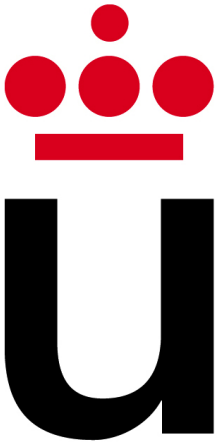


Universidad Rey Juan Carlos

Escuela Técnica Superior Ingeniería Informática

XXXXXX XXXX

TFG

GIS - Carlos Vázquez Sánchez

Móstoles - 27 de marzo de 2017

Índice general

Índice de figuras

Capítulo 1

Introducción y nuevos objetivos

1.1. Introducción

Introducción contando la stuación actual

1.2. Versiones anteriores

Este Trabajo Final de Grado es la continuación del trabajo que llevaron a cabo Diego Ruiz Aguado y Gonzalo Quevedo García en 2012 en sus Proyectos Finales de Carrera, los cuales se apoyaron a su vez en la Tesis Doctoral de Alba Agustín Martín (2011).

A continuación se hace una breve descripción del trabajo de Diego Ruiz Aguado y Gonzalo Quevedo García:

1. Los datos del problema se encontraban en una base de datos no relacional con redundancias. El primer paso consistió en migrar esta base de datos no relacional a una base de datos MySQL relacional y bien estructurada.

2. Para obtener los datos que necesitaba el problema, se realizó un programa en JAVA que se conectaba a la BBDD y creaba varios ficheros .txt en la que se volcaba toda la información necesaria para el posterior modelado del problema.

3. A continuación, el programa en java leía estos ficheros .txt y creaba las estructuras de datos necesarias(árbol de rutas, vuelos, wapoints, etc).

4. Posteriormente una subrutina en C se encargaba de definir un problema de CIPLEX con la función objetivo y las restricciones necesarias.

5. Finalmente, se ejecutaba el problema de optimización mediante la librería CIPLEX para obtener la mejor solución del problema.

1.3. Nuevos objetivos

La versión anterior del problema adolecía de un importante inconveniente: no podía salir de los máximos locales, ya que el heurístico que se utilizaba para lanzar los vuelos era un algoritmo voraz.

De esta forma, el resultado del problema dependía en gran medida del orden en que se intentara encontrar una solución para cada vuelo.

Por tanto los objetivos marcados para este Trabajo de Fin de Grado han sido los siguientes (ordenados en decreciente prioridad):

1. **Mejorar heurístico:** el objetivo principal de este TFG consiste en sustituir el algoritmo voraz por un heurístico que permita al problema escapar de los máximos locales, y por tanto encontraqr una solución mejor al problema.

2. **Desacoplar el programa de CIPLEX:** con la implementación de los nuevos heurísticos no es necesaria la librería de optimización. Se pasará de un sistema clásico de optimización (función objetivo y restricciones) a una estructura de objetos que permitan un manejo óptimo de las estructuras de datos durante la ejecución del algoritmo.

3. **Mejorar el sistema de lectura de datos:** la versión actual del programa crea ficheros .txt en los que se vuelca toda la A2 del problema (vuelos, waypoints, routas, etc) que pueden superar las 100.000 lineas. Estos ficheros auxiliares pueden sustituirse por ficheros mucho más pequeños en los que se exporta la A2 de la base de datos, y de forma interna el problema se encarga de crear las estructuras necesarias.

4. **Representación gráfica:** aunque estrictamente no aporta a mejorar la solución del problema, su representación gráfica puede ayudar a modelizar mejor el algoritmo, ya que permite visualizar de manera rápida y sencilla el estado del problema .

Capítulo 2

Descripción del problema

2.1. Introducción

Intro hablando de que es un problema clásico

1. explicar problema: nodos, rutas retrasos,
2. Restricciones
3 ejemplo vuelo pequeño con su arbol de rutas

El modelo se puede resumir como un problema de optimización en el que una serie de vuelos con unas rutas ....:XXXXXXXXXXXXXX

Las entidades por tanto que tenemos en el modelo son las siguientes:

2.2. Sectores

Representan las zonas del espacio en el que se encuentran los waypoints. Los sectores pueden tener waypoints en su zona interior o en la frontera, de modo que para pasar de un sector a otro habrá que atravesar siempre un waypoint. Los sectores tienen un límite de capacidad, de modo que para un instante e tiempo t solo pueden haber un número n de vuelos simultáneamente. De este modo, Si un vuelo tiene programada una ruta en un momento de tiempo que pasa por un sector que está al límite de su capacidad, esa ruta no será válida, por lo que tendrá que intentar retrasar su ruta, intentar una ruta alternativa, y si no tiene otra alternativa, cancelarse.
Para las pruebas que hemos realizado se ha considerado el escenario más restrictivo posible: la capacidad de un sector en cada instante de tiempo es 1.

