Universidad de La Laguna

SECCIÓN DE MATEMÁTICAS | FACULTAD DE CIENCIAS

Marta Hernández Hernández

Un problema de recogida y entrega de mercancía con costos dependiendo del tiempo de entrega

An unpaired pickup and delivery problem with time dependent assignment costs

Trabajo Fin de Grado
Grado en Matemáticas
La Laguna, Junio de 2017

Dirigido por

Hipólito Hernández Pérez

Hipólito Hernández Pérez
Departamento de Estadística e Investigación Operativa
Universidad de La Laguna
38271 La Laguna, Tenerife

Agradecimientos

A mi tutor Hipólito Hernández Pérez por guiarme en la realización de este trabajo.

A mi tutor de grado Fernando Pérez González por motivarme durante los años de carrera.

A mi familia y amigos por su incondicional apoyo.

Resumen · Abstract

Resumen

El problema de recogida y entrega de mercancía con costos dependiendo del tiempo de entrega, es un problema de optimización cuya aplicación busca el mínimo coste de llevar a cabo operaciones de organización y distribución por parte de empresas dedicadas al transporte de mercancías. Estas empresas son las encargadas de mobilizar la mercancía desde la ubicación de los clientes hasta los aeropuertos, y de la elección adecuada de itinerarios de vuelo para cada cargamento que posteriormente será transportado por vía aérea. La toma de estas decisiones se convierte entonces, en un problema de resolución compleja. En esta memoria, se estudia un modelo descrito en la literatura, y se buscan alternativas a éste con el objetivo de reducir sus tiempos de ejecución.

Palabras clave: *Recogida y entrega – Ruta de vehículos – Costos dependiendo del tiempo*

Abstract

The unpaired pickup and delivery problem with time dependent assignment costs is an optimization problem whose aplication seeks the minimum cost of carrying out operations of organization and distribution by companies engaged in freight transport. These companies are in charge of mobilizing the merchandise from the location of customers to the airports, and the appropriate choice of flight itineraries for each cargo that will later be transported by air. The taking of these decisions becomes a problem of complex resolution. In this report, a model described in the literature is studied, and alternatives are looked for to reduce its execution times.

Keywords: *Pickup and delivery – Route of vehicles – Time dependent costs*

Contenido

Introducción

En esta memoria, se analiza un problema que ha surgido recientemente, de recogida y entrega de la mercancía de clientes hasta aeropuertos, para su posterior transporte aéreo. La formulación del modelo asociado a este problema, minimiza los costos de generar las rutas adecuadas del transporte terrestre de las mercancías y la elección de los itinerarios de vuelos para cada cliente.

Con el paso de los años, el transporte por vía aérea ha adquirido mayor importancia debido al aumento global en las movilizaciones comerciales. Se sabe de la existencia de informes que predicen que este crecimiento en la industria de la aviación aún no ha finalizado, pues ahora, al haber más opciones de envío y a más localizaciones, esta forma de transporte es mucho más accesible.

La construcción de nuevos aeropuertos, la extensión de los ya existentes, la conversión de aeropuertos militares a comerciales y otros ejemplos, motivan la creación de lo que se denomina como *Multi-Airport regions*, regiones donde pueden encontrarse más de un aeropuerto para su uso comercial. La propuesta de la utilización de estas regiones multi-aeropuertos, supondría la reducción en los recorridos de vehículos comerciales, la disminución de la mano de obra y la mejora del nivel de servicio de entrega de las mercancías de distintas empresas en los aeropuertos, además de poder evitar el tráfico aéreo.

A las empresas que se dedican al transporte de mercancía desde los clientes hasta los aeropuertos, se les presenta entonces un problema de organización y administración de gran complejidad: la búsqueda de itinerarios de vuelos apropiados para cada cliente, la recogida de su mercancía correspondiente y la entrega a tiempo en el aeropuerto para su posterior transporte aéreo. Éste, es el problema que se presenta en el artículo de F. Azadian, A. Murat, R-B. Chinnam[1], donde se propone un modelo matemático en busca de su resolución óptima. A pesar de la posibilidad de utilizar un mismo itinerario de vuelo para la mercancía de más de un cliente, supone un problema la elección de las mejores rutas terrestres y los programas aéreos en busca de los itinerarios de vuelo más adecuados. En

2012, Azadian[2] propone el algoritmo con el que se consiguen los itinerarios de vuelos óptimo. Su investigación es necesaria para el cálculo de los costos que intervienen en el problema a estudiar.

Si fuera necesario, además de las reservas de los itinerarios, estas empresas también son las encargadas del alquiler de la flota necesaria de vehículos capaces de realizar la recogida y entrega de las mercancías. Los vehículos deben ser aprovechados en su totalidad, tanto es así, que si un vehículo tiene capacidad para la carga de más de un cliente, realizará también sus correspondientes transportes.

El beneficio de estas empresas dedicadas al transporte de mercancías, depende directamente de la correcta elección de estas decisiones, que en conjunto forman un complejo problema de organización. El objetivo de este proyecto, es estudiar el modelo propuesto por Azadian et al.[1], con el que pretende minimizar los costos de la distribución de los itinerarios de vuelos entre los clientes y la elección de las rutas de transporte por carretera con mayor rentabilidad.

Además se proponen variantes al modelo original en busca de tiempos de ejecución del problema más cortos que beneficien su resolución. Se aplicarán estos modelos a un conjunto de instancias y finalmente se compararán los resultados de forma computacional mediante un software específico, para su posterior análisis y obtención de conclusiones.

1

Descripción y características del problema

Se procede a la descripción de los elementos y parámetros que darán paso a la formulación del problema propuesto por Azadian et al.[1], *An unpaired pickup and delivery problem with time dependent assignment costs: Application in air cargo transportation* (Un problema de recogida y entrega de mercancía con tiempo dependiente de la asignación de costos: Aplicación en el transporte aéreo), en adelante ATD-PDP. Su objetivo es optimizar los recursos en el transporte de las mercancías de ciertos clientes hasta los aeropuertos, buscando las rutas terrestres y los itinerarios de vuelos con menor coste posible.

1.1. Descripción

El problema trata de elaborar un plan operacional para la recogida y entrega de las mercancías de los clientes, que están geográficamente dispersos, mediante una flota de vehículos, y la entrega a distintos aeropuertos en una región basándose en las opciones de vuelo para cada cliente. Se plantean entonces dos conjuntos de decisiones:

- Decisiones para seleccionar los vuelos correspondientes en los que se transportará la mercancía de cada cliente.
- Decisiones para la obtención de las rutas que realizarán los vehículos para la recogida y entrega de la mercancía.

El objetivo de este problema, es minimizar el costo total, en el que intervienen los costos de los vuelos, los costos del transporte por carretera y posibles sanciones debido a la tardanza en llegar al destino.

Existirán restricciones denominadas *ventanas de tiempo*, debido al corto tiempo entre los horarios de salida de los vuelos, en los que se fijará un intervalo de tiempo para completar las entregas. Se asume que el número de vehículos idénticos que intervienen es fijo y que no existen restricciones de capacidad,

pudiendo ser no necesaria la utilización de todos ellos. Además, no se consideran divisiones entre la recogida y la entrega de mercancía ya que no supone ninguna variación en el transporte aéreo posterior. Es decir, la mercancía recogida de un cliente, o la entrega de ésta en un aeropuerto, es cargada por un mismo vehículo y transportada de una vez.

1.2. Notación utilizada

Se considera el grafo dirigido

$$G_0 = (V_0, E_0)$$

donde V_0 es el conjunto de vértices y E_0 es el conjunto de los arcos. El conjunto de vértices incluye las localizaciones de los clientes, del depósito, y de los aeropuertos

$$V_0 = \{d\} \cup C \cup H$$

siendo d el depósito, C el conjunto de clientes y H el conjunto de aeropuertos.

El depósito se mantendrá abierto durante un tiempo prefijado con una hora de apertura θ_{op} y una hora de clausura θ_{cl}, del que saldrán los vehículos, cuyo conjunto se denomina K. Los vehículos deberán volver al depósito antes de su cierre.

Serán conocidos el costo c_{ij} y el tiempo t_{ij} del recorrido del arco $(i, j) \in E_0, i \neq j$, con $c_{ij} \geq 0$ y $t_{ij} \geq 0$. Se asume que los costos y los tiempos verifican la desigualdad triangular, esto es

$$t_{ij} + t_{jl} \geq t_{il} \quad , \quad c_{ij} + c_{jl} \geq c_{il} \qquad \forall i, j, l \in V_0$$

En el caso de existir, se denominará c_{kj} al costo que produciría la utilización del vehículo $k \in K$ cuyo destino fuese $j \in C \cup H$. El tiempo de carga y descarga es despreciable y no habrán ventanas de tiempo durante la recogida de la mercancía.

Cada cliente tendrá diferentes opciones de vuelos con los que transportar su mercancía, según los itinerarios ofrecidos por los aeropuertos. Se considera R_h al conjunto de los vuelos disponibles del aeropuerto $h \in H$ por cada día de operación y F_{ir}^h el costo asociado al vuelo $r \in R_h$ del cliente $i \in C$, que incluye el costo del transporte de la mercancía y posibles penalizaciones por la tardanza. El tiempo en el que comienza el vuelo $r \in R_h$ es denotado por Q_r^h, y se verifica que $Q_r^h \geq \theta_{op}$, es decir, los vuelos no pueden partir antes de que abra el depósito.

Es posible que más de un cliente utilice el mismo vuelo. Para estos vuelos compartidos, coincide el tiempo Q_r^h, pero cada cliente mantiene su costo F_{ir}^h. Además, si a un cliente no se le puede asociar un vuelo disponible, la opción se desecha, es decir, si un vuelo no forma parte de una solución factible, podrá ser eliminado del modelo.

En la tabla 1.1, se resume la notación que será utilizada.

Resumen de la notación utilizada	
C	Conjunto de clientes
H	Conjunto de aeropuertos
d	Depósito
K	Conjunto de vehículos
R_h	Conjunto de vuelos asociados al aeropuerto $h \in H$
V_0	Conjunto de vértices: $V_0 = \{d\} \cup C \cup H$
θ_{op}	Tiempo de apertura del depósito
θ_{cl}	Tiempo de clausura del depósito
c_{ij}	Costo por recorrer el arco (i, j)
t_{ij}	Tiempo en recorrer el arco (i, j)
F_{ir}^h	Costo asociado al vuelo $r \in R_h$ para el cliente $i \in C$
Q_r^h	Tiempo en que comienza el vuelo $r \in R_h$

Tabla 1.1. Notación para la formulación

1.3. Costos dependientes del tiempo de entrega

En el trabajo de Azadian et al.[1], se hace un estudio de las características de este problema que serán resumidas a lo largo de esta sección.

Para que sea factible la asignación de los vuelos para cada cliente $i \in C$, el vehículo que transporta su mercancía deberá llegar al aeropuerto no más tarde de la hora en que comienza su vuelo asignado, esto es, $t \leq Q_r^h$. Si no hubiesen vuelos disponibles cuando el vehículo llega al aeropuerto, $t > \text{máx}_{r \in R_h}\{Q_r^h\}$, se cambiará el itinerario de vuelo, por ejemplo, a un vuelo $r_0 \notin R_h$ al día siguiente, lo que supondría una penalización económinca $F_{i0}^h > F_{ir}^h$.

Es por ello, por lo que se define una función del tiempo dependiente del costo de entrega al aeropuerto h para la carga del cliente i, que se expresa como sigue:

$$f(h,i,t) = \begin{cases} min_{r \in R_h}\{F_{ir}^h | t \leq Q_r^h\}, & \text{si } t \leq max_{r \in R_h}\{Q_r^h\} \\ F_{i0}^h, & \text{en otro caso} \end{cases} \tag{1.1}$$

La definición de esta función, no implica la consideración de todas las opciones de vuelo disponibles para un cliente. Podemos excluir del conjunto de itinerarios, aquellos vuelos que no sean capaces de formar parte de una solución factible, y también de los denominados vuelos *dominados*. Esto es, dados $r, r' \in R_h$ dos itinerarios de vuelos, el itinerario r es dominado por el itinerario r' si:

$$\text{(a) } F_{ir'}^h \leq F_{ir}^h \text{ y } Q_r^h < Q_r'^h.$$
$$\text{(b) } F_{ir'}^h < F_{ir}^h \text{ y } Q_r^h \leq Q_r'^h.$$

Además, si $F_{ir'}^h = F_{ir}^h$ o $Q_r^h = Q_{r'}^h$, cualquiera de los dos se puede considerar dominado del otro.

Una vez sean eliminados los vuelos dominados, no existirán más de un vuelo por cliente y aeropuerto con el mismo costo o el mismo tiempo. La función del coste de entrega del aeropuerto $f(h, i, t)$ basada en los vuelos no dominados, será entonces, una función escalonada y no decreciente con discontinuidades en cada Q_r^h, $\forall r \in R_h$.

El ejemplo que sigue, muestra la utilización de la función $f(h, i, t)$, para un sólo aeropuerto y dos clientes.

Ejemplo 1.1. Se supone un aeropuerto $h \in H$ y dos clientes $i, j \in C$, con disponibilidad de dos itinerarios de vuelos $r = 1, 2$. El cliente i, podría utilizar ambos vuelos, pero para el cliente j, sólo está disponible el primer vuelo ya que su mercancía no tiene como destino el que proporciona el vuelo $r = 2$. Las funciones $f(h, i, t)$ y $f(h, j, t)$, son representadas como sigue:

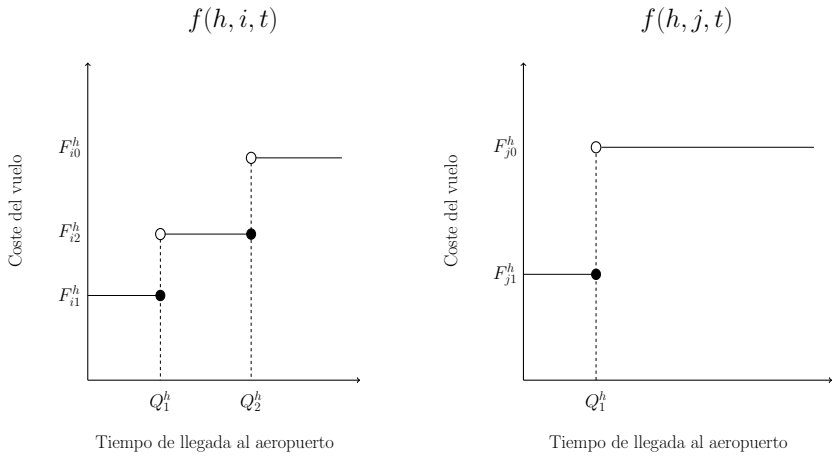

Figura 1.1. Función dependiente del costo

1.4. Grafo transformado

El camino óptimo del ATD-PDP, no siempre será *hamiltoniano*[*], pues cada nodo puede visitarse en más de una ocasión. Luego, para modelar el problema, será necesario seguir el orden de éstas visitas para cada vehículo introduciendo variables adicionales con las que se podrán manejar las características de la función del coste de entrega del aeropuerto. Para evitar estos contratiempos, se realiza una modificación del grafo original $G_0 = \{V_0, E_0\}$, el cual será denominado en adelante como *grafo transformado* y será representado como $G = \{V, E\}$.

