conocido como explosión combinatoria, donde el tiempo requerido para encontrar soluciones se ve incrementado de forma exponencial conforme el número de vuelos aumenta. En el segundo sistema, aunque el tiempo de ejecución es ligeramente mayor para las instancias de 38 y 58 vuelos debido a la cantidad de iteraciones, se mantiene un incremento uniforme (lineal) y no exponencial, de tal manera que el tiempo necesario para la resolución de la instancia de 96 vuelos en este algoritmo es 3.5 veces menor en comparación al primero.

Sin duda alguna, la segunda IA se lleva la ventaja mayor en cuanto a tiempo de ejecución, y se confirma la parte teórica que menciona que las técnicas incompletas, aunque no entregan soluciones de tan buena calidad como las técnicas completas, son mejores en cuanto a tiempo de ejecución, y se recomienda utilizarlas cuando las completas no pueden hallar soluciones de calidad en determinado tiempo.

6.3.5 Mejores soluciones encontradas

En esta sección final se ponen a comparación las mejores soluciones encontradas de cada sistema para cada horario de vuelos, y se harán los comentarios y comparaciones respectivos sobre cada uno de ellos.

Horario de 38 vuelos

Forward Checking con Graph Back Jumping		**Simulated Annealing**	
Crew 1: 2, 6, 10, 16, 20, 35	Costo: $10500.00	Crew 1: 26, 36	Costo: $2750.00
Crew 2: 1, 32, 8, 11	Costo: $9479.17	Crew 2: 31, 10, 16, 23	Costo: $8500.00
Crew 3: 29, 12, 15, 19, 23	Costo: $9020.83	Crew 3: 1, 32, 8, 11	Costo: $9479.17
Crew 4: 31, 13, 33, 26	Costo: $10000.00	Crew 4: 29, 3, 33, 23	Costo: $8833.33
Crew 5: 4, 7, 17, 30, 22	Costo: $10375.00	Crew 5: 4, 7, 17, 34	Costo: $8291.67
Crew 6: 9	Costo: $1000.00	Crew 6: 9, 12, 20, 30, 22	Costo: $9833.33
Crew 7: 35, 21, 5, 38, 28	Costo: $10145.83	Crew 7: 13, 35, 21, 5, 38	Costo: $10333.33
Crew 8: 14, 18	Costo: $3250.00	Crew 8: 37, 27	Costo: $2750.00
Crew 9: 25, 24	Costo: $2750.00	Crew 9: 14, 18	Costo: $3250.00
Crew 10: 37, 27	Costo: $2750.00	Crew 10: 2, 6, 15, 19	Costo: $8500.00
Crew 11: 36	Costo: $1000.00	Crew 11: 25, 24	Costo: $2750.00
		Crew 12: 38, 28	Costo: $2562.50
2 crews sin retorno: 6, 11	Pen.: $14000.00	2 «deadheads». ID: 23, 38	Pen.: $14000.00
1 vuelo sin cubrir. ID: 3	Pen.: $11900.00	Todos los vuelos cubiertos	Pen.: $0.00
Costo total	$96,170.83	Costo total	$91,833.33

Tabla 6.3.1. Mejores soluciones encontradas de ambos sistemas para la instancia de 38 vuelos. Fuente: elaboración propia.

Con los resultados mostrados en la tabla 6.3.1 puede observarse que, para el horario de 38 vuelos, la mejor solución hallada fue por el segundo sistema. Cabe mencionar que no se incluyó la nueva mejor solución encontrada por el primer sistema, de cuando se disminuyó el máximo de horas de trabajo, aunque esta respete el máximo de 12 h, pues se considera que el hecho de modificar una restricción temporal vuelve a los resultados directamente como no comparables.

En conclusión, al ser un horario con el número de vuelos más reducido, ambos sistemas fueron capaces de encontrar soluciones de gran calidad en general, aunque ninguna completa. El segundo sistema cubre todos los vuelos a cambio de dos «deadheadings», mientras que la primera

deja un vuelo sin cubrir y dos tripulaciones varadas. Claro está que el hecho de que se haya encontrado una mejor solución con el segundo sistema, no lo excluye de que los resultados promedio obtenidos con este hayan sido peores que los obtenidos con el primero.