2.3. Waypoints

Los waypoints representan puntos de ruta en las trayectorias de los vuelos. Como ya se ha explicado, un waypoint puede pertenecer a un sector o a varios, dependiendo de si es un waypoint interior o de frontera, respectivamente.

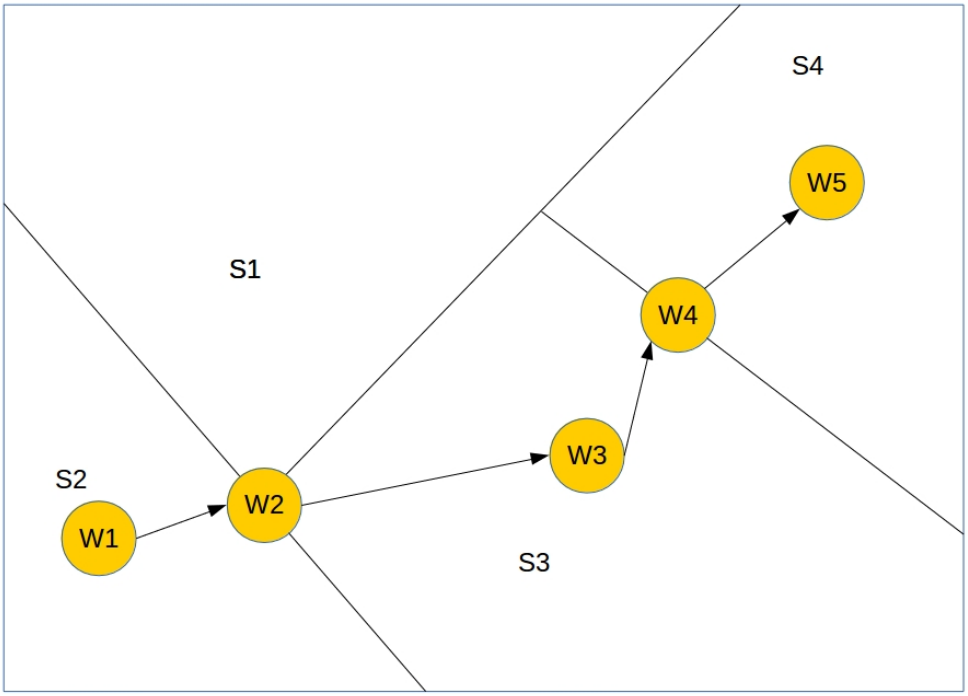


Figura 2.1: Ejemplo sectores y waypoints

2.4. Vuelos

Los vuelos son los elementos que tenemos que optimizar. Tienen una serie de rutas o trayectorias predefinidas, las cuales siempre empiezan y finalizan en un aeropuerto. Cada vuelo tiene una ruta predefinida, que será siempre igual o mejor (más rápida) que el resto de sus rutas. Tanto en los aeropuertos como en cada trayectoria entre waypoints, se permite un retraso máximo, de forma que entre 2 waypoints un vuelo puede recorrer esa distancia entre $[T_{mín}, T_{máx}]$ .Esto significa que un vuelo puede retrasar alguna de sus trayectorias entre 2 waypoints si el próximo sector al que va a acceder está sobrecargado o el arco por el que pretende pasar ya contiene otro vuelo.

Además, los vuelos tienen un coste de cancelación y están asociados a determinadas aerolíneas. En este problema consideramos que el todos los vuelos son iguales en importancia, por tanto el coste de cancelación de todos ellos es el mismo, independientemente de la longitud de su ruta, sectores que atraviesa, momento de despegue, etc.

Los vuelos a los que si se les encuentre solución, podrán ser de diferentes tipos:

- **Solución por defecto:** la solución del vuelo es la ruta inicial sin retrasos. Es el mejor resultado posible.

- **Retrasado:** se encuentra una solución factible en la ruta por defecto del vuelo, pero se ha producido un retrasos entre alguno de sus waypoints

- **Desviado:** la solución encontrada para el vuelo no es la ruta a priori. Puede ser an corta como la solución por defecto.

2.5. Rutas

Las rutas son el conjunto de trayectorias que tiene un vuelo para llegar desde su aeropuerto de origen al de destino. En el problema las representamos como un grafo ponderado y dirigido, en el que el coste de cada arista coincide con el intervalo de valores entre los cuales un vuelo puede hacer el trayecto entre 2 waypoints. Para modelar que los aeropuertos no tienen capacidad, creamos unos waypoints $aeropuerto'$ que simulan el waypoint en el que los vuelon aterrizan o despegan (estos waypoints sí que tienen las restricciones habituales del resto de waypoints).