En el ATD-PDP, habrá una solución óptima donde todos los clientes son visitados como mucho una vez, lo cual no implica que los aeropuertos también sean visitado a lo sumo en una ocasión. Sin embargo, la aplicación del grafo transformado asegura que existe una solución óptima del problema donde cada vehículo entrega la mercancía de los clientes en el aeropuerto para cada itinerario de vuelo una sola vez. Concretamente, cualquier aeropuerto $h \in H$ es visitado como mucho $|R_h| + 1$ veces por cada vehículo, utilizando itinerarios en días posteriores si así se requiere.

En la transformación del grafo, se divide el vértice de cada aeropuerto h en $|R_h| + 1$ nodos, que serán denominados *nodos-vuelos*, y que hacen referencia individualmente a cada itinerario. La notación utilizada anteriormente es modificada, pues de tratar con los conjunto H y R_h, se pasa a tratar directamente con el conjunto R de vuelos cuyo tamaño será

$$|R| = \sum_{h \in H} |R_h| + |H|$$

En la tabla 1.2 se resumen los cambios en la notación utilizada.

$G_0 = \{V_0, E_0\}$	$G = \{V, E\}$	**Descripción**
Q_r^h	Q_r	Tiempo en que comienza el vuelo $r \in R$
F_{ir}^h	F_{ir}	Costo para el cliente $i \in C$ por utilizar el itinerario $r \in R$

Tabla 1.2. Notación para la utilización del grafo transformado

El grafo transformado hereda el conjunto de arcos del grafo original que inicialmente conectaban el depósito con los clientes y los clientes entre ellos. Cada circuito formado por la ruta que realiza un vehículo, visitará cada vértice a lo sumo una vez en el grafo transformado, y la solución factible que se obtenga en el nuevo grafo, corresponderá a una del grafo $G_0 = \{V_0, E_0\}$.

[*] Sucesión de arcos adyacentes de un grafo que visita todos y cada uno de los nodos exactamente una vez.

Ejemplo 1.2. Sea el grafo $G_0 = \{V_0, E_0\}$ asociado a un problema ATD-PDP en el que interviene un único depósito del que parten dos vehículos, dos aeropuertos con dos itinerarios de vuelo disponibles y cinco clientes.

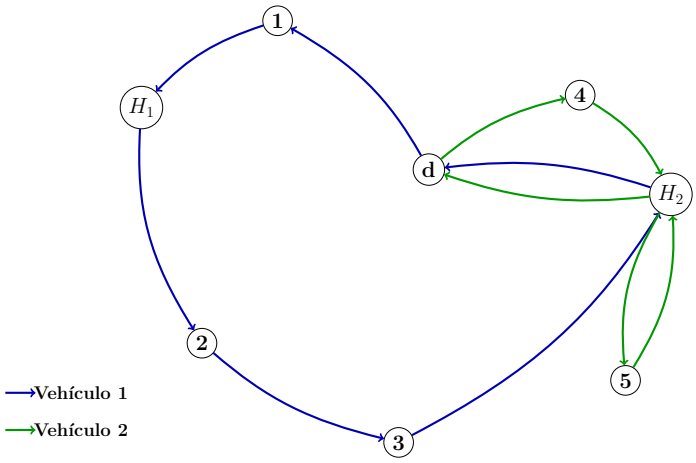

Figura 1.2. Grafo original

Se observa en la figura 1.2 como, partiendo del depósito, el vehículo 1 es responsable de transportar las mercancías del cliente 1 al aeropuerto H_1 y a continuación las mercancías de los clientes 2 y 3 al aeropuerto H_2. A su vez, el vehículo 2, abandona el depósito para transportar desde la localización del cliente 4 al aeropuerto H_2 su mercancía, para posteriormente recoger la del cliente 5 y volver a éste aeropuerto. Una vez ambos vehículos han concluido sus respectivas rutas, regresan al depósito. En esta solución se observa que el aeropuerto H_2 es visitado una vez por el primer vehículo y dos veces más por el segundo.

Al tomar cada itinerario de vuelo como un nodo independiente del grafo, se consigue que cada nodo sea visitado a lo sumo una sola vez por cada vehículo, como se observa en la figura 1.3. Se asocia la mercancía del cliente 1 al segundo vuelo r_2, que sigue siendo transportada por el vehículo 1. Éste mismo vehículo visita posteriormente a los clientes 2 y 3, cuyos vuelos asociados es para ambos r_3. El vehículo 2 por su parte, se encarga de transportar hasta el vuelo r_3 la carga del cliente 4, para después visitar al cliente 5 y movilizar su mercancía

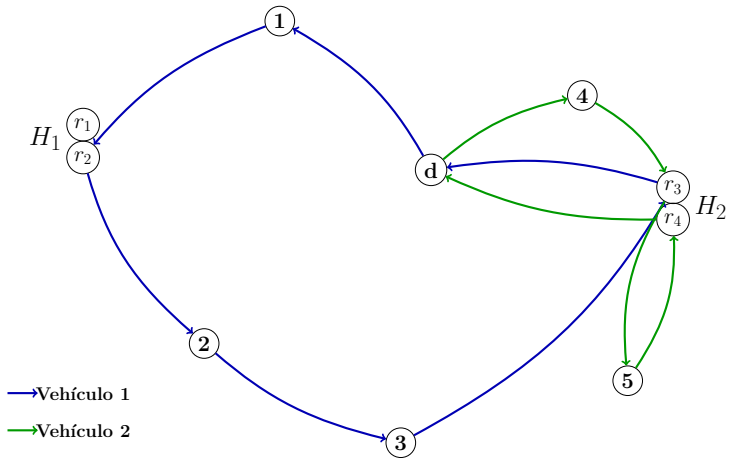

Figura 1.3. Grafo transformado

hasta el vuelo r_4. De esta forma, el circuito realizado por cada vehículo visita cada nodo a lo sumo una vez, y la solución, tanto para el grafo original como para el transformado, es la misma.

2

Formulación del problema

En este capítulo se incluye la formulación del ATD-PDP propuesta por Azadian et al.[1]. Se proponen además, variantes a este modelo con modificaciones en la notación y restricciones utilizadas.

2.1. Modelo con tres índices

El objetivo principal de este problema, es la optimización global de los costos. El funcionamiento del ATD-PDP consiste en la recogida de las mercancías de los clientes, la asignación de los intinerarios de vuelos para el transporte de dichas cargas, y su entrega a tiempo en los aeropuertos.

En la tabla 1.1 se muestra la notación utilizada en este problema. Sin embargo, al utilizar el grafo transformado, la notación varía ligeramente. En la tabla 2.1 se observa la notación para el grafo transformado que se utilizará a lo largo de este capítulo.

2.1.1. Modelo con un primer tipo de desigualdades de simetría

Se define una primera variable binaria de decisión que indica cuándo un vehículo utiliza un determinado arco del grafo transformado:

$$x_{ij}^{k} = \begin{cases} 1 & \text{si el vehículo } k \in K \text{ utiliza el arco } \big(i,j\big) \in E \\ 0 & \text{en otro caso} \end{cases}$$

La segunda variable de decisión binaria informa sobre qué vehículo transporta la mercancía de un cliente y cuál es su itinerario de destino:

$$y_{ir}^{k} = \begin{cases} 1 & \text{si el vehículo } k \in K \text{ envía la carga del} \\ & \text{cliente } i \in C \text{ por el vuelo } r \in R \\ 0 & \text{en otro caso} \end{cases}$$

Resumen de la notación utilizada	
C	Conjunto de clientes
H	Conjunto de aeropuertos
d	Depósito
K	Conjunto de vehículos
R	Conjunto de vuelos disponibles en el grafo transformado
V	Conjunto de vértices: $V = \{d\} \cup C \cup R$
θ_{op}	Tiempo de apertura del depósito
θ_{cl}	Tiempo de clausura del depósito
c_{ij}	Costo por recorrer el arco (i, j)
t_{ij}	Tiempo en recorrer el arco (i, j)
F_{ir}	Costo asociado al vuelo $r \in R$ para el cliente $i \in C$
Q_r	Tiempo en que comienza el vuelo $r \in R$

Tabla 2.1. Notación con la aplicación del grafo transformado

Una última variable continua determina el tiempo de llegada de cada vehículo a cada vértice

$$a_j^k = \text{tiempo de llegada del vehículo } k \in K \text{ al nodo } j \in V$$

Teniendo en cuenta cada costo representativo en el beneficio de la operación, el problema finalmente queda expresado de la siguiente forma:

$$\text{mín} \sum_{k \in K} \left[\sum_{i \in V} \sum_{j \in V \setminus \{i\}} c_{ij} x_{ij}^k + \sum_{i \in C} \sum_{r \in R} F_{ir} y_{ir}^k \right] \tag{2.1}$$

sujeto a:

$$\sum_{k\in K}\sum_{r\in R} y_{ir}^k = 1 \qquad \forall i \in C \tag{2.2}$$

$$\sum_{j\in V\setminus\{i\}} x_{ij}^k \leq 1 \qquad \forall i \in V, \forall k \in K \tag{2.3}$$

$$\sum_{j\in V\setminus\{i\}} x_{ij}^k = \sum_{j\in V\setminus\{i\}} x_{ji}^k \qquad \forall j \in V, \forall k \in K \tag{2.4}$$

$$a_i^k \geq \theta_{op} + t_{di} \qquad \forall i \in C, \forall k \in K \tag{2.5}$$

$$a_r^k \leq \min\{\theta_{cl} - t_{rd}, Q_r\} \qquad \forall r \in R, \forall k \in K \tag{2.6}$$

$$a_i^k \leq \overline{a}_i = \max_{r\in R}\{\min\{Q_r, \theta_{cl} - t_{rd}\} - t_{ir}\} \qquad \forall i \in C, \forall k \in K \tag{2.7}$$

$$a_r^k \leq \underline{a}_r^k = \theta_{op} + \min\{t_{di} + t_{ir}\} \qquad \forall r \in R, \forall k \in K \tag{2.8}$$

$$x_{ji}^k\big(M_{ij} - t_{ij} - t_{ji}\big) + \big(x_{ij}^k - 1\big)M_{ij} + a_i^k + t_{ij} \leq a_j^k \qquad \forall i \in V, \; \forall j \in V\setminus\{i, d\} \tag{2.9}$$

$$\big(y_{ir}^k - 1\big)M_{ir} + a_i^k + t_{ir} \leq a_r^k \qquad \forall i \in C, \forall r \in R, \; \forall k \in K \tag{2.10}$$

$$y_{ir}^k \leq \sum_{j\in V\setminus\{i\}} x_i^k j \qquad \forall i \in C, \forall r \in R, \; \forall k \in K \tag{2.11}$$

$$y_{ir}^k \leq \sum_{j\in V\setminus\{r\}} x_r^k j \qquad \forall i \in C, \forall r \in R, \; \forall k \in K \tag{2.12}$$

$$\sum_{j\in V\setminus\{i\}} x_{ij}^k = \sum_{r\in R} y_{ir}^k \qquad \forall i \in C, \forall k \in K \tag{2.13}$$

$$\sum_{i\in V\setminus\{r\}} x_{ir}^k \leq \sum_{i\in C} y_{ir}^k \qquad \forall r \in R, \forall k \in K \tag{2.14}$$

$$\sum_{i\in V}\sum_{j\in V\setminus\{i\}} c_{ij}x_{ij}^k \leq \sum_{i\in V}\sum_{j\in V\setminus\{i\}} c_{ij}x_{ij}^{k+1} \qquad \forall k = 1, .., |K| - 1 \tag{2.15}$$

$$y_{ir}^k, x_{ij}^k \in \{0, 1\}, a_i^k \geq 0 \qquad \forall i, j \in V, i \neq j \; \forall k \in K \tag{2.16}$$

Con el conjunto de restricciones (2.2), se asegura que cada cliente tiene asignado un único vehículo que transportará su mercancía a un único vuelo. De cada nodo saldrá a lo sumo un arco por cada vehículo que los visite, lo que queda reflejado con (2.3). Asimismo, las inecuaciones (2.4) mantienen la igualdad entre el número de arcos que llegan y salen de cada nodo, conservando así, la cantidad de vehículos que visitan cada cliente.

Puesto que los vehículos salen desde el depósito, el tiempo de llegada a los nodos será después de su apertura. Con (2.6), los nodos no pueden ser visitados por los vehículos después de la salida del vuelo que tengan asignado, en cuyo caso, el vehículo volverá al depósito antes de que éste cierre. A su vez, el conjunto de restricciones (2.7), determina que lo más tarde que un vehículo puede visitar un nodo, dependerá de la cantidad de nodos que haya visitado anteriormente. Lo más temprano que llegarían los vehículos a su destino, si el trayecto fuese directo desde el depósito, queda reflejado con el conjunto de restricciones (2.5).

Es evidente que el tiempo de llegada a cada nodo será menor que el de salida, luego para evitar posibles subciclos se añade el conjunto de restricciones (2.9) y con (2.10), las cargas deben ser recogidas antes de ser entregadas a los nodos-vuelos. En estas restricciones se define M_{ij} como:

$$M_{ij} = \overline{a}_i + t_{ij} \qquad \forall i \in V, \forall j \in V \backslash \{d, i\}, \forall k \in K$$

siendo

$$\overline{a}_i = \max_{r \in R} \{\min\{Q_r, \theta_{cl} - t_{rd}\} - t_{ir}\} \qquad \forall i \in V$$

La entrega y recogida es realizada por el mismo vehículo tanto en los nodos como en los aeropuertos, y esto queda reflejado con las restricciones (2.11) y (2.12). Las restricciones en (2.13) aseguran que cada vehículo visita a un cliente sólo si debe transportar su mercancía hasta algún nodo-vuelo.

Para evitar posibles simetrías y reforzar el rendimiento del problema, considerando que los vehículos son idénticos, se añaden las restricciones (2.15).

Por último, con (2.16), se añaden las restricciones que hacen referencia a las condiciones de las variables de decisión.

El problema que proponen Azadian et al.[1] no incluye el conjunto de desigualdades (2.14), que fueron añadidas durante el estudio del problema porque se observó que una solución factible del modelo podía formar subciclos, lo cual indicaba que las restricciones (2.9) no eran suficientes para evitar los subciclos. Se ve a continuación la solución a un ejemplo sencillo de forma gráfica.

Ejemplo 2.1. Se supone un problema que concierne a siete clientes, un aeropuerto con tres itinerarios posibles de vuelo y tres vehículos para el transporte de la mercancía de éstos clientes, cuyos localizaciones se muestran en la tabla 2.1 Las coordenadas de la localización del aeropuerto, han sido triplicadas una vez por cada itinerario de vuelo disponible, como se indica en la utilización del grafo transformado.