Horario de 58 vuelos

Forward Checking con Graph Back Jumping		**Simulated Annealing**	
Crew 1: 1, 4, 46, 55	Costo: $9541.67	Crew 1: 1, 4, 33, 35	Costo: $9562.50
Crew 2: 7, 19, 33, 35	Costo: $9270.83	Crew 2: 41, 51, 11, 21	Costo: $9250.00
Crew 3: 41, 51, 29, 32	Costo: $9375.00	Crew 3: 27, 30, 3, 6	Costo: $9729.17
Crew 4: 27, 30, 10, 21	Costo: $9125.00	Crew 4: 42, 50, 29, 32	Costo: $8625.00
Crew 5: 42, 52, 45, 54, 12, 22	Costo: $10500.00	Crew 5: 7, 18, 10, 21	Costo: $9250.00
Crew 6: 43, 50, 3, 6	Costo: $8729.17	Crew 6: 43, 52, 46, 53, 47, 56	Costo: $10375.00
Crew 7: 8, 18, 11, 24	Costo: $10000.00	Crew 7: 8, 19, 12, 23	Costo: $9625.00
Crew 8: 17, 2, 5, 13	Costo: $8416.67	Crew 8: 17, 2, 5, 13	Costo: $8416.67
Crew 9: 9, 20, 37, 39	Costo: $9416.67	Crew 9: 9, 20, 15, 25	Costo: $10000.00
Crew 10: 44, 53, 47, 56	Costo: $8000.00	Crew 10: 44, 54, 37, 39	Costo: $8666.67
Crew 11: 28, 31, 14, 23	Costo: $7937.50	Crew 11: 28, 31, 13, 22	Costo: $7562.50
Crew 12: 15, 25, 49, 58	Costo: $6812.50	Crew 12: 10, 21, 16, 26	Costo: $8500.00
Crew 13: 34, 36	Costo: $3083.33	Crew 13: 45, 55, 34, 36	Costo: $8458.33
Crew 14: 38, 40	Costo: $3250.00	Crew 14: 14, 24, 49, 58	Costo: $7562.50
Crew 15: 16, 26	Costo: $2750.00	Crew 15: 38, 40, 49, 58	Costo: $6500.00
Crew 16: 48, 57	Costo: $2750.00	Crew 16: 48, 57, 49, 58	Costo: $6062.50
Todos los crews volvieron a su base	Pen.: $0.00	8 «deadheads». ID: 10, 13, 21, 49, 58	Pen.: $56000.00
Todos los vuelos cubiertos	Pen. $0.00	Todos los vuelos cubiertos	Pen.: $0.00
Costo total	$118,958.34	Costo total	$194,145.83

Tabla 6.3.2. Mejores soluciones encontradas de ambos sistemas para la instancia de 58 vuelos. Fuente: elaboración propia.

De la tabla 6.3.2 se puede observar que ambos resultados cubren en su totalidad los vuelos, lo cual hace que la calidad de ambas soluciones sea excelente por este simple hecho. Sin embargo, el primer sistema inteligente logró que todas las tripulaciones regresaran a sus bases, mientras que el segundo sistema tuvo la presencia de 8 «deadheads», lo que le aumentó una penalización de 56,000 dólares, cantidad bastante alta. Además, cabe mencionar que la mejor solución elegida

para el segundo sistema es aquella obtenida en la etapa de incremento del espacio de búsqueda, ya que la variación de parámetros del algoritmo entrega resultados que son directamente comparables.

Finalmente, se puede concluir que aparte de que los resultados promedio son mejores en la primera IA, esta también entregó la mejor de ambos sistemas. No obstante, la mejor solución hallada por el segundo sistema no se queda tan atrás de la del primero, ya que lo único que la diferencia es la gran penalización sufrida por la presencia de «deadheads»; en otras palabras, el desempeño de la técnica incompleta fue bastante bueno, aunque no mejor, en comparación al de las técnicas completas, para esta instancia en particular.