Por ejemplo, un vuelo entre 2 aeropuertos con un waypoint entre ellos en el que se permita en cada trayectoria un retraso de 1, daría como resultado el siguiente vuelo:

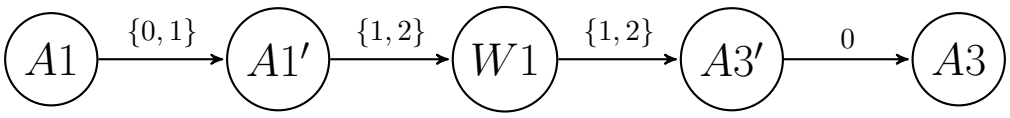


Figura 2.2: Ejemplo ruta de un vuelo

Para crear el grafo que represente las posibles rutas de este vuelo, tenemos que crear un nodo por cada posible waypoint en cada posible instante de tiempo t. De esta forma, tendremos un grafo con un conjunto de nodos de la forma W_t, siendo W el nombre del waypoint y t el instante de tiempo. De esta forma se obtiene que un vuelo puede estar en el mismo waypoint en diferentes modelos de tiempo (si se ha eleigo otra ruta más larga o se ha producido un retraso) en distintos momentos de tiempo. En el ejemplo anterior, la forma expandida del grafo sería (suponiendo que el vuelo despega en el instante $t = 0$):

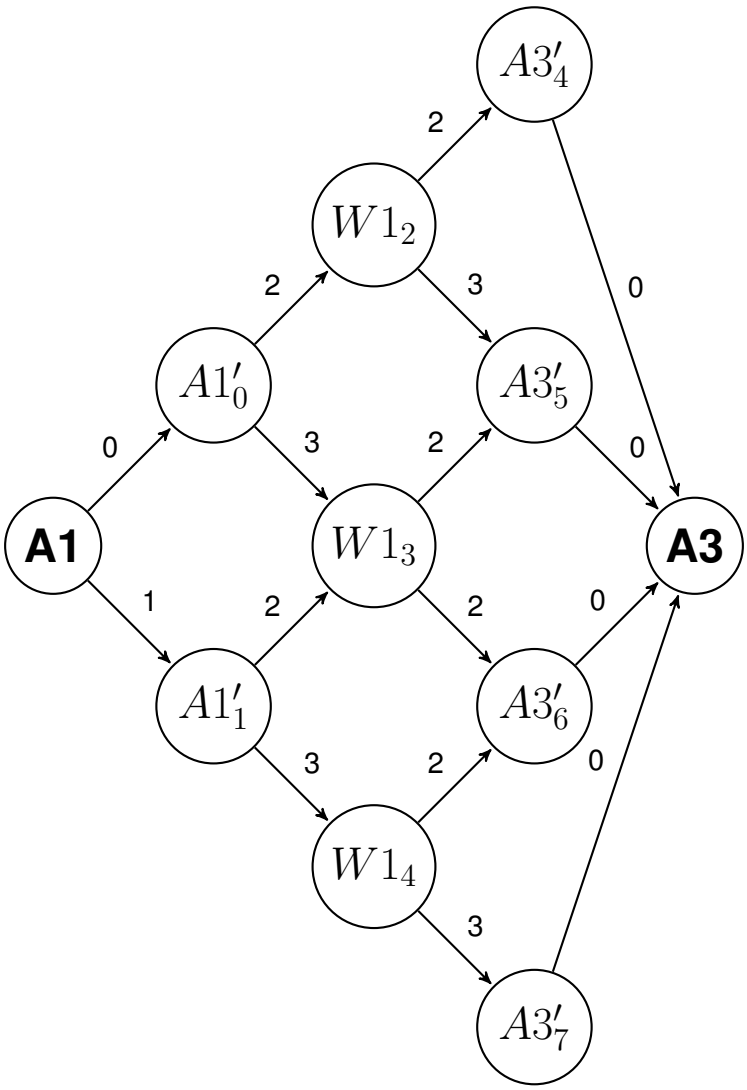


Figura 2.3: Ejemplo grafo de recorridos de un vuelo

Un vuelo puede además tener más de una ruta, de forma que si el ejemplo anterior tuviera una ruta adicional quedaría de la siguiente manera:

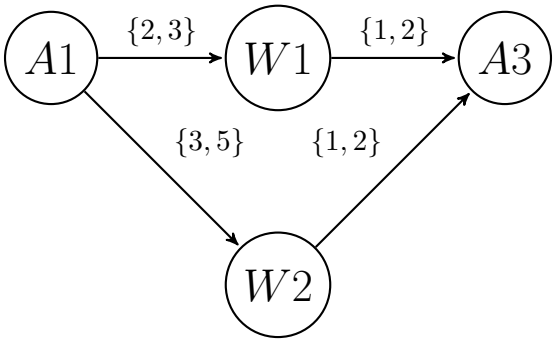


Figura 2.4: Ejemplo vuelo con 2 rutas

Su grafo de recorridos se compondría de forma análoga a los anteriores

2.6. Descripción del modelo

Por tanto, el modelo se puede expresar como

- **Función objetivo:** hay que maximizar el valor de la función objetivo. Para ello se ha creado un sistema de evaluación de vuelos con solución factible: los vuelos colocados en su solución a priori aportan 5 pts, los retrasados 3, los desviados en tiempo 2 y los desviados y retrasados 1. Este sencillo sistema se ideó para evaluar lo factible que es una solución, ya que como se explicó anteriormente no estamos utilizando el coste de cancelación (en ese caso se trataría de un problema de minimización de costes, en vez de maximizar la solución).
- **Restricciones:** serían tan sólo dos:
    1. No pueden haber 2 o más aviones en un mismo arco que conecta dos waypoints en el mismo instante de tiempo.
    2. No pueden haber 2 o más vuelos simultáneamente en el mismo sector.

Capítulo 3

Tecnologías utilizadas

Las tecnologías que se han utilizado a lo largo de este Trabajo de Fin de Grado han sido las siguientes:

3.1. C++

Todo el código ha sido desarrollado utilizando el lenguaje C++, creado por Bjarne Stroustrup en 1983. Sigue el paradigma de programación imperativa, y es considerado un lenjuaje orientado a objeto *híbrido* al ser una extensión del enguaje C. Desde los años 90 se ha mantenido como uno de los lenguajes más utilizados, y actualmente ocupa el puesto 3º en el rankin TIOBE.

Aunque se comenzó utilizando C en las primeras versiones del TFG, finalmente se optó por utilizar C++. Esto fue debido principalmente a 4 factores, que son además las ventajas de C++ sobre C:

- **Permite la orientación a objetos:** a la hora de modelar de nuevo el problema, era necesario crear una estructura de objetos que permitieran un fácil manejo de los datos, y características de la orientación a objetos como la herencia o los constructores permiten manejar la información de forma sencilla y estructurada.
- **Estructuras de datos:** aunque en C también existen, en C++ ya vienen implementadas como parte básica del lenguaje. El uso de estructuras como vectores, tuplas o mapas (y sus métodos) permiten implementaciones más sencillas y eficientes.
- **Sigue permitiendo un manejo de memoria a bajo nivel:** debido a la índole del problema este punto era muy importante, ya que un mal uso de la memoria podría hacer inviables problemas demasiado grandes o con muchas iteraciones.
- **Portabilidad:** el cambio de C a C++ es prácticamente inmediato, por lo que se pudo reaprovechar todo el trabajo realizado

3.2. MySQL

MySQL es un sistema relacional de gestión de base de datos creado en 1995 por Michael Widenius, y actualmente es uno de los 3 sistemas de BBDD más utilizados del mundo, junto a Oracle y Microsoft SQL Server.

Al igual que en la versión anterior del proyecto, la BBDD que usamos será la relacional que realizó Diego Ruiz Aguado en el 2012. En esta base de datos se encuentran todos los datos del problema: waypoints, rutas, aeropuertos, vuelos ... etc, los cuales serán utilizados para crear la estructura del problema.

3.3. HTML y JavaScript

HTML es un lenguaje de marcado creado por Tim Berners-Lee en 1991. Es el lenguaje más utilizado para la elaboración de páginas web, además de un estándar a cargo del consorcio WWW.

Java Script es un lenguaje de programaión interpretado, imperativo, dinámico, debilmente tipado y orientado a objetos. Es parte del estándar ECMAScript, soportado por la gran mayoria de los navegadores desde 2012, lo que lo convierte en el lenguaje más utilizado para el lado cliente en aplicaciones web.

Para la representación gráfica del problema se ha utilizado JS para leer y parsear la información almacenada en un fichero y HTML para su visualización (para la creación del grafo se ha utilizado la librería vis.js).

3.4. Git y Github

Git es un software de control de versiones distribuido creado por Linus Torvalds en 2005, cuyos dos mayores sistemas de hosting son Bitbucket y Github.

Se ha utilizado Git como sistema de versiones debido a las facilidades que otorga para trabajar desde distintos terminales, así como su fáil manejo de versiones . Todo el código se puede descargar y consultar en Github

Capítulo 4

Heurístico

El algoritmo heurístico que se ha creado para el problema está inspirado en el metahurístico *GRASP (Greedy Randomized Adaptive Search Procedure)*, sobre el que se han realizado una serie de modificaciones para adaptarlo al modelo en cuestión. A continuación se explicará el funcionamiento de un GRASP, para después explicar el algoritmo propuesto comparando sus diferencias y semajanzas.