Aplicando el problema anterior sin las restricciones (2.14), se obtiene como solución, el grafo dirigido que se muestra en la figura 2.1*. Bajo las restricciones propuestas por Azadian et al.[1], se obtiene un valor objetivo de 1853'256, siendo

* Las coordenadas correspondientes al aeropuerto, han sido desviadas ligeramente para apreciar en el grafo los errores mencionados

Depósito		Clientes		Aeropuertos	
15	76	5	79	50	48
		25	32	50	48
		25	80	50	48
		71	20		
		68	26		
		90	70		
		3	72		

Tabla 2.2. Localizaciones del conjunto V

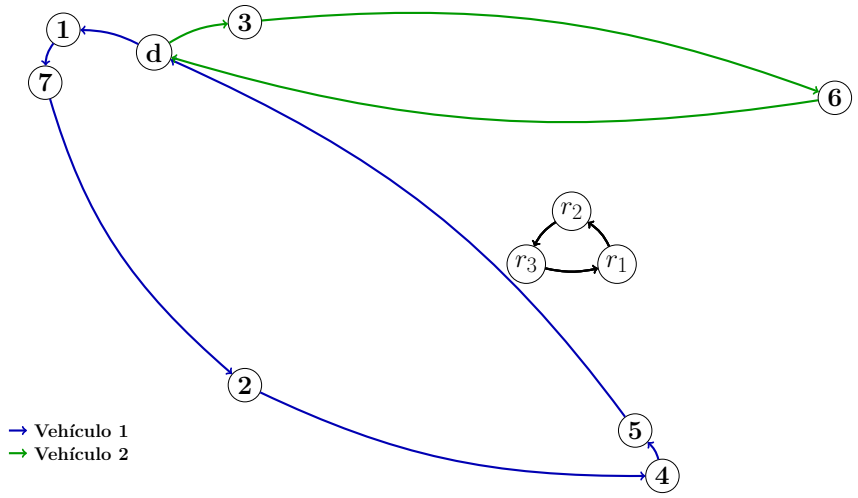

Figura 2.1. Solución al problema de 7 clientes, 1 aeropuerto y 3 vehículos

todos los nodos de los clientes visitados por los vehículos 1 o 2, pero sin llegar a transportar sus respectivas cargas hasta los nodos de los aeropuertos, los cuales son visitados por ambos vehículos en un continuo circuito. Esta solución no es factible, luego, se ha supuesto que las restricciones en (2.9) no son suficientes para el correcto funcionamiento del modelo.

Es por ello, que en este trabajo se ha propuesto añadir el conjunto de restricciones (2.14), reforzando así las condiciones para evitar subciclos en la solución.

Ejemplo 2.2. Para el anterior ejemplo 2.1, aplicando el problema con el conjunto de restricciones anterior, se obtiene una solución factible con valor objetivo 1868'874 y nuevamente, sólo dos vehículos son utilizados. En la figura 2.2, se muestra la inexistencia de subciclos.

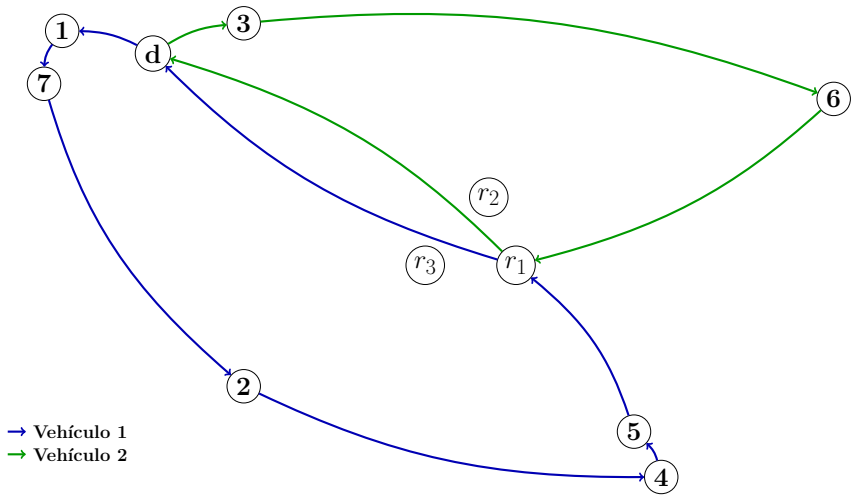

Figura 2.2. Solución al problema de 7 clientes, 1 aeropuerto y 3 vehículos, con las restricciones añadidas

Asimismo, inicialmente se aseguraba que los nodos del conjunto de vuelos, fuesen alcanzados en el menor tiempo posible, buscándose así el trayecto más corto para alcanzar el nodo-vuelo, con el conjunto de restricciones (2.8), las cuales, a la hora de la ejecución del modelo, no siempre producían una solución factible. En concreto, podría pasar que

$$\underline{a}_r > \overline{a}_i$$

lo que supone incoherencias entre los resultados. Finalmente se decidió incluirlas en el modelo salvo cuando se diera el caso en el que se obtuviese dicha condición.

2.1.2. Modelo con un segundo tipo de desigualdades de simetría

En un primer modelo pensado por Azadian et al.[1] para la mejora de los tiempos de ejecución del problema, el conjunto de restricciones (2.15) es

sustituído por:

$$\sum_{i \in C} 2^{|C|-i} \tau_i^k \leq \sum_{i \in C} 2^{|C|-i} \tau_i^{k+i} \qquad \forall k = 1, ..., |k| - 1 \tag{2.17}$$

siendo $\tau_i^k = \sum_{r \in R} y_{ir}^k$, la variable que cuenta las veces que el vehículo $k \in K$ destina la carga del cliente $i \in C$ al vuelo $r \in R$. Este nuevo conjunto de restricciones no depende de los parámetros de costo del problema, lo cual supone una ventaja a la hora de resolverlo.

El objetivo de esta modificación, es comparar ambos modelos para determinar cual se ejecuta en el menor tiempo.

2.2. Modelo con dos índices

En un intento de simplificar el modelo anterior disminuyendo el conjunto de variables utilizadas, se propone en este trabajo la eliminación del índice k asociado al conjunto de vehículos. Esto supone una reformulación del problema generalizando el conjunto K, a un conjunto de rutas sin importar el vehículo que las recorra. En este nuevo modelo, cada nodo se visita a lo sumo una vez, en total, por todos los vehículos.

Con el fin de proponer un modelo capaz de reducir los tiempos de ejecución en la resolución del problema, se procede a la duplicación de los nodos-vuelos, del mismo modo que para el modelo original se multiplicaron los nodos aeropuertos tantas veces como itinerarios de vuelo ofrecieran. Esto es que, para un máximo de m vehículos, se contempla la posibilidad de que cada uno de ellos sea capaz de tener como destino el mismo vuelo.

El objetivo es aplicar el mismo procedimiento que se siguió para la obtención del grafo transformado. Si bien la solución de éste es transferible al grafo original, se espera obtener los mismo resultados con esta variación en el modelo.

Duplicación de los nodos-vuelos

Se establece un nuevo parámetro m asociado al número de veces que un nodo-vuelo es visitado por los vehículos. Se realizan entonces, réplicas de los itinerarios de vuelo m veces, lo que conlleva una alteración en la notación utilizada en lo referido al conjunto R.

Se propone un conjunto R_1 como primera copia de los nodos-vuelo cuyo dominio es

$$R_1 = \{1, 2, 3, \ldots, |R|\}$$

Un segundo conjunto R_2 como una copia de los mismos con dominio

$$R_2 = \{|R| + 1, |R| + 2, |R| + 3, \ldots, 2|R|\}$$

De esta manera

$$R_m = \{(m-1)|R|+1, (m-1)|R|+2, (m-1)|R|+3, \ldots, m|R|\}$$

Si hasta ahora, $r \in R$ hacía referencia a los itinerarios de vuelos disponibles, con la adición del parámetro m, se puede utilizar

$$r \in R^*$$

donde R^* es el nuevo conjunto de nodos-vuelos en los que se incluye la duplicación de éstos. Es decir:

$$R^* = \bigcup_{l=1}^{m} R_l$$

En el ejemplo siguiente, se muestra gráficamente la aplicación de esta duplicación al grafo transformado para un máximo de dos vehículos capaces de visitar un mismo nodo-vuelo, es decir, para $m = 2$.

Ejemplo 2.3. Dadas las condiciones mencionadas en el ejemplo del grafo transformado 1.2, se procede a la duplicación de los nodos-vuelos bajo el mismo razonamiento.

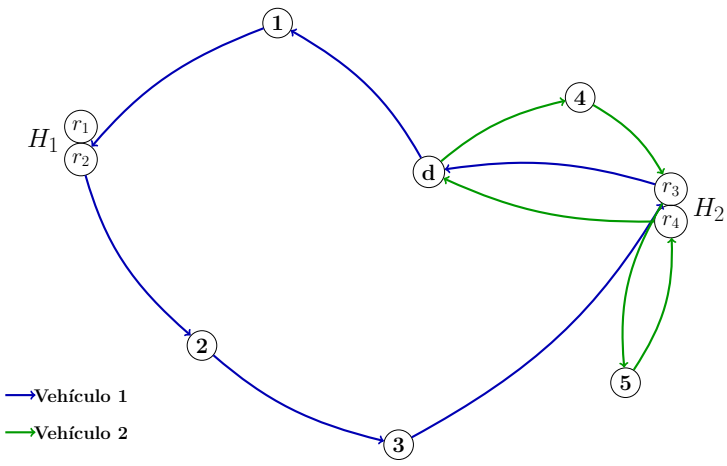

Figura 2.3. Grafo transformado

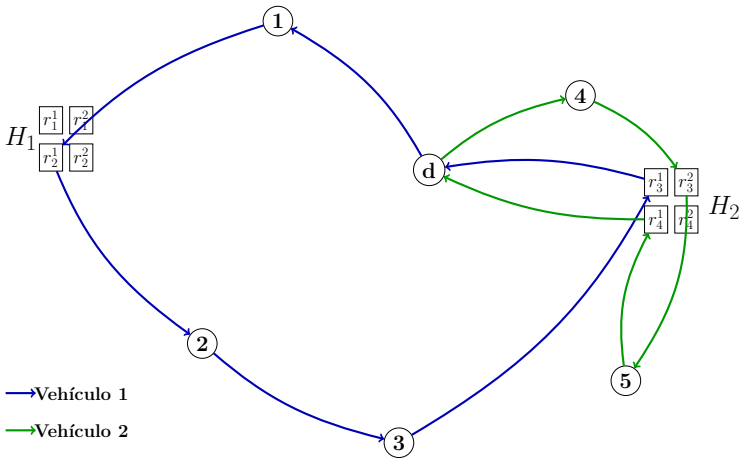

Figura 2.4. Grafo transformado con duplicación de los nodos-vuelos

Como se observa en la figura 2.3, tanto el vehículo 1 como el vehículo 2, son responsables de transportar las mercancías de todos los clientes hasta sus respectivos itinerarios de vuelo. Sin embargo, sólo el nodo-vuelo r_3, es visitado por ambos.

Al realizar la duplicación de los itinerarios de vuelos, se consigue, como se muestra en la figura 2.4, que ningún nodo sea visitado por más de un vehículo en más de una ocasión. Esto supone la creación de rutas en las que solo tienen como nodo común, el nodo asociado al depósito, tanto en la partida de la flota de vehículos, como a la vuelta de los mismos. Sin embargo, esto no siempre podría ser cierto, ya que en el caso de haber más de dos vehículo, sería posible que, a pesar de esta duplicación en los nodos-vuelos, más de un vehículo tenga como destino el mismo itinerario de vuelo. Aún así, la ejecución de este modelo, podría disminuir el tiempo en la resolución del problema.

Al excluir de la notación el índice k, las variables de decisión quedan definidas de la siguiente manera:

$$x_{ij} = \begin{cases} 1 & \text{si el arco } (i,j) \in E \text{ está en la ruta de algún vehículo} \\ 0 & \text{en otro caso} \end{cases}$$

$$y_{ir} = \begin{cases} 1 & \text{si la carga del cliente } i \in C \text{ es transportada hasta el vuelo } r \in R \\ 0 & \text{en otro caso} \end{cases}$$

$$a_j = \text{tiempo de llegada del vehículo al nodo } j \in V$$

Ante la ausencia del índice k, será necesario añadir una variable de flujo para controlar el paso de la mercancía por los arcos. Esta nueva variable se define de la forma:

$$f_{ij}^{c} = \begin{cases} 1 & \text{si por el arco } (i,j) \text{ va la mercancía del cliente } c \in C \\ 0 & \text{en otro caso} \end{cases}$$

Además se incluye una nueva variable capaz de identificar cuándo un itinerario de vuelo es utilizado. Esta variable se define como:

$$z_r = \begin{cases} 1 & \text{si el vuelo } r \in R^* \text{ es utilizado por algún vehículo} \\ 0 & \text{en otro caso} \end{cases}$$

La formulación del modelo queda como sigue:

$$\text{mín} \sum_{i \in V} \sum_{j \in V \setminus \{i\}} c_{ij} x_{ij} + \sum_{i \in C} \sum_{r \in R^*} F_{ir} y_{ir}$$

sujeto a:

$$\sum_{r\in R} y_{ir} = 1 \qquad \forall i \in C \tag{2.18}$$

$$\sum_{j\in V} x_{dj} = \sum_{j\in V} x_{jd} \leq |K| \tag{2.19}$$

$$\sum_{j\in V} x_{ij} = \sum_{j\in V} x_{ji} = 1 \qquad \forall i \in C \tag{2.20}$$

$$\sum_{i\in V\setminus\{r\}} x_{ri} = \sum_{i\in V\setminus\{r\}} x_{ir} = z_r \qquad \forall r \in R^* \tag{2.21}$$

$$a_i \geq \theta_{op} + t_{di} \qquad \forall i \in V \tag{2.22}$$

$$a_r \leq \text{mín}\{\theta_{cl} - t_{rd}, Q_r\} \qquad \forall r \in R^* \tag{2.23}$$

$$a_i \leq \overline{a}_i = \max_{r\in R}\{\text{mín}\{Q_r, \theta_{cl} - t_{rd}\} - t_{ir}\} \qquad \forall i \in C \tag{2.24}$$

$$a_r \leq \underline{a}_r = \theta_{op} + \text{mín}\{t_{di} + t_{ir}\} \qquad \forall r \in R^* \tag{2.25}$$

$$x_{ji}\big(M_{ij} - t_{ij} - t_{ji}\big) + \big(x_{ij} - 1\big)M_{ij} + a_i + t_{ij} \leq a_j \qquad \forall i \in V, \; \forall j \in V\setminus\{i,d\} \tag{2.26}$$

$$\big(y_{ir} - 1\big)M_{ir} + a_i + t_{ir} \leq a_r \qquad \forall i \in C, \forall r \in R^* \tag{2.27}$$

$$y_{ir} \leq \sum_{j\in V\setminus\{i\}} x_i j \qquad \forall i \in C, \forall r \in R^* \tag{2.28}$$

$$y_{ir} \leq \sum_{j\in V\setminus\{r\}} x_r j \qquad \forall i \in C, \forall r \in R^* \tag{2.29}$$

$$f^c_{di} = f^c_{id} = 0 \qquad \forall i \in V, \forall c \in C \tag{2.30}$$

$$\sum_{j\in V} f^c_{ij} - \sum_{j\in V} f^c_{ji} = \begin{cases} 0 & \text{si } c \in C \text{ y } i \neq c \\ 1 & \text{si } i = c \\ -y_{ci} & \text{si } i \in R \end{cases} \tag{2.31}$$

$$f^c_{ij} \leq x_{ij} \qquad \forall i, j \in V, \forall c \in C \tag{2.32}$$

$$y_{ir} \leq z_r \qquad \forall i \in C, \forall r \in R^* \tag{2.33}$$

$$z_r \geq z_{r+1} \qquad \forall r \in R^* \tag{2.34}$$

$$y_{1r} = 0 \qquad \forall r \in R^*\setminus R_1 \tag{2.35}$$

$$x_{ij},\; y_{ir},\; z_r \in \{0,1\},\; a_i \geq 0,\; 0 \leq f^c_{ij} \leq 1 \qquad \forall i, j \in V, i \neq j, \; \forall r \in R^* \tag{2.36}$$

Se añaden en este modelo, con respecto al *Modelo con tres índices*, el conjunto de restricciones (2.19), que establece que la cantidad de arcos que salen del depósito y llegan a él, debe ser menor que el número de vehículos disponibles. Se sustituyen las restricciones (2.4) por las inecuaciones (2.20) y (2.21), que aseguran que sólo un vehículo visita cada cliente y que todas sus mercancías tienen como destino algún itinerario de vuelo respectivamente.