Horario de 96 vuelos

Forward Checking con Graph Back Jumping		**Simulated Annealing**	
Crew 1: 1, 8, 14, 6, 41, 42	Costo: $7750.00	Crew 1: 1, 55, 56, 17	Costo: $7979.17
Crew 2: 3, 91, 92, 22	Costo: $10166.67	Crew 2: 89, 90	Costo: $3750.00
Crew 3: 61, 62, 10, 16	Costo: $8416.67	Crew 3: 61, 62, 10, 12, 17, 22	Costo: $10458.33
Crew 4: 33, 56, 19	Costo: $7666.67	Crew 4: 33, 36, 12, 19	Costo: $7895.83
Crew 5: 2, 7, 11, 17, 18, 23	Costo: $10562.50	Crew 5: 4, 41, 42, 18	Costo: $8270.83
Crew 6: 4, 67, 68, 25	Costo: $10041.67	Crew 6: 35, 40, 85, 86	Costo: $9145.83
Crew 7: 55, 36, 43, 44, 24	Costo: $9895.83	Crew 7: 43, 44, 28, 31	Costo: $8645.83
Crew 8: 35, 40, 47, 48	Costo: $9520.83	Crew 8: 65, 66, 16, 21	Costo: $8416.67
Crew 9: 79, 80, 69, 70	Costo: $8812.50	Crew 9: 7, 11, 15, 45, 46, 27	Costo: $9937.50
Crew 10: 65, 66, 45, 46	Costo: $8479.17	Crew 10: 9, 13, 47, 48	Costo: $8770.83
Crew 11: 5, 13, 87, 88	Costo: $9791.67	Crew 11: 39, 38, 69, 70	Costo: $8104.17
Crew 12: 37, 38, 85, 86	Costo: $8645.83	Crew 12: 79, 80, 87, 88	Costo: $10250.00
Crew 13: 9, 15, 81, 82	Costo: $10291.67	Crew 13: 91, 92, 81, 82	Costo: $8812.50
Crew 14: 39	Costo: $1083.33	Crew 14: 67, 68, 51, 52	Costo: $9166.67
Crew 15: 12, 20, 83, 84	Costo: $10020.83	Crew 15: 14, 20, 83, 84	Costo: $9270.83
Crew 16: 21, 73, 32	Costo: $5416.67	Crew 16: 57, 58, 63, 64	Costo: $6375.00
Crew 17: 57, 58, 59, 60	Costo: $8312.50	Crew 17: 71, 72, 77, 78	Costo: $8750.00
Crew 18: 71, 72, 77, 78	Costo: $8750.00	Crew 18: 23, 32	Costo: $4354.17
Crew 19: 49, 50, 53, 54	Costo: $7770.83	Crew 19: 24, 30, 93, 94	Costo: $8541.67
Crew 20: 26, 29	Costo: $2854.17	Crew 20: 49, 50, 53, 54	Costo: $7770.83
Crew 21: 27, 30, 93, 94	Costo: $7791.67	Crew 21: 26, 29, 59, 60	Costo: $6729.17

Crew 22: 51, 52	Costo: $2916.67	Crew 22: 73, 74, 77, 78	Costo: $7625.00
Crew 23: 95, 96	Costo: $2750.00	Crew 23: 95, 96	Costo: $2750.00
Crew 24: 28, 31	Costo: $2854.17	Crew 24: 75, 76	Costo: $3250.00
Crew 25: 75, 76	Costo: $3250.00		
Crew 26: 63, 64	Costo: $2750.00		
Crew 27: 89, 90	Costo: $3750.00		
1 crew sin retorno: 14	Pen.: $7000.00	4 «deadheads». ID: 12, 17, 77, 78	Pen.: $28000.00
2 vuelos sin cubrir. ID: 34, 74	Pen.: $23800.00	8 vuelos sin cubrir. ID: 2, 3, 5, 6, 8, 25, 34, 37	Pen.: $95200.00
Costo total	$221,112.52	Costo total	$308,220.83

Tabla 6.3.3. Mejores soluciones encontradas de ambos sistemas para la instancia de 96 vuelos. Fuente: elaboración propia.