4.1. GRASP

Un Grasp es un metaheurístico constructivo desarrollado inicialmente por T. Feo M. Resende, y se definie como ->cita libro Abraham<- *un GRASP es un procedimiento multiarranque en el que cada arranque se corresponde con una iteración. Cada iteracción tiene dos fases bien diferenciadas, la fase de construcción, que se encarga de obtener una solución factible de alta calidad; la fase de mejora, que se basa en la optimización (local) de la solución obtenida en la primera fase.*

4.1.1. Fase constructiva

La fase constructiva es un proceso iterativo en el que se construye una solución elemento a elemento.
Inicialmente se parte de un componente o conjunto de componentes que conforman una solución parcial, y no serán seleccionables. Posteriormente se ordenan los elementos seleccionables utilizando una función voraz (Greedy), la cual asignará a cada elemento un valor que indique como variaría la función objetivo si se añade a la solución parcial.
Una vez están ordenados todos los elementos seleccionables, hay que seleccionar qué elemento es añadido a la solución parcial. GRASP no añade el mejor candidato posible, ya que esto no garantiza que se obtenga la solución óptima, sino que se elije aleatoriamente un candidato del conjunto de candidatos restringido (o RCL por sus siglas en inglés). Para crear esta RCl, se crea un conjunto de candidatos que cumplan

$$RCL_{umbral} = (c_{min} + \alpha(c_{max} - c_{min})) \tag{4.1}$$

donde c_{min} y c_{max} son respectivamente los valores más bajo y más alto del coste asignado a los elementos seleccionables, y el parámetro :$\alpha : 0 \leq \alpha \leq 1$ determina el umbral permitido, de forma que se escoge un candidato aleatorio. Si fijamos $\alpha = 1$, en la RCL solo contaríamos con el mejor candidato, y sería una función miope pura. Por el contrario si fijamos $\alpha = 0$, serán seleccionados todos los

candidatos.

Una vez seleccionado el elemento o elementos, se introducen en la solución parcial y se les marca como no seleccionables. El resto de candidatos siguen siendo seleccionables, por tanto cuando se les vuelva a evaluar para ser candidatos, sus valores serán distintos a los de la iteración anterior, ya que la solución parcial ha variado al haber añadido el último elemento conjunto de candidatos.

La fase constructiva finaliza cuando se dispone de una solución factible.

4.1.2. Fase de mejora

Pero la fase constructiva no garantiza que la solución sea óptima respecto a su vecindad. Por ello GRASP incorpora la fase de mejora, que consiste en un procedimiento de optimización local, el cual puede ser otra metaheurística o una función de búsqueda local.

4.2. Heurístico implementado

4.2.1. Introducción

El algoritmo diseñado para este problema se resume en el siguiente pseducódigo:

```
inizializarProblema();
while N < iteracionesMáximas do
    añadirVuelosEnColaCandidatos();
    lanzarVuelosSoloSoluionesIniciales();
    intercambiarVuelos();
    lanzarVuelosPermitiendoRetrasos();
    lanzarVuelosPermitiendoDesvíos();
    buscarWaypointsSinUsar();
    retrasarVuelos();
    if N %númeroSolucionesExaminar == 0 then
        crearColaCandidatos();
    end
end
```

Algorithm 1: Esquema algoritmo implementado

Como se comentó anteriormente, al igual que un algoritmo GRASP tradicional, se compone de una fase constructiva en la que se obtiene una solución de alta calidad y una fase constructiva que se produce cada N iteraciones, en la cual se analizarán las solucions anteriores para seleccionar los vuelos más prometedores y añadirlos al problema, de forma que tengan más posibilidad de ser seleccionados antes. A continuación se explican con más detalle ambas fases.

4.2.2. Fase constructiva

Dado que este problema tiene siempre una solución factible (cancelar todos los vuelos), el objetivo de esta fase es conseguir una solución de alta calidad.

A continuación se detalla cada función que se lleva a cabo en la fase constructiva;:

1. **Añadir vuelos a la cola de candidatos:** es el primer paso de la fase constructiva, y solo se lleva a cabo si en la iteración anterior se ha creado una cola de candidatos. Esta función se encarga de generar de forma aleatoria el orden en el que se lanzarán los vuelos en el siguiente paso del algoritmo. Funciona de la siguiente manera:

   a) Si no hay cola de candidatos, se obtiene el id de cada vuelo del problema y se ordenan de forma aleatoria:

   $$\{1, 2, 3, 4, 5\} \Rightarrow \{2, 4, 5, 1, 3\} \tag{4.2}$$

   b) Si por el contrario si que tenemos una cola de candidatos, juntaremos el array de ids de los vuelos del problema junto a el array de candidatos. posteriormente se ordenan y se descartan los elementos repetidos, de forma que los elementos que estén repetidos tendrán más posibilidades de estar al principio del array:

   $$\begin{gathered}\{1, 2, 3, 4, 5\} + \{2, 4\} = \{1, 2, 3, 4, 5, 2, 4\} \\ \{1, 2, 3, 4, 5, 2, 4\} \Rightarrow \{2, 5, 4, 3, 1\}\end{gathered} \tag{4.3}$$

2. **Lanzar vuelos con las soluciones iniciales:** se lanzan todos los vuelos de manera aleatoria sin permitir retrasos o desvíos, la única ruta que se permite es la solución por defecto.
   Tras realizar varias pruebas, se ha podido comprobar que los pasos siguientes del algoritmo dependen en gran medida de los vuelos a los que se encuentra solución en esta fase. Esto se debe a que un "mal" vuelo colocado en su solución inicial puede sobrecargar sectores clave para otros muchos vuelos. En las pruebas que hemos realizado, en esta fase se colocan con éxito entre un 5 % y un 15 % de los vuelos.