Se excluyen del primer modelo, las restricciones para evitar subciclos y simetrías, (2.13), (2.14) y (2.15), ya que la adición de la nueva variable de flujo trae consigo el conjunto de restricciones (2.30) y (2.31). Con el primero, se controla el flujo entre los nodos y el depósito, mientras que con el segundo conjunto se evitan las posibles simetrías. Es necesario añadir una relación entre la variable flujo f_{ij}^c y la variable x_{ij}, que queda reflejada en (2.32). Para reforzar las restricciones contra simetrías y subciclos, se añaden las desigualdades (2.33), (2.34) y (2.35).

Las demás restricciones no mencionadas en este modelo, además de la función objetivo, se mantienen, a pesar de la eliminación del índice k, con el mismo objetivo que en el modelo original, no permitiendo que dos o más vehículos visiten el mismo vuelo.

Limitar el número de visitas a cada nodo-vuelo, dará una solución factible que no siempre será óptima. Así, la solución que se obtenga al aplicar este modelo será una cota superior a la obtenida con el *Modelo con tres índices*, alcanzando la óptima sólo cuando m es suficientemente grande, como por ejemplo, cuando $m = |K|$.

2.2.1. Modelo con dos índices y $m = 1$

Si el parámetro m de la sección anterior, tomase el valor $m = 1$, no sería necesaria la duplicación de los itinerarios de vuelo. Esto supondría, seguir trabajando con el conjunto de itinerarios disponibles R del *Modelo con tres índices* y la exclusión de la variable z_r.

Se eliminan del modelo, las restricciones (2.33), (2.34) y (2.35), y se sustituye el conjunto de desigualdades (2.21) por

$$\sum_{i \in V \setminus \{r\}} x_{ir} = \sum_{i \in V \setminus \{r\}} x_{ri} \quad \forall r \in R \tag{2.37}$$

Además, se añaden las restricciones

$$\sum_{j \in V} x_{ir} \leq 1 \quad \forall r \in R \tag{2.38}$$

que junto a las (2.18) aseguran que cada cliente tenga asignado un sólo itinerario de vuelo.

2.2.2. Modelo con dos índices y $m = 2$

Otro caso particular de este modelo, es cuando se hace $m = 2$. En este caso, la cota superior obtenida será de mejor calidad, asegurando que es el óptimo cuando no más de dos vehículos visitan el mismo nodo-vuelo. Sin embargo, el problema es más difícil de resolver, como se verá en el capítulo de esta memoria dedicado a los resultados computacionales.

2.3. Modelo con una variable para cada ruta

Uno de los principales inconvenientes de tratar de resolver este problema, surge al manejar un número no moderado de variables. Es por ello por lo que, como variante al modelo original, se procede a la resolución de un problema maestro que tome como variables de decisión, cada ruta posible de la solución del ATD-PDP, que engloba a un subconjunto de clientes.

Para la formulación de este problema maestro, se define una variable de decisión que indica cuándo una ruta es seleccionada de la siguiente manera:

$$x_t = \begin{cases} 1 & \text{si la ruta } t \in T \text{ es seleccionada} \\ 0 & \text{en otro caso} \end{cases}$$

Además, se genera un parámetro que indica cuándo un cliente es visitado en una ruta concreta:

$$I_t^c = \begin{cases} 1 & \text{si la visita al cliente } c \in C \text{ forma parte de la ruta } t \in T \\ 0 & \text{en otro caso} \end{cases}$$

donde T es el conjunto de las rutas posibles. Se denota por α_t, con $t \in T$, el costo asociado a cada una de estas rutas. El objetivo del problema, es la elección óptima de las rutas para que todos los clientes sean visitados y sus respectivas mercancías sean transportadas a sus vuelos correspondientes, con el mínimo coste.

El problema maestro será:

$$\text{mín} \sum_{t \in T} \alpha_t x_t \tag{2.39}$$

sujeto a:

$$\sum_{t \in T} I_t^c x_t = 1 \qquad \forall c \in C \tag{2.40}$$

$$\sum_{t \in T} x_t \leq |K| \tag{2.41}$$

$$x_t \in \{0, 1\} \qquad \forall t \in T \tag{2.42}$$

Este modelo incluye al conjunto de restricciones en (2.40), para indicar que los clientes sólo son visitados una vez, es decir, que de haber más de una ruta, un mismo cliente sólo debe formar parte de una. Con las desigualdades en (2.41), se asegura que no hay más rutas que número de vehículos disponibles. Finalmente con (2.42), se hace referencia a la condición binaria de la variable x_t.

Para la obtención del conjunto T de rutas y sus costos α_t, $t \in T$, se resuelve un subproblema en el que se restringe la visita a los nodos por un sólo vehículo.

Esto supondría no tratar con el índice k del conjunto de vehículos, luego es el *Modelo con dos índices*, el que se modifica para la resolución del subproblema.

Las restricciones

$$\sum_{r \in R} y_{ir} = I_t^i \qquad \forall i \in C \tag{2.43}$$

se diferencian del conjunto de desigualdades en (2.18), por la adición del nuevo parámetro I_t^c, pero su función sigue siendo la de asegurar que cada cliente tiene asignado un itinerario de vuelo. Además las inecuaciones en (2.19) y (2.21), son sustituídas por

$$\sum_{j \in V \setminus \{i\}} x_{ij} = \sum_{j \in V \setminus \{i\}} x_{ji} \qquad \forall i \in V \tag{2.44}$$

Como el modelo está formulado para un sólo vehículo no son necesarias las variables de flujo y no se incluyen las restricciones en (2.30), (2.31) y (2.32). Además, la variable z_r es prescindible, luego las desigualdades (2.33), (2.34) y (2.35) se excluyen y es utilizado el conjunto de restricciones (2.43), en vez de las (2.21). Las demás restricciones se mantienen igual.

A la hora de resolver el subproblema, el parámetro I_t^c, es un vector de tamaño $|C|$ cuyas coordenadas van rotando y transformándose en nuevos parámetros, uno por cada ruta t calculada. De esta forma, al terminar de resolverlo, se obtiene un conjunto de posibles rutas con sus respectivos costos, que son los datos tomados para la resolución del problema maestro.

Algoritmo de Generación de Columnas

La estrategia utilizada en esta sección para la resolución del ATD-PDP, se denomina *Algoritmo de Generación de Columnas*, y es aplicado a problemas de programación lineal donde el número de variables de decisión es exponencial. Este tipo de problemas, a menudo genera ciertas dificultades en su resolución debido a la cantidad de columnas presentes en sus restricciones. Por ello, el algoritmo consta de dos problemas a resolver que reciben el nombre de *problema maestro* y *subproblema*. Aplicado al ATD-PDP, el procedimiento ha sido:

1. Resolver el subproblema en busca de las rutas más adecuadas para visitar cada cliente y transportar su mercancía hasta los aeropuertos. Esta solución generará las columnas necesarias para poder resolver el problema maestro.
2. Resolver el problema maestro para decidir cuáles de las rutas obtenidas en el subproblema son las que producen el menor costo, mediante las columnas generaras anteriormente.

Algunos ejemplos de su uso, se encuentran en problemas como el *Travelling Salesman Problem* (TSP), por J-J Salazar González[3], y el *Capacitated vehicle routing problem* (CVRP), por P. Toth y D.Vigo[4]. El objetivo de su aplicación, es comprobar si realmente minimiza los tiempos de resolución del ATD-PDP.

3

Resultados computacionales

Se aplican los modelos estudiados a un conjunto de instancias con variaciones en el número de clientes, aeropuertos y vehículos, en busca de diferencias significativas en los tiempos de resolución para cada modelo descrito anteriormente, analizando los resultados obtenidos.

3.1. Conjunto de instancias

Una vez estudiados los modelos alternativos al propuesto por Azadian et al.[1], se procede a la transcripción de éstos a un software específico de optimización combinatoria con el fin de comparar sus rendimientos.

Los modelos son aplicados al conjunto de instancias propuestos por Azadian et al.[1]. En ellas se encuentran las localizaciones del depósito, los clientes y los aeropuertos, los costos de los vuelos asociados a cada cliente y el tiempo de salida de los mismos. Además, estas instancias se estructuran según el número de clientes, aeropuertos y vehículos:

$$|C| = \{7, 10, 15\} \qquad |H| = \{1, 2\} \qquad |K| = \{3, 4, 5\}.$$

Para cada valor de $|C|$, $|H|$, y $|K|$ hay 20 instancias, con lo que tenemos un total de 360 situaciones posibles en las que poder aplicar el problema del ATD-PDP.

Ejemplo 3.1. Las instancias que han sido utilizadas para aplicar los modelos estudiados, son como se muestra en la figura 3.1. Concretamente en este problema, la empresa encargada de realizar los transportes, dispone de tres vehículos para la recogida de la mercancía de siete clientes, y su posterior entrega en un aeropuerto con tres opciones de vuelos. Dado que hay tres itinerarios de vuelos disponibles, al proceder en busca del grafo transformado, la localización del aeropuerto debe triplicarse.

```
Number of Customers:7
Number of Airports:1
Number of Vehicles:3

Problem 1 of 20

Depot Location:
15  76

Customers Locations:
5  79
25  32
25  80
71  20
68  26
90  70
3  72

Projected Airports Locations:
50  48
50  48
50  48

Flight-Itinerary Departure Time
159  198  540

Flight-Itinerary Costs
(rows=customers, columns=itinararies)
240  580  582
232  462  506
273  364  518
188  279  576
243  348  523
176  432  457
159  361  475
```

Figura 3.1. Ejemplo de instancia

3.2. IBM ILOG CPLEX Optimization Studio

La aplicación de los modelos al conjunto de instancias, se ha realizado a través de *IBM ILOG CPLEX Optimization Studio*, en adelante CPLEX[5], que es un software de modelado para la resolución de problemas de programación matemática.

El lenguaje de programación utilizado en CPLEX, está basado en lenguaje OPL[5] (*Optimization Programming Language*), con el que poder modelar programación lineal, entera o mixta entre otros. Además, cuenta con librerías para otros lenguajes de programación tradicionales como son C, C++, Java, Phyton, etc. OPL permite modelar gran cantidad de problemas de optimización siguiendo la estructura que se muestra en la figura 3.2.

En estos fichero con extensión *.mod*, se modela el problema a resolver, y el conjunto de datos sobre el que aplicar el modelo, se encontrará en otro fichero de extensión *.dat*.

Para la resolución de los modelos en CPLEX, es necesaria la utilización de *configuraciones de ejecución*, donde se agregan el modelo y el conjunto de datos sobre el que aplicar un problema concreto. Surge así la necesidad de la creación de un fichero .dat con cada uno de los problemas de cada instancia para su posterior resolución.

Dada la gran cantidad de situaciones posibles a las que aplicar cada uno de los modelos, y la limitación de las configuraciones de ejecución, se ha creado un programa en CPLEX capaz de rotar cada conjunto de datos para cada modelo

```
//Variables
dvar float+   variable 1;
.
.
 variable n;

//Objetive
minimize/maximize   function

//Restrictions
suject to
{
 restriction 1;
.
.
 restriction n;
}
```

Figura 3.2. Estructura de un modelo en CPLEX

y grabar los resultados obtenidos para su posterior análisis. La formulación de los modelos y de este programa en CPLEX, se adjunta en el apéndice A.

3.3. Visual Studio 2015

Otro software utilizado en el estudio de este proyecto, es el *Visual Studio 2015*, un entorno de desarrollo integrado (*IDE*), capaz de trabajar con lenguajes de programación como C++, C, Visual Basic, etc.

Ante la necesidad de transformar los datos de las instancias generadas por Azadian et al.[1], se codifica un programa en cógido C++ mediante este software, con el que se crea un fichero con la extensión .dat para cada una de las instancias, un total de 360 conjuntos de datos como se ha mencionado anteriormente. El programa creado se adjunta en el apéndice B. Se muestra en la figura 3.3, la instancia del ejemplo 3.1, una vez aplicado el programa generado en C++.

3.4. Resultados obtenidos

Cada modelo estudiado, ha sido codificado en CPLEX y aplicado a los conjuntos de datos, proponiendo un tiempo límite de 1800 segundos de ejecución,

```
//Problem 1 of 20

// Horario de apertura
del deposito://
Op= 0;
Cl= 840;

// N de clientes e
itinerarios://
numC= 7;
numR= 3;

// Localizaciones de los
nodos://
locVd=[
[15,76],
[5,79],
[25,32],
[25,80],
[71,20],
[68,26],
[90,70],
[3,72],
[50,48],
[50,48],
[50,48]
];

//Tiempo de salida de los
itinerarios://
tR=
[159,198,540];

//Costos por cliente y
vuelo://
cost=[
[240,580,582],
[232,462,506],
[273,364,518],
[188,279,576],
[243,348,523],
[176,432,457],
[159,361,475]
];

//N de vehiculos://
numK= 3;
```

Figura 3.3. Ejemplo de fichero con extensión *.dat*

para controlar el tiempo que trabaja el programa. La tabla 3.1 especifica la notación que se le dará a cada modelo hasta ahora estudiado en el segundo capítulo.

Nombre del modelo		**Notación**
Modelo con tres índices	y las desigualdades de simetría (2.15)	M1.1
	y las desigualdades de simetría (2.17)	M1.2
Modelo con dos índices	y con $m = 1$	M2.1
	y con $m = 2$	M2.2
Modelo con una variable para cada ruta		M3

Tabla 3.1. Notación de los modelos

El análisis realizado sobre los resultados obtenidos, se estructura según el número de clientes, aeropuertos y vehículos presentes en las instancias. Se comienza por la elaboración de una tabla con el tiempo promedio de ejecución (T) y el número de problemas que han llegado al tiempo límite (t.l.), seguida de

otra tabla con el mejor valor medio objetivo obtenido (Obj.), y el porcentaje de superación de éste para cada modelo (%).

			Modelo									
$\|C\|$	$\|H\|$	$\|K\|$	**M1.1**		**M1.2**		**M2.1**		**M2.2**		**M3**	
			T	**t.l.**	**T**	**t.l.**	**T**	**t.l.**	**T**	**t.l.**	**T**	**t.l.**
7	1	3	0,9	0	0,9	0	1,2	0	17,9	0	15,8	0
		4	1,4	0	1,1	0	1,1	0	22,1	0	22,4	0
		5	1,5	0	1,0	0	0,5	0	11,3	0	21,1	0
	2	3	11,2	0	10,4	0	21,3	0	1223,4	12	53,1	0
		4	22,6	0	15,4	0	16,2	0	1064,9	9	36,9	0
		5	15,8	0	7,6	0	14,5	0	708,0	4	35,6	0
10	1	3	23,5	0	24,3	0	151,6	1	1053,4	10	166,4	0
		4	29,0	0	26,9	0	150,5	1	822,8	5	171,6	0
		5	47,8	0	36,5	0	216,5	2	905,0	7	153,9	0
	2	3	250,6	1	182,1	0	509,8	2	1775,5	19	397,9	1
		4	765,4	5	488,0	3	661,8	3	1766,7	19	430,7	0
		5	889,0	8	612,9	3	840,2	7	1711,3	19	453,6	0

Tabla 3.2. Promedios de los tiempos de ejecución y número de veces en alcanzar el tiempo límite para siete y diez clientes

En la tabla 3.2, se muestran los resultados obtenidos para $|C| = \{7, 10\}$, y para cada uno de los modelos estudiados en este trabajo. Se observa que para un conjunto pequeño de datos, siete clientes y un aeropuerto, los tiempos de ejecución son bajos y no existen problemas en los que se haya alcanzado el límite de tiempo. A medida que aumenta el tamaño del conjunto de datos, aumentan los tiempos de ejecución y es más probable que algún problema alcance el tiempo límite durante su resolución, como puede observarse para diez clientes y dos aeropuertos. De los modelos aplicados, el que mejores tiempos obtiene es el modelo M1.2. Se recuerda que, a diferencia del modelo M1.1, en éste las restricciones contra posibles simetrías no dependían directamente de los costos. Podría seguirle como mejor modelo el M1.1, en cuanto a tiempos de ejecución se refiere, pero para el modelo M3, a pesar de que los tiempos de ejecución para un número menor de datos no es tan bueno como para el modelo M1.1, al aumentar el número de clientes, aeropuertos o vehículos, se consigue una mejora gradual de los tiempos de ejecución, lo que es visible en la cantidad de problemas que alcanzan el tiempo límite para M1.1 y M3. Cabe destacar que el problema maestro del modelo M3, siempre fue resuelto en menos de 0,5 segundos, siendo el subproblema el que tardaba más en resolverse.