La instancia de 96 vuelos fue la más complicada de resolver para ambos sistemas, debido al tamaño del espacio de búsqueda. Las mejores soluciones para ambos, como se observa en la tabla 6.3.3, a pesar de ser mucho más baratas que los promedios, ninguna pudo cubrir por completo los vuelos ni carecer de «deadheads» o tripulaciones sin retorno; sin embargo, la solución de la primera IA es con diferencia la más barata de ambas, ya que solamente peca de 2 vuelos sin cubrir y una tripulación varada. Es importante mencionar que la mejor solución mostrada del segundo sistema es aquella que se obtuvo en la etapa de reducción del espacio de búsqueda.

Como conclusión, puede decirse que para esta instancia es cuando se tuvo una diferencia mayor en cuanto a mejores resultados se refiere. No obstante, se debe tener presente que al primer algoritmo le tomó 4 veces más de tiempo hallar esta solución que al segundo, que ciertamente es mejor en precio, pero si en lugar de tiempos en milisegundos se hablara de horas o días, la diferencia sería más que considerable, y seguramente se optaría por algún resultado de la técnica incompleta. Esto, por supuesto, es lo que ocurre en la práctica real dentro del área de investigación de operaciones, ya que las aerolíneas pueden llegar a tener programados decenas de miles de vuelos para intervalos de semanas o meses completos, y la única alternativa viable para tener los «pairings» listos en un tiempo razonable, sea optar por una técnica incompleta.

Para finalizar la sección de comparativas entre los dos sistemas, y en específico la sección de las mejores soluciones halladas, se considera relevante el mencionar algunos comentarios en relación al porqué se eligieron las mejores soluciones de distintas etapas para el segundo algoritmo, y no todas las de los parámetros iniciales. Las técnicas incompletas y SA en concreto, tienen características (parámetros) propias de la manera en que trabajan, cuyos valores no están predefinidos y tienen que ser establecidos por la persona quien las diseñe.

Lo cierto es que no hay forma de saber cuáles son los valores óptimos para cada uno de ellos, cada problema es diferente y, por lo tanto, no existe ninguna manera de asegurarse de qué valores son los que darán lugar a los mejores resultados. De hecho, podrían incluso ser programados para que estos cambiaran de forma dinámica y no como constantes, aunque hacerlo incrementaría en gran manera la complejidad del algoritmo. En resumen, lo importante es tener presente que la variación y «sintonización» de los parámetros son algo propio y esencial del diseño de técnicas incompletas.

7. Conclusiones

Se diseñaron y programaron dos sistemas inteligentes en lenguaje C, los cuales se encargan de encontrar soluciones al problema de «Airline Crew Scheduling» mediante el uso de técnicas completas e incompletas. A estos se les realizaron distintos perfeccionamientos que, aunque no forman parte de las técnicas implementadas en sí, les son de gran utilidad para orientarse bajo variadas situaciones y evitar que se atasquen en la mayoría de ellas; además de algunos aportes extra que mejoran y facilitan la búsqueda de soluciones como parte adicional dentro del funcionamiento de las técnicas.

Los algoritmos fueron puestos a prueba con 3 instancias de vuelos distintas para jornadas de trabajo de un día, siendo capaces de entregar resultados de calidad que compiten con los hallados en el artículo del que los horarios fueron extraídos. Asimismo, se sometieron a experimentos mediante la variación de valores de parámetros y restricciones, pudiéndose observar si estos fueron capaces o no de funcionar bajo ellos, y analizar los efectos producidos en cuanto a funcionamiento y calidad de las soluciones. Posteriormente se hicieron comparativas individuales y entre ambos sistemas, cuyos resultados mostraron un comportamiento que concuerda con la teoría de las técnicas de solución para problemas de optimización.

El primer sistema inteligente, que utiliza técnicas completas, resultó entregar en la mayoría de casos mejores resultados que el segundo en cuanto al precio de las soluciones; sin embargo, para el horario con más vuelos programados ocurrió el fenómeno conocido como explosión combinatoria, lo que causó que el tiempo de ejecución se viera incrementado de forma drástica. Por otro lado, la técnica incompleta, a pesar de entregar resultados mayores en costo, tuvo un tiempo de ejecución menor y siempre proporcional al número de vuelos y de iteraciones, que en ningún momento se vio aumentado de la misma manera que en el primer algoritmo.