3. **Intercambio de vuelos:** se intenta sustituir uno de los vuelos exitosos del paso anterior por 2 o más vuelos aleatorios a los que no se les halló solución.
   Este paso fue introducido para paliar el problema que se indicaba en el paso anterior, y se realiza partiendo de una idea básica: si cancelando un vuelo que teníamos con solución conseguimos colocar con éxito 2 o más vuelos, será siempre una mejora.

   Por ejemplo, si en la fase anterior fue colocado con éxito un vuelo de la siguiente forma

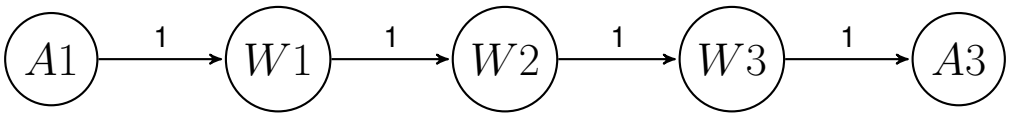
   

   Figura 4.1: Ejemplo vuelo colocado en la fase 2

   Podría cancelarse para colocar 2 vuelos más cortos:

Figura 4.2: Resultado tras el intercambio de vuelos

4. **Se intentan colocar vuelos permitiendo retrasos**: se lanzan aleatoriamente los vuelos que aun no tienen solución, permitiendo retrasos en sus rutas, pero no desvíos.
   En el problema se considera que un vuelo retrasado es preferible a un vuelo desviado, así que primero se intenta encontrar soluciones que no conlleven desvíos. En este paso se colocan entr un 50 % y un 65 % de los vuelos.

5. **Se intentan colocar vuelos permitiendo retrasos y desvíos**: se lanzan aleatoriamente los vuelos que aun no tienen solución, permitiendo retrasos y desvíos en sus rutas.
   En este punto si un vuelo tiene alguna solución factible, se le asignará. Se colocan con éxito entre el 20 % y el 30 %.

6. **Se buscan los waypoints sin usar y se les intenta asignar una ruta**: se localizan los waypoints por los que no pasa ningún vuelo a lo largo de todo el problema. Si algún vuelo tiene alguna solución factible que utilice alguno de estos waypoints, se la asigna.

7. **Retrasar vuelos con solución para colocar 2 o más cancelados**: se intenta retrasar alguno de los vuelos con solución factible para poder encontrar de forma aleatoria uno o más vuelos que estaban cancelados.
   Este paso se asa en el mismo concepto que en el intercambio de vuelos: si a costa de empeorar la solución de vuelo (un vuelo en su solución inicial o uno ya retrasado) se consigue encontrar la solución de 1 o más vuelos cancelados, se esta mejorando el resultado global.

4.2.3. Fase de mejora

Tras *N* iteraciones, se estudian las soluciones obtenidas y se selecciona de forma proporcional acorde a lo "buena"que haya sido la iteración, un número máximo de vuelos que se añadrán a la cola de vuelos adicionales. Aquí radica la mayor diferencia con un GRASP: en el metahurístico constructivo hay que evaluar y seleccionar de los candidatos disponibles a uno o varios de ellos según la ya citada fórmula

$$RCL_{umbral} = (c_{min} + \alpha(c_{max} - c_{min})) \tag{4.4}$$

la *RCL (Restricted Candidate List)*, para a continuación volver a evaluar a la lista de candidatos disponibles y repetir el proceso con la solución parcial actualizada.
En este algoritmo se analizan las iteraciones anteriores y se trata de extraer lo mejor de ellas. El proceso se realiza en 2 fases:

1. **Analizar las soluciones anteriores:** de las N iteraciones anteriores, calculamos el valor de la función objetivo para cada una de ellas para identificar así “como de buena ha sido”.

2. **Selección proporcional de candidatos:** una vez asignado un valor a cada iteración, se seleccionan vuelos que fueron colocados o en su solución inicial o retrasados muy poco de forma

proporcional al valor de la solución. El número máximo de vuelos que se seleccionan está marcado por el parámetro G.