Una vez resueltos los problemas, se comparan las soluciones promedio obtenidas de cada modelo. Seleccionado el mejor de estos en la columna (Obj.) de la tabla 3.3, se calcula el porcentaje de superación de cada modelo con respecto

$\|C\|$	$\|H\|$	$\|K\|$	**Obj.**	**Modelo**				
				M1.1	**M1.2**	**M2.1**	**M2.2**	**M3**
				%	%	%	%	%
7	1	3	2056,5	0,0	0,0	2,5	0,0	0,0
		4	2118,3	0,0	0,0	1,4	0,0	0,0
		5	2201,6	0,0	0,0	0,8	0,2	0,0
	2	3	1820,0	0,0	0,0	0,5	0,1	0,0
		4	1855,8	0,2	0,0	0,4	0,2	0,0
		5	1947,6	0,0	0,0	0,2	0,0	0,0
10	1	3	2799,4	0,0	0,0	2,0	0,1	2,4
		4	2924,0	0,0	0,0	1,6	0,0	2,0
		5	2893,6	0,0	0,0	2,0	0,1	2,5
	2	3	2537,3	0,0	0,0	0,3	0,4	2,7
		4	2690,2	0,0	0,0	0,6	0,9	2,1
		5	2685,6	0,5	0,3	0,6	1,9	1,7

Tabla 3.3. Mejores promedios de valores objetivos y porcentajes de superación para siete y diez clientes

a este mejor resultado. Cada 0,0 % mostrado en la tabla, indica que todas las soluciones obtenidas son óptimas. Nuevamente, el mejor de los modelos es el M1.2, que en muy pocos casos no alcanza la solución óptima, seguido del modelo M1.1. Los peores resultados se obtienen para los modelos con dos índices, los modelos M2.1 y M2.2, ya que en la mayoría de los problemas, obtienen soluciones factibles pero no óptimas. Además, se observa que para siete clientes, dos aeropuertos y cuatro vehículos, el modelo M1.2 obtiene una solución óptima diferente a los demás (con valor óptimo menor). Esto se debe a que el modelo M1.2 fue más tolerante a la hora de cumplir las restricciones de las ventanas de tiempo del comienzo de los vuelos

$\|C\|$	$\|H\|$	$\|K\|$	**Modelo**			
			M1.1		**M1.2**	
			T	**t.l.**	**T**	**t.l.**
15	1	3	867,5	8	921,9	9
		4	1055,8	10	960,0	10
		5	1209,6	10	943,8	6

Tabla 3.4. Promedios de los tiempos de ejecución y número de veces en alcanzar el tiempo límite para quince clientes

Para quince clientes, $|C| = 15$, sólo se incluyen en las tablas 3.4 y 3.5, los resultados obtenidos para los modelos M1.1 y M1.2 para un sólo aeropuerto.

Esta decisión fue tomada, en base a la cantidad de problemas que alcanzaban el tiempo límite para dos aeropuerto. Igualmente, como se muestra en la tabla 3.4, el modelo M1.2, ha resuelto más problemas que el M1.1 en menos de 1800 segundos, luego sigue siendo el mejor modelo.

				Modelo	
$\|C\|$	$\|H\|$	$\|K\|$	**Obj.**	**M1.1**	**M1.2**
				%	%
15	1	3	4413,70	0,06	0,10
		4	4509,37	0,40	0,26
		5	4384,49	0,05	0,06

Tabla 3.5. Mejores promedios de valores objetivos y porcentajes de superación para quince clientes

Sin embargo, los porcentajes que se muestran en la tabla 3.5, implicarían un mejor rendimiento, en cuanto a las soluciones promedio óptimas obtenidas y por un margen de diferencia muy pequeño, por parte del modelo M1.1, que consigue, salvo para cuatro vehículos, mejores porcentajes de superación que el modelo M1.2.

4

Conclusiones

En esta memoria, se ha estudiado el *An unpaired pickup and delivery problem with time dependent assignment costs*, ATP-PDP, un problema particular del *Pickup and delevery problem*, PDP, en el que no sólo entran en juego decisiones para la óptima elección de las rutas que serán recorridas para la recogida y entrega de la mercancía de ciertos clientes, sino que además, estas rutas están condicionadas por la entrega a tiempo en los aeropuertos para su posterior transporte. La industria de la aviación, ha evolucionado en las últimas décadas debido al aumento del transporte por vía aérea de mercancías y cargamentos comerciales. Es por ello, por lo que las empresas dedicadas a transportar las cargas por vía terrestre de los clientes hasta los aeropuertos, deben seleccionar correctamente las rutas a realizar. La importancia de establecer los trayectos de forma adecuada, influye directemente en el beneficio económico de estas empresas, ya que la óptima solución del ATD-PDP, les proporciona el mínimo coste de llevar a cabo estas operaciones.

La correcta formulación de un modelo matemático para la resolución del problema en el menor tiempo posible, tendrá como recompensa la aplicación directa en situaciones reales en las que poder aplicar el ATD-PDP. Con lo cual, en este trabajo, se ha buscado la manera de disminuir los tiempos de ejecución del modelo estudiado por Azadian et al.[1], con la propuesta de modelos alternativos que fueron comparados posteriormente.

La aplicación de softwares matemáticos para la resolución de problemas de optimización, es esencial para el estudio de problemas de programación matemática de gran complejidad operacional. La utilización de *IBM ILOG CPLEX Optimization Studio* en esta memoria, ha supuesto una ventaja significativa en la obtención de los resultados necesarios para el estudio del ATD-PDP, ya que posee una interfaz muy intuitiva que proporciona mayor comodidad en su uso que otros softwares de aplicaciones similires. Aunque no es un programa informático de adquisición gratuita, la compañía IBM posee licencias para profesorado y

alumnado de instituciones públicas sin coste económico, lo que ha facilitado su uso en este trabajo.

De los modelos estudiados en esta memoria, los propuestos por Azadian et al.[1], son los que mejores resultados obtienen una vez son aplicados a cierto conjunto de instancias y situaciones posibles en las que resolver el ATD-PDP. El mejor de ellos, es el *Modelo con tres índices* y el conjunto de restricciones (2.17) cuya función es evitar simetrías en las soluciones. A diferencia de las desigualdades (2.15), este conjunto no depende directamente de los costos, luego esta es la ventaja que se ha asociado en esta memoria a la obtención de los bajos tiempos de ejecución.

A estos modelos les sigue el *Modelo con una variable para cada ruta* si es aplicado a un conjunto no muy grande de clientes, ya que, aunque los tiempos de ejecución no son tan buenos como con los anteriores modelos, el valor objetivo resultante en la mayoría de los casos es el óptimo. Si el modelo es aplicado a más de siete clientes, estos valores objetivos pierden su condición óptima y pasa a ser el que peor resultados consigue.

Los *Modelos con dos índices* propuestos en esta memoria, no cumplen los objetivos previstos de disminuir los tiempo de ejecución, aunque entre ellos, si bien para $m = 1$ la resolución de los problemas es más corta que para $m = 2$, los valores objetivos que proporcionan se alejan más para el primero que para el segundo. Sólo serán óptimos los resultados, cuando el número final de vehículos utilizados en la solución coincide con el valor de m.

Es evidente que la complejidad del ATD-PDP crece al aumentar el tamaño del problema, lo que es visible en los resultados obtenidos para el conjunto de quince clientes, en los que sólo han podido ser aplicados los *Modelos con tres índices*. Con el modelo M3, sólo se ha podido garantizar la solución óptima con siete clientes, ya que sólo en este caso se resolvían todos los subproblemas que generaban la ruta óptima fijados los clientes a visitar. Sin embargo, si el cálculo de los costos de estas rutas se realizara de forma más inteligente (sin necesidad de calcularlos todos) con estrategias de generación de columnas, los resultados computacionales serían mejores. Una línea de trabajo futura, podría ser el estudio del modelo M3 de manera más extensa.

A

Apéndice I

Se muestran a continuación los modelos programados en CPLEX con los que se obtuvieron los resultados como parte del estudio del problema de Azadian et al.[1].

A.1. Modelo con tres índices en su notación

El código en CPLEX de los modelos con tres índices estudiados en el capítulo 2, son los siguientes.

Modelo con un primer tipo de desigualdades de simetría

```
//Datos
//int numD=...; //depósito
int numC=...; //clientes
int numR=...; //itinerarios
int numK=...; //vehículos

//range D= 1..numD;
range C= 1..numC; //número de clientes
range R= numC+1..numC+numR; //número de
itinerarios
range Vd= 0..numC+numR; //conjunto V con
el depósito
range V= 1..numC+numR; //conjunto V sin
el depósito
range COORD= 1..2;
range K= 1..numK;
range K1= 1..numK-1;

//Datos del depósito y los vuelos
int locVd[Vd][COORD]=...;
int Op=...;
int Cl=...;
int tR[R]=...; //Tiempo de salida de los
itinerarios
int cost[C][R]=...; //Costo de vuelo
cliente-itinerario

//Coordenadas del conjunto V con depósito
// int XVd[Vd]=...;
// int YVd[Vd]=...;

//Constante M
int M=1200;

//Ejecución de distancias: Distancia
entre nodos del conjunto V con depósito
float distVd[Vd][Vd];
execute{
var i;
var j;
for (i in Vd){
for (j in Vd){
distVd[i][j]=
Opl.sqrt(Opl.pow(locVd[i][1]-
locVd[j][1],2)+Opl.pow(locVd[i][2]-
locVd[j][2],2));
} } }

float asup[R];
execute{
var r;
for (r in R){
asup[r]= Opl.minl(Cl-distVd[r][0], tR[r]);
} }

float amax[Vd];
execute{
var i;
var r;
var temp;
for (i in C){
temp = 0;
for(r in R){
if( temp < asup[r]-distVd[i][r]) {
temp =  asup[r]-distVd[i][r]
}
amax[i]= temp
} }
for (r in R){
amax[r]= asup[r];
}
amax[0]= Cl;
}

float ainf[R];
execute{
var r;
var i;
var temp;
for (r in R){
temp = 100000;
for (i in C){
if(temp> distVd[0][i]+ distVd[i][r]) {
temp=  distVd[0][i]+ distVd[i][r]
} }
ainf[r]= Op + temp;
} }

float Mij[Vd][Vd];
execute{
var i;
var j;
for(i in Vd){
for (j in Vd){
Mij[i][j]= amax[i]+distVd[i][j];
```

```
} } }

//Variables
dvar boolean x[Vd][Vd][K];
dvar boolean y[C][R][K];
dvar float+ a[Vd][K];

//Función objetivo
minimize
sum (k in K) (
sum(i in Vd, j in Vd: j!=i)
 distVd[i][j]*x[i][j][k] +
 sum(i in C, r in R) cost[i][r] *
 y[i][r][k]);

//Restriciones
subject to {

// Fijamos los arcos que van de un
nodo a si mismo a 0
forall (i in Vd, k in K)
x[i][i][k] == 0;

forall (i in C)
sum(r in R, k in K)
y[i][r][k]==1;

forall (i in Vd, k in K)
sum(j in Vd: j!=i)
x[i][j][k]<=1;

forall (j in Vd, k in K)
sum(i in Vd: i!=j) x[i][j][k]-
sum(i in Vd: i!=j) x[j][i][k]==0;

forall (i in Vd, j in V, k in K: j!=i)
x[j][i][k]*(Mij[i][j] - distVd[i][j] -
distVd[j][i])+(x[i][j][k]-1)*Mij[i][j] +
 a[i][k] + distVd[i][j]
<= a[j][k];

forall (i in C, k in K)
a[i][k]>= Op+distVd[0][i];

forall (r in R, k in K)
a[r][k] <= asup[r];

// Con eta restricción hay que tener
cuidado ya que puede suceder que
asup[r] < ainf[r] con lo que este modelo
no tendría solución sin embargo tendría
sentido formular el problema eliminando
los r que cumplen esto. Así es preferible
no añadirla al modelo.
/* forall (r in R, k in K)
a[r][k] >= ainf[r]; */

forall (i in C, r in R, k in K)
(y[i][r][k]-1)*Mij[i][r] + a[i][k] +
distVd[i][r] <= a[r][k];

forall (i in C, r in R, k in K)
y[i][r][k]<= sum(j in Vd: j!=i) x[i][j][k];

forall (i in C, r in R, k in K)
y[i][r][k]<= sum(j in Vd: j!=r) x[r][j][k];

forall (i in C, k in K)
a[i][k]<= amax[i];

forall (i in C, k in K)
sum(j in Vd: j!=i) x[i][j][k] ==
sum(r in R) y[i][r][k];

// Hemos detectado un problema en el modelo
del artículo. Creemos que se evita con esta
restricción
forall (r in R, k in K)
sum(i in Vd: i !=r) x[i][r][k] <=
sum(i in C) y[i][r][k];

forall (k in K1)
sum(i in Vd, j in Vd: j!=i)
 distVd[i][j]*x[i][j][k] <=
sum(i in Vd, j in Vd: j!=i)
distVd[i][j]*x[i][j][k+1];
}
```