Siguiendo los comportamientos antes mencionados, aun cuando el primer algoritmo es capaz de entregar mejores resultados, si se siguiera aumentando el número de vuelos, se llegaría a un punto en que se vuelve inviable el tener que esperar a que este encontrase una solución, y se aceptarían las soluciones halladas en el segundo por el simple hecho de que la técnica incompleta podría entregarlas en un tiempo mucho menor.

La técnica incompleta implementada no solamente tiene como ventaja un tiempo de ejecución menor a las completas. También, aunque podría realizarse algo similar con las técnicas completas mediante modificaciones en su funcionamiento, algo propio de la incompleta es el permitir priorizar el tipo de soluciones que se desean. Como se observó en la etapa de modificación de penalizaciones, el hecho de aumentar o disminuir la penalización recibida por incumplir una restricción, afecta directamente al tipo de soluciones que se encuentran, pues el algoritmo procura evitar aquellas que inflijan las penalizaciones más altas. Por supuesto que no existe nada mejor que el no incumplir con ninguna restricción, como sería el caso de un óptimo global hallado por una técnica completa, pero puede resultar de gran utilidad para ocasiones en las que se necesiten hacer prioridades.

Por otro lado, aunque se considera que los sistemas diseñados son bastante robustos gracias a las mejoras implementadas, no se exentan de poder ser mejorados aún más, lo cual podría hacerse a futuro como continuación de este proyecto. Algunas de las mejoras que no fueron implementadas son las siguientes:

La primera de ellas es aplicable a ambos algoritmos. Consiste en una nueva característica en sus códigos que permitiera poder ejecutarlos por una duración de tiempo establecida, es decir, que estos se mantuvieran buscando soluciones durante un periodo ingresado manualmente, y cuando el tiempo terminara, mostraran y exportaran la mejor solución hallada de toda la ejecución.

Otra de ellas, para el primer sistema, sería que en vez de reiniciar la construcción de la solución por completo ya que se ha alcanzado el número máximo de reinicios de tripulación, podría implementarse un reinicio gradual, y que este se regresara un cierto número de tripulaciones hacia atrás para comenzar de nuevo desde allí y no desde el principio. Esto sería una mejora directa a la técnica de GBJ, ya que le permitiría regresar aún más en el tiempo, no sólo a nivel de una misma tripulación.

Tanto para la primera como para la segunda IA (en la etapa de generación de «pairings»), se podría hacer una modificación importante, la cual fue descubierta cuando se experimentó con la reducción de la jornada de trabajo. Los algoritmos procuran utilizar al máximo las horas de trabajo de los «crews», ya que la única manera de que estos terminen su jornada antes del máximo es cuando no tienen más vuelos por cubrir, sumado al hecho de que pueden llegar y volverse a ir de su base en algunas ocasiones mediante vuelos intermedios. En vez de que funcionara de esta

manera, se podría añadir la posibilidad de que, si estos han cubierto un número de vuelos menor y se encuentran en su base aun pudiendo continuar y cubrir más, exista la probabilidad de que su jornada termine quedándose allí. La mejor solución encontrada por el primer algoritmo, en la instancia de 38 vuelos, fue hallada debido a que se redujo la jornada de trabajo de las tripulaciones, forzándolas a cubrir menos vuelos.

Con lo mencionado en el párrafo anterior, se concluye finalmente que los problemas de optimización y en específico el ACSP, pueden poseer espacios de búsqueda muy complejos, donde existan óptimos que, contrario a lo que podría pensarse, no necesariamente utilicen al máximo las horas de trabajo de cada uno de los «crews» que los conformen. En otras palabras, las mejores soluciones no siempre estarán relacionadas directamente con un número de «crews» reducido, sino con un balance adecuado entre estos y la carga de trabajo de cada una de sus jornadas. Además, la misma complejidad del espacio fuerza la existencia de restricciones blandas, pues nunca se tendrá total certeza de, si para el horario de vuelos preestablecido, existen soluciones que satisfagan por completo todas las condiciones puestas, ya que hay que recordar que entre más realista se es, más restricciones deben considerarse para el planteamiento y resolución del problema.