Por ejemplo, si tenemos $N = 3$ y $G = 3$, y las soluciones a, b, c tienen valores de $100, 200, 300$ respectivamente, se escogerán vuelos de la solución c con una probabilidad de $1/6$, e la solución b $2/3$ y de la solución c con $3/6$

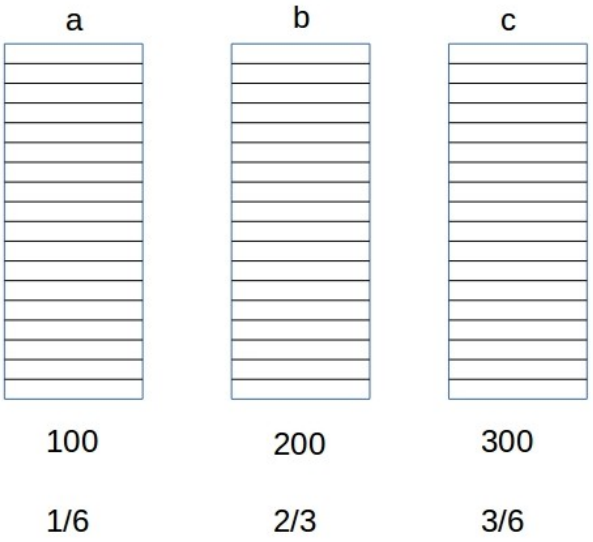


Figura 4.3: Elección ponderada de candidatos

3. **Creación de cola de vuelos extra:** estos vuelos conformarán la cola extra que se añadirá al conjunto de vuelos original, haciendo que éstos tengan mayor posibilidad de ser lanzados antes durante las siguientes *N* iteraciones, tal y como se indica en la fase constructiva:

4. **Descarte de mala solución** si la mejor de las N soluciones candidatas que se analizan no supera el valor de la mejor solución encontrada hasta el momento, se descarta la cola actual y se reinicia el problema. Si se da este caso, indicará que la última cola de candidatos que seleccionamos no ha conseguido mejorar la solución del problema, por tanto podemos descartarla.

4.2.4. Parámetros heurístico

Por tanto el heurístico depende de dos parámetros:

- N: cada cuantas iteraciones actualizamos la cola de vuelos adicional.
- G: el tamaño máximo de la cola de vuelos adicionales.

Tras realizar pruebas sobre distintos problemas, se ha obtenido que las mejores soluciones se tienen a alcanzar con un parámetro N pequeño y G grande, de forma que la cola de vuelos candidatos sea lo más grande posible y que la fase constructiva se realice con mucha frecuencia. Los resultados se pueden observar en la siguiente sección.