Modelo con un segundo tipo de desigualdades de simetrías

```
//Datos
int numC=...; //clientes
int numR=...; //itinerarios
int numK=...; //vehículos

range C= 1..numC; //número de clientes
range R= numC+1..numC+numR; //número de
itinerarios
range Vd= 0..numC+numR; //conjunto V con
el depósito
range V= 1..numC+numR; //conjunto V sin
el depósito
range COORD= 1..2;
range K= 1..numK;
range K1= 1..numK-1;

//Datos del depósito y los vuelos
int locVd[Vd][COORD]=...;
int Op=...;
int C1=...;
int tR[R]=...; //Tiempo de salida de los
itinerarios
int cost[C][R]=...; //Costo de vuelo
cliente-itinerario

//Coordenadas del conjunto V con depósito
// int XVd[Vd]=...;
// int YVd[Vd]=...;

//Constante M
// int M=...;

//Ejecución de distancias
//Distancia entre nodos del conjunto V
con depósito
float distVd[Vd][Vd];
execute{
var i;
var j;
for (i in Vd){
for (j in Vd){
distVd[i][j]=
Opl.sqrt(Opl.pow(locVd[i][1]-
locVd[j][1],2)+Opl.pow(locVd[i][2]-
locVd[j][2],2));
} } }

float asup[R];
execute{
var r;
for (r in R){
asup[r]= Opl.minl(C1-distVd[r][0], tR[r]);
} }

float amax[Vd];
execute{
var i;
var r;
var temp;
for (i in C){
temp = 0;
for(r in R){
if( temp < asup[r]-distVd[i][r]) {
temp =  asup[r]-distVd[i][r]
}
amax[i]= temp
} }
for (r in R){
amax[r]= asup[r];
}
amax[0]= C1;
}

float ainf[R];
execute{
var r;
var i;
var temp;
for (r in R){
temp = 100000;
for (i in C){
if(temp> distVd[0][i]+ distVd[i][r]) {
temp=  distVd[0][i]+ distVd[i][r]
} }
ainf[r]= Op + temp;
} }

float Mij[Vd][Vd];
execute{
var i;
var j;
for(i in Vd){
for (j in Vd){
Mij[i][j]= amax[i]+distVd[i][j];
} } }

//Variables
dvar boolean x[Vd][Vd][K];
dvar boolean y[C][R][K];
dvar float+ a[Vd][K];

//Función objetivo
minimize
sum (k in K) (
sum(i in Vd, j in Vd: j!=i)
distVd[i][j]*x[i][j][k] +
sum(i in C, r in R) cost[i][r] * y[i][r][k]);

//Restriciones
subject to {

// Fijamos los arcos que van de un nodo a si
mismo a 0
forall (i in Vd, k in K)
x[i][i][k] == 0;

forall (i in C)
sum(r in R, k in K)
y[i][r][k]==1;

forall (i in Vd, k in K)
sum(j in Vd: j!=i)
x[i][j][k]<=1;

forall (j in Vd, k in K)
sum(i in Vd: i!=j) x[i][j][k]-
sum(i in Vd: i!=j) x[j][i][k]==0;

forall (i in Vd, j in V, k in K: j!=i)
x[j][i][k]*
(Mij[i][j] - distVd[i][j] - distVd[j][i])+
(x[i][j][k]-1)*Mij[i][j] + a[i][k] +
distVd[i][j] <= a[j][k];

forall (i in C, k in K)
a[i][k]>= Op+distVd[0][i];

forall (r in R, k in K)
a[r][k] <= asup[r];

// Con eta restricción hay que tener cuidado
ya que puede suceder que asup[r] < ainf[r]
con lo que este modelo no tendría solución
sin embargo tendría sentido formular el
problema eliminando los r que cumplen esto.
Así es preferible no añadirla al modelo.
/* forall (r in R, k in K)
```

```
a[r][k] >= ainf[r]; */

forall (i in C, r in R, k in K)
(y[i][r][k]-1)*Mij[i][r] + a[i][k] +
distVd[i][r] <= a[r][k];

forall (i in C, r in R, k in K)
y[i][r][k]<= sum(j in Vd: j!=i) x[i][j][k];

forall (i in C, r in R, k in K)
y[i][r][k]<= sum(j in Vd: j!=r) x[r][j][k];
```

```
forall (i in C, k in K)
a[i][k]<= amax[i];

forall (i in C, k in K)
sum(j in Vd: j!=i) x[i][j][k] ==
sum(r in R) y[i][r][k];

// Hemos detectado un problema en el
modelo del artículo. Creemos que se
evita con esta restricción
```

```
forall (r in R, k in K)
sum(i in Vd: i !=r) x[i][r][k] <=
sum(i in C) y[i][r][k];

forall (k in K1)
sum(i in C) pow(2,numC-i)*
(sum(r in R) y[i][r][k]) >=
sum(i in C) pow(2,numC-i)*
(sum(r in R) y[i][r][k+1]);
}
```

A.2. Modelo con dos índices en su notación

A continuación, se muestra el código para los modelos con dos índices.

Modelo con dos índices y $m = 1$

```
//Datos
int numC=...; //clientes
int numR=...; //itinerarios
int numK=...; //vehículos

range C= 1..numC; //número de clientes
range R= numC+1..numC+numR; //número de
itinerarios
range Vd= 0..numC+numR; //conjunto V con
el depósito
range V= 1..numC+numR; //conjunto V sin
el depósito
range COORD= 1..2;
range K= 1..numK;

//Datos del depósito y los vuelos
int locVd[Vd][COORD]=...;
int Op=...;
int Cl=...;
int tR[R]=...; //Tiempo de salida de los
itinerarios
int cost[C][R]=...; //Costo de vuelo
cliente-itinerario

//Coordenadas del conjunto V con depósito
// int XVd[Vd]=...;
// int YVd[Vd]=...;

//Constante M
//int M=...;

//Ejecución de distancias
//Distancia entre nodos del conjunto V con
depósito
float distVd[Vd][Vd];
execute{
var i;
var j;
for (i in Vd){
for (j in Vd){
distVd[i][j]=
Opl.sqrt(Opl.pow(locVd[i][1]-
locVd[j][1],2)+Opl.pow(locVd[i][2]-
locVd[j][2],2));
} } }

float asup[R];
execute{
var r;
for (r in R){
asup[r]= Opl.minl(Cl-distVd[r][0], tR[r]);
} }

float amax[Vd];
execute{
var i;
```

```
var r;
var temp;
for (i in C){
temp = 0;
for(r in R){
if( temp < asup[r]-distVd[i][r]) {
temp =  asup[r]-distVd[i][r]
}
amax[i]= temp
} }
for (r in R){
amax[r]= asup[r];
}
amax[0]= Cl;
}

float ainf[R];
execute{
var r;
var i;
var temp;
for (r in R){
temp = 100000;
for (i in C){
if(temp> distVd[0][i]+ distVd[i][r]) {
temp=  distVd[0][i]+ distVd[i][r]
} }
ainf[r]= Op + temp;
} }

float Mij[Vd][Vd];
execute{
var i;
var j;
for(i in Vd){
for (j in Vd){
Mij[i][j]= amax[i]+distVd[i][j];
} } }

//Variables
dvar boolean x[Vd][Vd];
dvar boolean y[C][R];
dvar float+ a[Vd];
dvar boolean f[C][Vd][Vd]; //Nueva
variable de flujo

//Función objetivo
minimize
sum(i in Vd, j in Vd: j!=i)
distVd[i][j]*x[i][j] + sum(i in C, r in R)
cost[i][r] * y[i][r];

//Restriciones
subject to {

// Fijamos los arcos de a si mismo a 0
```

```
sum(i in Vd) x[i][i] == 0;

forall (i in C)
sum(r in R) y[i][r]==1;

sum(j in V) x[0][j] <= numK;

sum(j in V) x[j][0] - sum(j in V) x[0][j] == 0 ;

forall (i in C)
sum(j in Vd: j!=i) x[i][j]==1;

forall (i in C)
sum(j in Vd: j!=i) x[j][i]==1;

forall (r in R)
sum(j in Vd: j!= r) x[j][r]<=1;

forall (r in R)
sum(j in Vd: j!= r) x[r][j] -
sum(j in Vd: j!= r) x[j][r] == 0;

forall (i in Vd, j in V: j!=i)
x[j][i]*(Mij[i][j] - distVd[i][j] - distVd[j][i])
+(x[i][j]-1)*Mij[i][j] + a[i] + distVd[i][j]
<= a[j];

forall (i in C)
a[i]>= Op+distVd[0][i];

forall (r in R)
a[r]<= asup[r];

// Con eta restricción hay que tener cuidado ya
que puede suceder que asup[r] < ainf[r] con lo
que este modelo no tendría solución sin embargo
tendría sentido formular el problema eliminando
los r que cumplen esto. Así es preferible no
añadirla al modelo.
/* forall (r in R)
a[r]>= ainf[r]; */


forall (i in C, r in R)
(y[i][r]-1)*Mij[i][r] + a[i] + distVd[i][r] <= a[r];

forall (i in C, r in R)
y[i][r]<= sum(j in Vd: j!=i) x[i][j];

forall (i in C, r in R)
y[i][r]<= sum(j in Vd: j!=r) x[r][j];

forall (i in C)
a[i]<= amax[i];

/*Nuevas restricciones para la variable flujo*/
```

```
forall (c in C, i in V)
f[c][0][i]==0;

forall (c in C, i in V)
f[c][i][0]==0;
forall (c in C, i in Vd, j in Vd)
f[c][i][j]<=x[i][j];
forall (c in C, i in C: i!=c)
sum(j in Vd) f[c][i][j] -
sum(j in Vd) f[c][j][i] == 0;
forall (c in C)
sum(j in Vd) f[c][c][j] -
sum(j in Vd) f[c][j][c] == 1;
forall (c in C, i in R)
sum(j in Vd) f[c][i][j] -
sum(j in Vd) f[c][j][i] == -y[c][i];
}
```

Modelo con dos índices y $m = 2$

```
//Datos
int numC=...; //clientes
int numR=...; //itinerarios
int numK=...; //vehículos

range C= 1..numC; //número de clientes
range R= numC+1..numC+2*numR;//Itinerarios
range R1= numC+1..numC+numR;//Itinerarios
de la primer copia
range R2= numC+numR+1..numC+2*numR;//Itine-
rarios de la segunda copia
range Vd= 0..numC+2*numR; //conjunto V
con el depósito
range V= 1..numC+2*numR; //conjunto V
sin el depósito
range LOC= 0..numC+numR; // En las
localizaciones consideramos sólo la
primera copia de los vuelos
range COORD= 1..2;
range K= 1..numK;

// Las localizaciones de los vuelos se
pasan una sola vez
int locVd[LOC][COORD]=...;
int Op=...;
int Cl=...;

int tR[R1]=...;      //Tiempo de salida
de los itinerarios
int cost[C][R1]=...; //Costo de vuelo
cliente-itinerario

//Ejecución de distancias: Distancia entre
nodos del conjunto V con depósito
float distVd[Vd][Vd];
execute{
var i;
var j;
for (i in LOC){
for (j in LOC){
distVd[i][j]=
Opl.sqrt(Opl.pow(locVd[i][1]-
locVd[j][1],2)+Opl.pow(locVd[i][2]-
locVd[j][2],2));
} }
for (i in R2){
for (j in LOC){
distVd[i][j]= distVd[i-numR][j];
} }
for (i in LOC){
for (j in R2){
distVd[i][j]= distVd[i][j-numR];
} }
for (i in R2){
for (j in R2){
distVd[i][j]= distVd[i-numR][j-numR];
} } }

float asup[R];
execute{
var r;
for (r in R1){
asup[r]= Opl.minl(Cl-distVd[r][0], tR[r]);
}
for (r in R2){
asup[r]= asup[r-numR];
} }

float amax[Vd];
execute{
var i;
var r;
var temp;
for (i in C){
temp = 0;
for(r in R){
if( temp < asup[r]-distVd[i][r]) {
temp =  asup[r]-distVd[i][r]
}
amax[i]= temp
} }
for (r in R){
amax[r]= asup[r];
}
amax[0]= Cl;
}

float ainf[R];
execute{
var r;
var i;
var temp;
for (r in R){
temp = 100000;
for (i in C){
if(temp> distVd[0][i]+ distVd[i][r]) {
temp=  distVd[0][i]+ distVd[i][r]
} }
ainf[r]= Op + temp;
} }

float Mij[Vd][Vd];
execute{
var i;
var j;
for(i in Vd){
for (j in Vd){
Mij[i][j]= amax[i]+distVd[i][j];
} } }

//Variables
dvar boolean x[Vd][Vd];
dvar boolean y[C][R];
dvar float+ a[Vd];
/*Nueva variable de flujo*/
dvar boolean f[C][Vd][Vd];
dvar boolean z[R];   // Variable que indica
si el vuelo r es usado

//Función objetivo
minimize
sum(i in Vd, j in Vd: j!=i) distVd[i][j]*
x[i][j] +
sum(i in C, r in R1) cost[i][r] *
y[i][r] +
sum(i in C, r in R2) cost[i][r-numR] *
y[i][r];

//Restriciones
subject to {

forall (i in C)
sum(r in R) y[i][r]==1;
sum(j in V) x[0][j] <= numK;
sum(j in V) x[j][0] - sum(j in V) x[0][j] == 0 ;

forall (i in C)
sum(j in Vd: j!=0) x[0][j]<=numK;

forall (i in C)
sum(j in Vd: j!=i) x[i][j]==1;

forall (i in C)
sum(j in Vd: j!=i) x[j][i]==1;

forall (r in R)
sum(j in Vd: j!= r) x[j][r]<=1;

forall (r in R)
sum(j in Vd: j!= r) x[r][j] -
sum(j in Vd: j!= r) x[j][r] == 0;

forall (i in Vd, j in V: j!=i)
x[j][i]*(Mij[i][j] -
distVd[i][j] - distVd[j][i])+
(x[i][j]-1)*Mij[i][j] + a[i] + distVd[i][j]
<= a[j];

forall (i in C)
a[i]>= Op+distVd[0][i];

forall (r in R)
a[r]<= asup[r];

forall (r in R)
a[r]>= ainf[r];

forall (i in C, r in R)
(y[i][r]-1)*Mij[i][r] + a[i] + distVd[i][r]
<= a[r];

forall (i in C, r in R)
y[i][r]<= sum(j in Vd: j!=i) x[i][j];

forall (i in C, r in R)
y[i][r]<= sum(j in Vd: j!=r) x[r][j];

forall (i in C)
a[i]<= amax[i];

/*Nuevas restricciones para la variable
flujo*/
forall (c in C, i in V)
f[c][0][i]==0;

forall (c in C, i in V)
f[c][i][0]==0;
forall (c in C, i in Vd, j in Vd)
f[c][i][j]<=x[i][j];
forall (c in C, i in C: i!=c)
sum(j in Vd) f[c][i][j] -
sum(j in Vd) f[c][j][i] == 0;
forall (c in C)
sum(j in Vd) f[c][c][j] -
sum(j in Vd) f[c][j][c] == 1;
forall (c in C, i in R)
sum(j in Vd) f[c][i][j] -
sum(j in Vd) f[c][j][i] ==-y[c][i];
}
```

A.3. Código para la rotación de los modelos

Dada la gran cantidad de conjuntos de datos a los que aplicar estos modelos, se propuso la creación de un programa en CPLEX capaz de rotar cada problema y cada modelo con la grabación de los resultados en un fichero de extensión .txt. El código es el que sigue:

```
main{
var f = new IloOplOutputFile("resultados.txt");
f.writeln("file\tmodel\tairport\tcustomer\tvehicles\tdataset
\tObj.\tStatus\tTime");
for (i=2 ; i<=4; i++){
modelname = "Modelo_";
modelname = modelname + i + ".mod";
var Source = new IloOplModelSource(modelname);
var Def = new IloOplModelDefinition(Source);
var Cplex = new IloCplex();
Cplex.tilim = 1800;  // Tiempo límite en segundos
for(c=7; c<=10; c++) {
if(c != 7 && c != 10 && c != 15) continue;
for (k=1; k<= 2 ; k++){
for (h=3 ; h<= 5; h++){
for (j=1 ; j <= 20; j++){
filename = "C" + c + "A" + k + "V" + h + "_";
if (j < 10) filename += "0";
filename += j + ".dat";
Data = new IloOplDataSource(filename);
model = new IloOplModel(Def,Cplex);
model.addDataSource(Data);
model.generate();
//var time0 = new Date();
f.write(filename, "\t", i, "\t", k, "\t", c, "\t", h, "\t", j,
 "\t");
if ( Cplex.solve() ) {
var curr = Cplex.getObjValue();
f.write(curr, "\t");
}
else{
f.write("0\t");
}
var time = Cplex.getSolvedTime();
f.writeln(Cplex.getCplexStatus(), "\t", time);
//f.writeln(Cplex.getCplexStatus(), "\t", time, "\t", gap);
} /* for (j ... datos */
}  /* for (h ...  vehículos */
}  /* for (k ... aeropuertos */
}  /* for (c ...  clientes */
}  /* for (i ...   modelos */
f.close();
} /* main()  */
```