Referencias

[1] L. Á. Munárriz, «Inteligencia Artificial,» de *Fundamentos de inteligencia artificial*, España, Universidad de Murcia, 1994, pp. 18-22.

[2] M. T. Jones, «AI, Problem Solving and Games,» de *Artificial Intelligence: A Systems Approach: A Systems Approach*, Massachusetts, Jones and Bartlett Publishers, 2009, pp. 4-5.

[3] J. Merodio, «Inteligencia Artificial en Marketing,» 20 noviembre 2018. [En línea]. Available: https://www.juanmerodio.com/inteligencia-artificial-marketing-digital/. [Último acceso: 11 mayo 2019].

[4] J. Worth, «Stop Calling it Artificial Intelligence,» 2 octubre 2016. [En línea]. Available: https://www.joshworth.com/stop-calling-in-artificial-intelligence/. [Último acceso: 12 mayo 2019].

[5] R. M. Cascallana, «Metaheurística para problemas de Scheduling con múltiples recursos,» 24 febrero 2017. [En línea]. Available: http://hdl.handle.net/10651/42772. [Último acceso: 15 mayo 2019].

[6] M. S. F. S. Atoosa Kasirzadeh, «Airline crew scheduling: models, algorithms, and data sets,» Springer-Verlag, Berlín, 2015.

[7] I. Ö. Bahadır Zeren, A novel column generation strategy for large scale airline crew pairing problems, Turquía: Elsevier, 2015.

[8] M. Negnevitsky, «Introduction to knowledge-based intelligent systems,» de *Artificial Intelligence: A Guide to Intelligent Systems*, England, Addison-Wesley, 2002, pp. 1-20.

[9] Real Academia Española, «Diccionario de la Lengua Española,» 2019. [En línea]. Available: https://dle.rae.es/?id=LqtyoaQ|LqusWqH. [Último acceso: 19 agosto 2019].

[10] C. C. M. Mody, «Moore's Law,» de *The Long Arm of Moore's Law: Microelectronics and American Science*, Massachusetts, Massachusetts Institute of Technology, 2017, pp. 7-9.

[11] W. P. Warren McCulloch, «A Logical Calculus of the Ideas Immanent in Nervous Activity,» 1943. [En línea]. Available: http://www.cse.chalmers.se/~coquand/AUTOMATA/mcp.pdf. [Último acceso: 15 mayo 2019].

[12] A. Turing, «Computing Machinery and Intelligence,» 1950. [En línea]. Available: https://phil415.pbworks.com/f/TuringComputing.pdf. [Último acceso: 15 mayo 2019].

[13] C. Shannon, «Programming a Computer for Playing Chess,» [En línea]. Available: https://vision.unipv.it/IA1/aa2009-2010/ProgrammingaComputerforPlayingChess.pdf. [Último acceso: 16 mayo 2019].

[14] J. McCarthy, «Programs with Common Sense,» 1958. [En línea]. Available: https://www.cs.cornell.edu/selman/cs672/readings/mccarthy-upd.pdf. [Último acceso: 27 agosto 2019].

[15] F. Rosenblatt, «Principles of Neurodynamics,» 1961 marzo 15. [En línea]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/256582.pdf. [Último acceso: 27 agosto 2019].

[16] H. S. Allen Newell, «GPS, a program that simulates human thought,» 1961. [En línea]. Available: http://digitalcollections.library.cmu.edu/awweb/awarchive?type=file&item=33607. [Último acceso: 17 mayo 2019].

[17] L. Zadeh, «Fuzzy Sets,» 1965. [En línea]. Available: https://www-liphy.ujf-grenoble.fr/pagesperso/bahram/biblio/Zadeh_FuzzySetTheory_1965.pdf. [Último acceso: 27 agosto 2019].

[18] G. Beekman, «Heurísticas,» de *Introducción a la computación*, Pearson Educación, 1999, p. 213.

[19] B. L. Feigenbaum, «On generality and problem solving: A case study using the DENDRAL program,» 1969. [En línea]. Available: https://stacks.stanford.edu/file/druid:wn105jh4920/wn105jh4920.pdf. [Último acceso: 17 mayo 2019].