Cuadro 4.1: Comparativa de parámetros G y N. Ejemplo problema con 100 vuelos

	N=2, G=10	N=2, G=30	N=2, G=30
Problema 1	a b c		
Problema 2			
Problema 3			

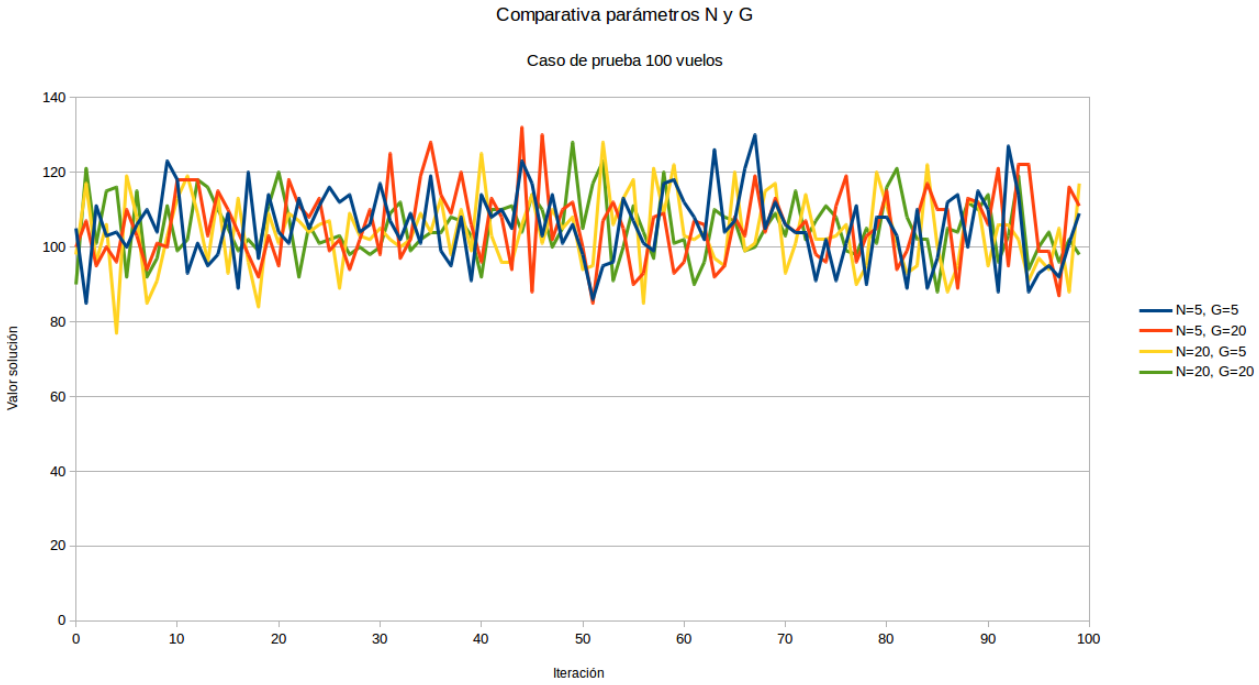


Figura 4.4: Comparativa de parámetros G y N

De esta forma, un alto parámetro G permitirá que muy probablemente muchos de los vuelos que en la mejor solución encontrada hasta ese momento fueron colocados con éxito, se lance, intentando así reproducir"la mejor solución que se había encontrado.
Por contra un bajo parámetro N hará que se cree cada pocas iteraciones una nueva cola de vuelos candidatos, lo que permitirá que en caso de no encontrase rápidamente una solución mejor, se elimine la cola actual y se reinicie el problema.
A continuación se muestran los datos obtenidos en una de las simulaciones:

Como se puede comprobar, esta combinación de parámetros es la que aporta más aleatoriedad en los resultados, pero es también de media la que aporta mejores resultados, lo que indica que se las malas soluciones son descartadas rápidamente pero se permite explorar un máximo local.

Cuadro 4.2: My caption

	N=2, G=10	N=2, G=30	N=2, G=50	N=5, G=10	N=5, G=30	N=5, G=50	N=10, G=10
Problema 1	Máx: 1 Med: 2 Desv: 3						
Problema 2							
Problema 3							

Capítulo 5

Resultados experimentales

A continuación se detallan los resultados obtenidos en distintos problemas. Los resultados corresponden a la media de 100 simulaciones con 100 iteraciones cada una. En estos datos además con la hipótesis más estricta de que la capacidad de un sector en un instante t es 1

5.1. Base de datos 1

5.1.1. 20 vuelos

Sobre el 90 %-100 %

5.1.2. 100 vuelos

Sobre el 60 %-70 %

5.1.3. 812 vuelos

SIN PROBAR

5.2. Base de datos 2

5.2.1. 20 vuelos

Sobre el 95 %-100 %

5.2.2. 100 vuelos

Sobre el 90 %

5.2.3. 3196 vuelos

SIN PROBAR

5.3. Base de datos 3

5.3.1. 20 vuelos

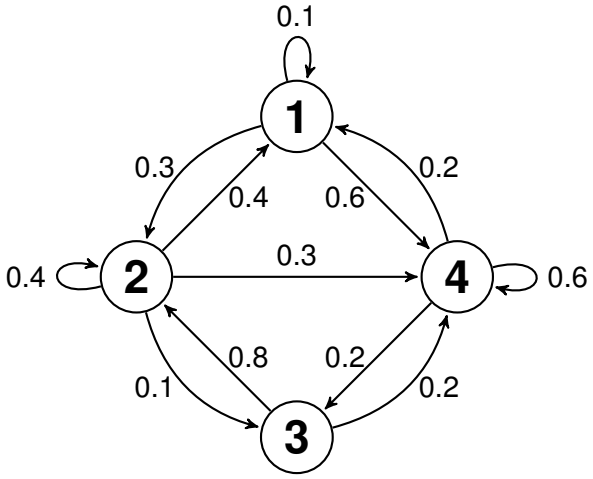
Sobre el 95 %-100 %

5.3.2. 100 vuelos

Sobre el 80 %

5.3.3. 6475 vuelos

SIN PROBAR



Capítulo 6

Conclusiones

6.1. Valoración de los resultados

Al final del Trabajo de Fin de Grado se han cumpido todos los objetivos que se establecieron en su momento:

- Modelar de nuevo el problema sin que éste dependiera de CIPLEX.
- Mejorar el sistema de lectura de datos.
- Mejora del heurístico para lanzar los vuelos.
- Creación de una simple representación visual del problema

Aunque se han cumplido todos los objetivos cumplidos, podemos comparar los resultados que se obtuvieron en la versión anterior del problema con los de este TFG:
En la versión de junio de 2012 con una base de datos de 65 vuelos y las capacidades de los sectores a 1, obtuvieron un rcentaje de vuelos cancelados de 11 %. Con el nuevo algoritmo lo reducimos a un 3 %

6.2. Futuras líneas de trabajo

Para enriquecer más el modelo, se podrían añadir algunas de las restricciones que se eliminaron para simplificar el modelo. La más importante sería la de añadir el coste de cancelación de un vuelo, de forma que a la hora de elegir entre 2 vuelos parejos se tenga en cuenta el coste de cancelación

Capítulo 7

Bibliografía