A.4. Modelo con una variable para cada ruta

Para este último modelo, han sido necesarios tres programas en CPLEX. EL primer programa, es el modelo con el que se estudian las mejores rutas capaces de visitar cada cliente para la posterior entrega de su mercancía en los aeropuertos.

```
//Datos
//int numD=...; //depósito
int numC=...; //clientes
int numR=...; //itinerarios
int numK=...; //vehículos

range C= 1..numC; //número de clientes
range R= numC+1..numC+numR; //número de
itinerarios
range Vd= 0..numC+numR; //conjunto V con
el depósito
range V= 1..numC+numR; //conjunto V sin
el depósito
range COORD= 1..2;
range K= 1..numK;
range K1= 1..numK-1;

//Datos del depósito
int locVd[Vd][COORD]=...;
int Op=...;
int Cl=...;

int tR[R]=...; //Tiempo de salida de
los itinerarios
int cost[C][R]=...; //Costo de vuelo
cliente-itinerario

//Ejecución de distancias
//Distancia entre nodos del conjunto V
con depósito

float distVd[Vd][Vd];
execute{
var i;
var j;
for (i in Vd){
for (j in Vd){
distVd[i][j]= Opl.sqrt(Opl.pow(locVd[i][1]-
locVd[j][1],2)+ Opl.pow(locVd[i][2]-
locVd[j][2],2));
} } }

float asup[R];
execute{
var r;
for (r in R){
asup[r]= Opl.minl(Cl-distVd[r][0], tR[r]);
}
}

float amax[Vd];
execute{
var i;
var r;
var temp;
for (i in C){
temp = 0;
for(r in R){
if( temp < asup[r]-distVd[i][r]) {
temp =  asup[r]-distVd[i][r]
}
amax[i]= temp
} }
for (r in R){
amax[r]= asup[r];
}
amax[0]= Cl;
}

float ainf[R];
execute{
var r;
var i;
var temp;
for (r in R){
temp = 100000;
for (i in C){
if(temp> distVd[0][i]+ distVd[i][r]) {
temp=  distVd[0][i]+ distVd[i][r]
} }
ainf[r]= Op + temp;
} }

float Mij[Vd][Vd];
execute{
var i;
var j;
for(i in Vd){
for (j in Vd){
Mij[i][j]= amax[i]+distVd[i][j];
} } }

int Inc[C];
execute{
var i;
for(i in C){
Inc[i]=0;
}
Inc[1]=1;
}

//Variables
```

```
dvar boolean x[Vd][Vd];
dvar boolean y[C][R];
dvar float+ a[Vd];

//Función objetivo
minimize
sum (k in K) (
sum(i in Vd, j in Vd: j!=i) distVd[i][j]*
x[i][j] *sum(i in C, r in R) cost[i][r] *
y[i][r]);

//Restriciones
subject to {

forall (i in C)
sum(r in R)
y[i][r]== Inc[i];

forall (i in Vd)
sum(j in Vd: j!=i)
x[i][j] <=1;

forall (j in Vd)
sum(i in Vd: i!=j) x[i][j] -
sum(i in Vd: i!=j) x[j][i] == 0;

forall (i in Vd, j in V: j!=i)
x[j][i] *(Mij[i][j] - distVd[i][j] -
distVd[j][i])+(x[i][j]-1) * Mij[i][j]
 + a[i] + distVd[i][j] <= a[j];

forall (i in C)
a[i] >= Op+distVd[0][i];

forall (r in R)
a[r] <= asup[r];

forall (r in R)
a[r] >= ainf[r];

forall (i in C, r in R)
(y[i][r]-1)*Mij[i][r] + a[i] + distVd[i][r] <= a[r];

forall (i in C, r in R)
y[i][r] <= sum(j in Vd: j!=i) x[i][j];

forall (i in C, r in R)
y[i][r]<= sum(j in Vd: j!=r) x[r][j];

forall (i in C)
a[i]<= amax[i];

forall (i in C, k in K)
sum(j in Vd: j!=i) x[i][j] == sum(r in R) y[i][r];

}
```

A continuación, se muestra el programa capaz de reescribir las soluciones al modelo anterior, para crear un fichero .dat que sería utilizado en la resolución del siguiente problema.

```
main{

var fr = new IloOplOutputFile("resultadosI.txt");
fr.writeln("file\tmodel\tairport\tcustomer\tvehicles\tdataset
\tObj.\tStatus\tTime\tTimeSub\tTimeMaster");

// Creating the master-model
var masterModelSource = new IloOplModelSource("Master.mod");
var masterDef = new IloOplModelDefinition(masterModelSource);
var masterCplex = new IloCplex();
masterCplex.tilim = 1800;

// Preparing sub-model source, definition and engine
var subModelSource = new IloOplModelSource("Modelo1V.mod");
var subDef = new IloOplModelDefinition(subModelSource);
var subCplex = new IloCplex();
subCplex.tilim = 100;  // Tiempo límite en segundos
// subOpl.addDataSource(subData);
// subOpl.generate();

for(var c=10; c<=10; c++) {
if(c != 7 && c != 10 && c != 15) continue;
for (var k=2; k<= 2 ; k++){
for (var h=5 ; h<= 5; h++){
for (var j=19 ; j <= 19; j++){

var fileSubData = "C" + c + "A" + k + "V" + h + "_I_";
if (j < 10) fileSubData += "0";
fileSubData += j + ".dat";
var subData = new IloOplDataSource(fileSubData);
var subOpl = new IloOplModel(subDef,subCplex);
subOpl.addDataSource(subData);
subOpl.generate();

var fileMasterData = "C" + c + "A" + k + "V" + h + "_M_";
if (j < 10) fileMasterData += "0";
fileMasterData += j + ".dat";
var fd = new IloOplOutputFile(fileMasterData);
fd.writeln("numC = ", c, ";");
fd.writeln("numK = ", h, ";");
fd.writeln("Routes = { ");

var counter = 1;
var column_generation_time = 0.0;
while (1){

// prepare next iteration
subDef = subOpl.modelDefinition;
subData = subOpl.dataElements;
subOpl = new IloOplModel(subDef,subCplex);

// Modificamos los datos para cada subproblema
var i;
var continuar = true;
while(continuar) {
for(i=c; i>=1 ; i--) {
if(subData.Inc[i] == 0) {
subData.Inc[i] = 1;
for(var j2 =i+1; j2<= c; j2++) subData.Inc[j2]= 0;
break;
}
}
var suma = 0;
for(var j2 =1; j2<= c; j2++) suma += subData.Inc[j2];
if(suma <= 6 || i < 1) continuar = false;
}
if(i <1 ) break; // por aquí salimos del while(1)

subOpl.addDataSource(subData);
subOpl.generate();

if ( subCplex.solve() ) {
var curr = subCplex.getObjValue();
var time = subCplex.getSolvedTime();
if(counter > 1) {
fd.write(",");
}
fd.write("<", counter, ", ", curr, ", [");
for(var j2=1; j2 < c; j2++) {
fd.write(subOpl.Inc[j2], ", ");
}
fd.writeln(subOpl.Inc[c], "] >  // ", time);
column_generation_time += time;
counter++;
} /* if ... */
else {
column_generation_time += subCplex.getSolvedTime();
}
}  // while (1)
fd.writeln("};");
fd.writeln("//", column_generation_time);
fd.close();

// Resolvemos ahora el master
var masterData = new IloOplDataSource(fileMasterData);
var masterOpl = new IloOplModel(masterDef, masterCplex);
masterOpl.addDataSource(masterData);
masterOpl.generate();

fr.write(fileSubData, "\t", 6, "\t", k, "\t", c, "\t", h, "\t", j, "\t");
if ( masterCplex.solve() ) {
var curr = masterCplex.getObjValue();
fr.write(curr, "\t");
}
else{
fr.write("0\t");
}
var time = masterCplex.getSolvedTime();
fr.writeln(masterCplex.getCplexStatus(), "\t", column_generation_time + time,
"\t", column_generation_time, "\t", time);
} // for (j ... datos
}  //  for (h ...  vehículos
}  //  for (k ... aeropuertos
}  //  for (c ...  clientes
```

```
fr.close();
} /* main()  */
```

Finalmente, el programa capaz de decidir cuáles son las rutas ótimas para la ejecución del problema, el problema maestro:

```
int numC=...; //clientes
int numK=...; // Vehículos
range C= 1..numC; //número de clientes

tuple routes {
key int id;
float cost;
int cust[C];}
{routes} Routes = ...;

dvar boolean x[Routes];

minimize
sum( r in Routes )
r.cost * x[r];

subject to {
forall( i in C )
sum( r in Routes )
r.cust[i] * x[r] == 1;
sum( r in Routes )
x[r] <= numK;
}
```

B

Apéndice II

Se comparte el código utilizado para la conversión de cada uno de los problemas de las instancias, en un fichero con extensión .dat.

B.1. Código de Visual Studio 2015

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <regex>

/*************************************/
/* We define constant                */
/*************************************/
// strings
#define OUT_EXTENSION ".dat"

// Numeric
#define OP_VAL 0
#define CL_VAL 840
#define CONST_M CL_VAL
#define NUM_PROBLEMS_INPUT_FILE 20

/*************************************/


/*************************************/
/* Functions' Declaration            */
/*************************************/
static void readKeyAndValue(std::
istringstream &iss, std::string &sKey,
std::string &sValue);
static int read_and_write(std::
string &input_filename, std::string
&output_string);
static int read_number_customers_airports
_and_vehicles(const std::string &s,
int &numC, int &numH, int &numV);
static int check_number_of_problems(const
std::string &s, const int numProblem);
static int check_depot_location(const std::
string &s);
static int check_customer_locations(const
std::string &s);
static int check_airports_locations(const
std::string &s);
static int check_flight_time(const std::
string &s);
static int check_flight_cost(const std::
string &s);
static void read_integers_and_store_
them_in_a_vector(const std::string &s,
std::vector<int> &v);
static bool is_an_integer_string(const std::
string &s);
static bool is_a_space_string(const std::
string &s);
static bool is_a_list_of_integers(const std::
string &s);



/*************************************/
/* Main                              */
/*************************************/

int    main(int ac, char **av)
{
// Para que no haya problemas con la ñ y
acentos
std::locale::global(std::locale("spanish"));



if (ac != 3) {
std::cout << "Use: " << av[0] << std::endl;
std::cout << av[0] << " input_filename
output_string" << std::endl;
std::cout << "input_filename: is the name
of the input file (i.e., C15A2V5.txt)" <<
std::endl;
std::cout << "output_string: is initial
commun name for output files." << std::endl;
std::cout << "For example if output_string=
\"C15A2V5\", the aoutput files will be" <<
std::endl;
std::cout << "\"C15A2V5_01" <<
OUT_EXTENSION << "\", \"C15A2V5_02"
<< OUT_EXTENSION <<"\", ... " <<
std::endl;
return(0);
}
std::string input_filename(av[1]);
std::string output_string(av[2]);
int err = read_and_write(input_filename,
output_string);
return(err);
}

static int read_and_write(std::string
&input_filename, std::string &output_string)
{
std::ifstream ifs(input_filename, std::ios::in);
if (!ifs) { std::cerr << "It can't open the
input file" << std::endl; return(1); }

std::string sLine, sWord, sValue;

int numC; // Number of Customers
int numH; // Number od Airports
int numK; // Number of Vehicles
int numR; // Number of Itineraries

// We read a line of the file and ignore the
space lines
while (std::getline(ifs, sLine) &&
is_a_space_string(sLine));

// if err != 0 an error have been happened
int err = read_number_customers_airports_and_
vehicles(sLine, numC, numH, numK);
if(err) {
std::cerr << "Error in function read_number_
customers_airports_and_vehicles"
<< std::endl;
return(1);
}

// We read each problem
for (int i = 1; i <= NUM_PROBLEMS_INPUT_FILE;
i++) {
// We read a line of the file and ignore the
empty lines
while (std::getline(ifs, sLine) && is_a_space
_string(sLine));

// We read the number of problems
err = check_number_of_problems(sLine, i);
```

```
if (err) { std::cerr << "Error in function
check_number_of_problem" << std::endl;
return(1); }

// We create a structure to store the
coordenates
std::vector<int> coordenates;

// We read the depot location and store
data
while (std::getline(ifs, sLine) &&
is_a_space_string(sLine));
err= check_depot_location(sLine);
if (err) { std::cerr << "Error in
function check_depot_location" <<
std::endl; return(1); }
while (std::getline(ifs, sLine) &&
is_a_space_string(sLine));
if(!is_a_list_of_integers(sLine))  {
std::cerr << "ERROR reading integers
in the input file" << std::endl;
return(1);
}
read_integers_and_store_them_in_a_
vector(sLine, coordenates);

while (std::getline(ifs, sLine) &&
is_a_space_string(sLine));

// We read customers' locations while
(std::getline(ifs, sLine) &&
is_a_space_string(sLine));

err= check_customer_locations(sLine);
if (err) { std::cerr << "Error in
function check_customer_locations"
<< std::endl; return(1); }
for (int j = 1; j <= numC; j++)
{
while (std::getline(ifs, sLine) &&
is_a_space_string(sLine));
if(!is_a_list_of_integers(sLine)) {
std::cerr << "ERROR reading integers
in the input file" << std::endl;
return(1);
}
read_integers_and_store_them_in_a_
vector(sLine, coordenates);
}
if (ifs.fail()) {
std::cerr << "ERROR reading the input
file" << std::endl;
return(1);
}



while (std::getline(ifs, sLine) &&
is_a_space_string(sLine));

// We read the projected airports
locations and store data
// We do not know the number of
locations (maybe 3* numH)
err= check_airports_locations(sLine);
if (err) { std::cerr << "Error in
function check_airports_locations"
<< std::endl; return(1); }
numR=0;  // We set the number of
itineraries
while (std::getline(ifs, sLine) &&
is_a_list_of_integers(sLine))
{
read_integers_and_store_them_in_a_
vector(sLine, coordenates);
numR++;
}
if (ifs.fail()) {
std::cerr << "ERROR reading the
input file" << std::endl;
return(1);
}



while (std::getline(ifs, sLine) &&
is_a_space_string(sLine));
```

```
//We read Flight-Itinerary Departure
Time
check_flight_time(sLine);
if (err) { std::cerr << "Error in
function check_flight_time" <<
std::endl; return(1); }
std::vector<int> itineraryDepartures;
while (std::getline(ifs, sLine) &&
is_a_space_string(sLine));
if (!is_a_list_of_integers(sLine)) {
std::cerr << "ERROR reading integers
in the input file" << std::endl;
return(1);
}
read_integers_and_store_them_in_a_
vector(sLine, itineraryDepartures);
if (ifs.fail()) {
std::cerr << "ERROR reading the
input file" << std::endl;
return(1);
}
if (numR != itineraryDepartures.size()){
std::cerr << "ERROR: numR != number
of iteniraries" << std::endl;
return(1);
}

while (std::getline(ifs, sLine) &&
is_a_space_string(sLine));