[20] C. Darwin, «La selección natural o la supervivencia de los más aptos,» de *El origen de las especies*, México, UNAM, 1859, pp. 172-175.

[21] J. Holland, Adaptation in Natural and Artificial Systems, Massachussets: MIT Press, 1975.

[22] J. J. Hopfield, «Neural Networks and Physical Systems with Emergent Collective Computational Abilities,» abril 1982. [En línea]. Available: https://www.researchgate.net/publication/16246447_Neural_Networks_and_Physical_Systems_with_Emergent_Collective_Computational_Abilities. [Último acceso: 11 septiembre 2019].

[23] J. M. David Rumelhart, «Parallel Distributed Processing: Explorations in the Microstructure of Cognition,» 1986. [En línea]. Available: https://pdfs.semanticscholar.org/ff2c/2e3e83d1e8828695484728393c76ee07a101.pdf?_ga=2.179371875.613238393.1568251339-276376656.1568251339. [Último acceso: 11 septiembre 2019].

[24] P. F. Daniel Mcneill, «Fuzzy Logic,» de *Fuzzy Logic: The Revolutionary Computer Technology That Is Changing Our World*, Nueva York, Simon & Schuster, 1993, pp. 10-13.

[25] MIlenio, «La inteligencia artificial y sus aplicaciones,» 23 agosto 2018. [En línea]. Available: https://www.milenio.com/opinion/varios-autores/universidad-politecnica-de-tulancingo/inteligencia-artificial-y-sus-aplicaciones. [Último acceso: 12 septiembre 2019].

[26] A. Edelsten, «NVIDIA DLSS: Your Questions, Answered,» 15 febrero 2019. [En línea]. Available: https://www.nvidia.com/en-us/geforce/news/nvidia-dlss-your-questions-answered/. [Último acceso: 12 septiembre 2019].

[27] K. C. Viktor Mayer-Schönberger, «Big data,» de *Big data: la revolución de los datos masivos*, Madrid, Turner Noema, 2013, pp. 11-15.

[28] Unizar, «Ramas de la inteligencia artificial,» 10 julio 2010. [En línea]. Available: https://ciberconta.unizar.es/leccion/ia/200.HTM. [Último acceso: 12 septiembre 2019].

[29] Avansis, «Las ramas de la inteligencia artificial,» 12 abril 2019. [En línea]. Available: https://www.avansis.es/2019/04/12/las-ramas-de-la-inteligencia-artificial-tipos-de-inteligencia-artificial-suave/. [Último acceso: 12 septiembre 2019].

[30] Edureka, «Types Of Artificial Intelligence You Should Know,» 7 agosto 2019. [En línea]. Available: https://www.edureka.co/blog/types-of-artificial-intelligence/. [Último acceso: 12 septiembre 2019].

[31] D. Graupe, «Introduction and Role to Artificial Neural Networks,» de *Principles of Artificial Neural Networks*, Singapur, World Scientific, 2007, pp. 1-3.

[32] A. M. David Poole, «Optimization,» de *Artificial Intelligence Foundations of Computational Agents*, Cambridge, Cambridge University Press, 2017, pp. 144-155.

[33] A. Konar, «Problem Solving by Intelligent Search,» de *Artificial Intelligence and Soft Computing*, Estado Unidos, CRC Press, 2000, pp. 70-75.

[34] D. B. F. Zbigniew Michalewicz, «Hill-Climbing Methods, Traditional Methods,» de *How to Solve It: Modern Heuristics*, Alemania, Springer, 1998, pp. 43-45, 55-61.

[35] U. Montanari, «On the Forward Checking Algorithm,» de *Principles and Practice of Constraint Programming - CP '95*, Nueva York, Springer, 1995, pp. 292-295.

[36] P. v. B. T. W. Francesca Rossi, «Backjumping algorithms,» de *Handbook of Constraint Programming*, Ámsterdam, Elsevier, 2006, p. 240.

[37] M. S. G. Tsuzuki, «Adaptive Neighborhood Heuristics for Simulated Annealing over Continuous Variables,» de *Simulated Annealing: Advances, Applications and Hybridizations*, Rijeka, Intech , 2012, pp. 3-7.