//We read Flight-Itinerary Costs
err = check_flight_cost(sLine);
if (err) { std::cerr << "Error in
function check_flight_cost" <<
std::endl; return(1); }
std::vector<int> flightCost;
for (int j = 1; j <= numC; j++)
{
while (std::getline(ifs, sLine) &&
is_a_space_string(sLine));
if (!is_a_list_of_integers(sLine)) {
std::cerr << "ERROR reading integers
in the input file" << std::endl;
return(1);
}
read_integers_and_store_them_in_a_
vector(sLine, flightCost);
}
if (ifs.fail()) {
std::cerr << "ERROR reading the
input file" << std::endl;
return(1);
}
if (numC * numR != flightCost.size()) {
std::cerr << "ERROR: numC * numR !=
flightCost.size()" << std::endl;
return(1);
}


// We generate the output file name
std::string output_filename(output_string);
if (i < 10) output_filename.append("0");
output_filename.append(std::to_string(i));
output_filename.append(OUT_EXTENSION);
// We open the output file name
std::ofstream ofs(output_filename,
std::ios::out);
if (!ofs) { std::cerr << "It can't open
the output file" << std::endl; return(1); }

// We write the head of the file
ofs << "/*********************************"
<< std::endl;
ofs << "* OPL 12.6.3.0 Data" << std::endl;
ofs << "* Author: Marta" << std::endl;
ofs << "*********************************/"
<< std::endl;
ofs << "\n// Horario de apertura del depósito:"
NUM_PROBLEMS_INPUT_FILE << std::endl;

// We write the time windows of the depot
```

```
<< std::endl;
ofs << "Op= " << OP_VAL << ";" << std::endl;
ofs << "Cl= " << CL_VAL << ";" << std::endl;

// We write parameters
ofs << "\n// Número de clientes e itinerarios:"
<< std::endl;
ofs << "numC= " << numC << ";" << std::endl;
ofs << "numR= " << numR << ";" << std::endl;


// We write locations
ofs << "\n// Localizaciones de los nodos:"
<< std::endl;
ofs << "locVd=[" << std::endl;
for (int j = 0; j < (numC + numR); j++) {
ofs << "\t[" << coordenates[j * 2] << ","
<< coordenates[j * 2 + 1] << "]," << std::endl;
}
ofs << "\t[" << coordenates[2 * (numC + numR)]
<< "," << coordenates[2 * (numC + numR) + 1] << "]"
<< std::endl;
ofs << "\t]" << ";" << std::endl;

/*********************************/
/* Falta rellenar aquí el resto de código
para que escriba el fichero*/
/* Para escribir utilizamos ofs */

// We write  the flight-itinerary
departure time
ofs << "\n//Tiempo de salida de los
itinerarios:" << std::endl;
ofs << "tR=" << std::endl << "\t[";
for (int j = 1; j < numR; j++) {
ofs <<  itineraryDepartures[j-1] << ",";
}
ofs << itineraryDepartures[numR-1] << "]"
<< ";" << std::endl;

// We write the flight-itinerary cost
ofs << "\n//Costos por cliente y vuelo:"
<< std::endl;
ofs << "cost=[" << std::endl;
int count = 0;
for (int k = 1; k <= numC; k++) {
ofs << "\t[";
for (int j = 1; j < numR; j++) {
ofs << flightCost[count++] << ",";
}
ofs << flightCost[count++] << "]";
if (k < numC)  ofs << ",";
ofs << std::endl;
}
ofs << "];" << std::endl;

ofs << "\n//Número de vehículos" <<
std::endl;
ofs << "numK= " << numK << ";" <<
std::endl;

/*********************************/

ofs.close();
}


ifs.close();
return(0);
}

static int read_number_customers_
airports_and_vehicles(const std::
string &s,
int &numC, int &numH, int &numK)
{
// First, we read Number of Customers,
Number of Airports and Number of Vehicles
// Number of Customers
std::istringstream iss(s);
std::string sWord, sValue;
iss >> sWord;
if (sWord != "Number") { std::cerr <<
"We expected read word \"Number\"" <<
std::endl; return(1);}
iss >> sWord;
```

```
if (sWord != "of") { std::cerr << "We
expected read word \"of\"" << std::endl;
return(1); }
readKeyAndValue(iss, sWord, sValue);
if(sWord != "Customers") { std::cerr <<
"We expected read word \"Customers\""
<< std::endl; return(1); }
numC = std::stoi(sValue);
// Number of Airports
iss >> sWord;
if (sWord != "Number") { std::cerr <<
"We expected read word \"Number\"" <<
std::endl; return(1); }
iss >> sWord;
if (sWord != "of") { std::cerr << "We
expected read word \"of\"" << std::endl;
return(1); }
readKeyAndValue(iss, sWord, sValue);
if (sWord != "Airports") { std::cerr <<
"We expected read word \"Airports\"" <<
std::endl; return(1); }
numH = std::stoi(sValue);
// Number of Vehicles
iss >> sWord;
if (sWord != "Number") { std::cerr << "We
expected read word \"Number\"" <<
std::endl; return(1); }
iss >> sWord;
if (sWord != "of") { std::cerr << "We
expected read word \"of\"" << std::endl;
return(1); }
readKeyAndValue(iss, sWord, sValue);
if (sWord != "Vehicles") { std::cerr << "We
expected read word \"Vehicles\"" << std::endl;
return(1); }
numK = std::stoi(sValue);
return(0);
}

static int check_number_of_problems( const
std::string &s, const int numProblem)
{
std::istringstream iss(s);
std::string sWord, sValue;
iss >> sWord;
if (sWord != "Problem") { std::cerr << "We
expected read word \"Problem\"" <<
std::endl; return(1); }
iss >> sValue;
if(numProblem != std::stoi(sValue))
{ std::cerr << "We expected read number "
<< numProblem << "instead of "
<< sValue << std::endl; return(1); }
iss >> sWord;
if (sWord != "of") { std::cerr << "We
expected read word \"of\"" << std::endl;
return(1); }
iss >> sValue;
if (NUM_PROBLEMS_INPUT_FILE != std::
stoi(sValue)) {
std::cerr << "We expected read number "
<< NUM_PROBLEMS_INPUT_FILE << "instead of "
<< sValue << std::endl; return(1);
}
return(0);
}

static int check_depot_location(const
std::string &s)
{
std::istringstream iss(s);
std::string sWord, sValue;
iss >> sWord;
if (sWord != "Depot") { std::cerr << "We
expected read word \"Depot\"" <<
std::endl; return(1); }
iss >> sWord;
if (sWord != "Location:") { std::cerr <<
"We expected read word \"Location:\"" <<
std::endl; return(1); }
return(0);
}

static int check_customer_locations(const
std::string &s)
{
std::istringstream iss(s);
std::string sWord, sValue;
iss >> sWord;
if (sWord != "Customers") { std::cerr <<
"We expected read word \"Customers\"" <<
std::endl; return(1); }
iss >> sWord;
if (sWord != "Locations:") { std::cerr <<
"We expected read word \"Locations:\"" <<
std::endl; return(1); }
return(0);
}

static int check_airports_locations(const
std::string &s)
{
std::istringstream iss(s);
std::string sWord, sValue;
iss >> sWord;
if (sWord != "Projected") { std::cerr <<
"We expected read word \"Projected\"" <<
std::endl; return(1); }
iss >> sWord;
if (sWord != "Airports") { std::cerr <<
"We expected read word \"Airports\"" <<
std::endl; return(1); }
iss >> sWord;
if (sWord != "Locations:") { std::cerr <<
"We expected read word \"Locations:\"" <<
std::endl; return(1); }
return(0);
}

//Flight_Time
static int check_flight_time(const std::
string & s)
{
std::istringstream iss(s);
std::string sWord, sValue;
iss >> sWord;
if (sWord != "Flight-Itinerary") { std::
cerr << "We expected read word \"Flight-
Itinerary\"" << std::endl; return(1); }
iss >> sWord;
if (sWord != "Departure") { std::cerr <<
"We expected read word \"Departure\"" <<
std::endl; return(1); }
iss >> sWord;
if (sWord != "Time") { std::cerr << "We
expected read word \"Time\"" <<
std::endl; return(1); }
return 0;
}

//Flight_Costs
int check_flight_cost(const std::
string & s)
{
std::istringstream iss(s);
std::string sWord, sValue;
iss >> sWord;
if (sWord != "Flight-Itinerary") { std::
cerr << "We expected read word \"Flight-
Itinerary\"" << std::endl; return(1); }
iss >> sWord;
if (sWord != "Costs") { std::cerr << "We
expected read word \"Costs\"" <<
std::endl; return(1); }
iss >> sWord;
if (sWord != "(rows=customers,") { std:
:cerr << "We expected read word
\"(rows=customers,\"" << std::endl;
return(1); }
iss >> sWord;
if (sWord != "columns=itinararies)") {
std::cerr << "We expected read word
\"columns=itinararies)\"" << std::endl;
return(1); }
return 0;
}



static void read_integers_and_store_them_
in_a_vector(const std::string &s, std::
vector<int> &v)
{
std::istringstream iss(s);
std::string sValue;
while(true) {
iss >> sValue;
if(iss.fail() || !is_an_integer_
string(sValue)) {
break;
}
v.push_back(std::stoi(sValue));
}
}

static void readKeyAndValue(std::
istringstream &iss, std::string &sKey,
std::string &sValue)
{
iss >> sKey;
std::size_t pos = sKey.find(':');

if (pos != std::string::npos) {
//Está pegado los dos puntos
if (pos < sKey.size() - 1) {
// Está pegado el número
sValue = sKey.substr(pos+1,
sKey.size() - pos - 1);
}
else { // No está pegado el número
iss >> sValue;
}
sKey.erase(pos);
}
else {
iss >> sValue;
if(sValue==":") iss >> sValue;
}
}

static bool is_an_integer_string
(const std::string &s)
{
return(std::regex_match(s, std::
regex("[\\-\\+]?[0-9]+")));
}

static bool is_a_space_string(const
std::string &s)
{
if(s.size() <= 1) return(true);
return(std::regex_match(s, std::
regex("\\s+")));
}

static bool is_a_list_of_integers(const
std::string &s)
{
return(std::regex_match(s, std::
regex("(\\s*[\\-\\+]?[0-9]+)+\\s*")));
}
```

Bibliografía

[1] F. Azadian, A. Murat, R-B. Chinnam. *An Unpaired Pickup and Delivery Problem with Time Dependent Assignment Cost: Application in Air Cargo Transportatio.* European Journal of Operational Research (2017)

[2] F. Azadian. *An integrated framework for freight forwarders:exploitation of dynamic information for multimodal transportation.* Wayne State University Dissertations (2012) 496.

[3] J-J. Salazar González. *Programación matemática.* Ediciones Díaz de Santos (2001)

[4] P. Toth, D. Vigo. *Models, relaxations and exact approaches for the capacitated vehicle routing problem.* Discrete Applied Mathematics 123 (2002) 487-512.

[5] IBM ILOG CPLEX Optimization Studio *Getting Started with the IDE*

Lista de Tablas

Lista de Figuras

An unpaired pickup and delivery problem with time dependent assignment cost

Marta Hernández Hernández

Facultad de Ciencias · Sección de Matemáticas

Universidad de La Laguna

alu0100699234@ull.edu.es

Abstract

The unpaired pickup and delivery problem with time dependent assignment costs (ATD-PDP) is a mathematical problem whose aplication seeks the minimum cost of carrying out operations of organization and distribution by companies engaged in freight transport. These companies are in charge of mobilizing the merchandise to the airports, and choose of flight itineraries for each cargo that will later be transported by air. In this report, this model is studied, and alternatives are looked for to reduce its execution times.

1. Introduction

This work is realted to a pickup and delivery problem where some costumers' loads are mobilized to airports that will later be transported by air. The formulation of the model associated to this problem, minimizes the costs of generating routes by road of the loads and the choice of the flights itineraries for each customer. To companies dedicated to loads transportation, find a complex organization problem: seek flight itineraries for each customer, pickup the loads and delivery them in airports on time. A math model is propused by F. Azadian, A. Murat and R-B. Chinnam[1], that seeks a solution of the problem. These companies' profits directly depend on these correct choices.
This report studies the model proposed by Azadian et al.[1] and proposes other two models. Besides the models are compared to decide what is the best.

2. Problem description

The ATD-PDP creates an operational plan to loads pickup and delivery of costumers to airports as soon as flight itineraries options for each customer, seeking minimum costs. In the first chapter, main characteristics are described. There are two sets of decisions:

- Decisions to select a flight itinerary for each costumer, where the load will be transportated by air.
- Decisions to extraction of the routes that will do the vehicles to pickup and delivery of the load.

Let $G_0 = (V_0, E_0)$ directed graph , with $V_0 = \{d\} \cup C \cup H$, where d is the depot, C is the set of customers and H is the set of airports. In addition, K is defined as a set of vehicles and the opening and closing times of the depot are known. Also the travel costs and the travel times are known, and the costs and the times of departure of the flight itineraries for each customer.

Time dependent delivery cost

The assignments of the flights will be feasible if the vehicles arrive at airport with the loads before the flight has started. The flight will not change to another flight unless there were not flights availables, that would suppose a penalty cost $F^h_{i0} > F^h_{ir}$, where F^h_{i0} is the penalty cost and F^h_{ir} is flight cost for itinerary $r \in R$ of the airport $h \in H$ for the customer $i \in C$. Then, the next function of dependent airport delivery cost of delivering customer $i \in C$'s load to airport $h \in R$ at time t is defined as follows:

$$f(h,i,t) = \begin{cases} min_{r \in R_h}\{F^h_{ir} | t \leq Q^h_r\}, & \text{si } t \leq max_{r \in R_h}\{Q^h_r\} \\ F^h_{i0}, & \text{otherwise} \end{cases} \quad (1)$$

Graph transformation

A modification is made in the original graph $G_0 = (V_0, E_0)$, that is defined as *transformed graph* and represents as $G = (V, E)$, where each airport are divided in as many nodes as flight itineraries owns. The application of transformed graph ensures there is an optimal solution of the problem where each vehicle delivers the load of the customers in the airport for each itineraries of flights at most once. The feasible solution that the new graph obtains, can be transferred to the original graph.

3. Problem formulation

The main objective of this problem is the global optimization of costs. Customers' loads are picked up, and flights are assigned por each customer for the transport of these loads.

Models with three index

In the second chapter of this report, we present the two models proposed by Azadian that differ from one to another in the set of constraints against possibles symmetries of the solution.

Models with two index

Besides, a model is proposed where the nodes associated with flight itineraries are duplicated for a number of times, following the same procedure to obtain the transformed graph, and is particularized for $m = 2$ and $m = 1$.

Model with a variable for each route

At the end of this chapter, a model based on the Branch-and-Price algorithm is studied.

4. Computational experiments

In the third chapter, the models studied before are compared applying them to set of instances with 360 possible situations where the ATD-PDP, using the IBM ILOG CPLEX Optimization Studio mathematical programming software.

5. Conclusions

Finally, The fourth is a chapter dedicated to the conclusions obtained during the work developed for the realization of this report.

References

[1] F. Azadian, A. Murat, R-B. Chinnam. *An Unpaired Pickup and Delivery Problem with Time Dependent Assignment Cost: Application in Air Cargo Transportatio.* European Journal of Operational Research (2017)

[2] F. Azadian. *An integrated framework for freight forwarders:exploitation of dynamic information for multimodal transportation.* Wayne State University Dissertations (2012) 496.

[3] J-J. Salazar González. *Programación matemática.* Ediciones Díaz de Santos (2001)

[4] P. Toth, D. Vigo. *Models, relaxations and exact approaches for the capacitated vehicle routing problem.* Discrete Applied Mathematics 123 (2002) 487-512.

[5] IBM ILOG CPLEX Optimization Studio *Getting Started with the IDE*