[38] R. Doganis, «Direct Operating Costs,» de *Flying Off Course: The Economics of International Airlines*, Londres, Routledge, 1985, pp. 78-81.

[39] J. M. P. Gutiérrez, «Algorithm creation and optimization for the Crew Pairing Problem,» 2013 enero 25. [En línea]. Available: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/18963/2/jpuentesg_TFM_0113_eng.pdf. [Último acceso: 22 septiembre 2019].

[40] S. I. B. K. B. Hatice Tekiner, «Robust Crew Pairing for Managing Extra Flights,» 12 julio 2008. [En línea]. Available: http://research.sabanciuniv.edu/13085/1/RobustCP_ExtraFlights_Modeling_v1_4.pdf. [Último acceso: 2 noviembre 2019].

[41] R. Riedel, An Approach to Predict Operational Performance of Airline Schedules Using Aircraft Assignment Key Performance Indicators, Massachussets: Massachussets Institute of Technology, 2006.

[42] T. Grosche, «Maintenance Flights,» de *Computational Intelligence in Integrated Airline Scheduling*, Alemania, Springer, 2009, pp. 41-43.

[43] E. L. J. Balaji Gopalakrishnan, «Airline Crew Scheduling: State-of-the-Art,» 2005. [En línea]. Available: https://linux.ime.usp.br/~lucasc/mac499/bibliografia/GOPALAKRISHNAN_1995.pdf. [Último acceso: 22 septiembre 2019].

[44] K. Godfrey, «Cabin crew reveal what 'deadheading' on a flight means,» News AU, 26 enero 2019. [En línea]. Available: https://www.news.com.au/travel/travel-advice/flights/cabin-crew-reveal-what-deadheading-on-a-flight-means/news-story/720492cd9cb9a096bc78cc6d6d0bb935. [Último acceso: 22 septiembre 2019].

[45] P. S. Harry Kornilakis, «Crew Pairing Optimization with Genetic Algorithms,» 2002. [En línea]. Available: https://pdfs.semanticscholar.org/79b9/10cda58e34657b106279a903ba46b6c26e24.pdf. [Último acceso: 22 septiembre 2019].

[46] M. S. Rasmussen, R. M. Lusby, D. Ryan y J. Larsen, «Subsequence Generation for the Airline Crew Pairing Problem,» 2011. [En línea]. Available: https://orbit.dtu.dk/files/5760507/rap9(2).11.pdf. [Último acceso: 22 septiembre 2019].

[47] Jeppesen, «Carmen Crew Pairing,» [En línea]. Available: http://ww1.jeppesen.com/documents/aviation/commercial/Pairing.pdf. [Último acceso: 22 septiembre 2019].

[48] F. S. Roy E. Marsten, «Exact Solution of Crew Scheduling Problems Using the Set Partitioning Model: Recent Successful Applications,» mayo 1979. [En línea]. Available: https://www.gsb.stanford.edu/faculty-research/working-papers/exact-solution-crew-scheduling-problems-using-set-partitioning-model. [Último acceso: 23 septiembre 2019].

[49] M. L. G. Villar, «Historia y características,» de *Informática. Temario A. Volumen Ii. Profesores de Educación Secundaria E-book*, MAD, España, 2004, p. 278.

[50] ISRD Group, «Overview of C language,» de *Programing and Problem Solving C*, Mc-Graw Hill, Nueva Dehli, 2008, p. 32.

[51] J. I. Carmona, «Why are several popular programing languages influenced by C,» Software Engineering, 17 febrero 2012. [En línea]. Available: https://softwareengineering.stackexchange.com/questions/135544/why-are-several-popular-programming-languages-influenced-by-c. [Último acceso: 11 octubre 2019].

[52] The Code::Blocks Team, «Code::Blocks,» 30 diciembre 2017. [En línea]. Available: http://www.codeblocks.org/. [Último acceso: 11 octubre 2019].

[53] B. Coppin, «Combinatorial Explosion,» de *Artificial Intelligence Illuminated*, Londres, Jones and Bartlett Publishers, 2004, p. 57.