Universidad
de La Laguna

Problemas de Rutas de Vehículos con Capacidades
Métodos exactos de resolución

Capacitated Vehicle Routing Problems
Exact solving methods

Juan Francisco Pérez Caballero

Trabajo de Fin de Grado

Facultad de Matemáticas

Universidad de La Laguna

La Laguna, 15 de julio de 2014

Dr. D. **Juan José Salazar Gonzalez**, con N.I.F. 43356435D catedrático de universidady Dr. D. **Hipólito Hernández Pérez**, con N.I.F. 45452715T profesor de la universidad, adscritos al Departamento de Matemáticas, estadística e investigación operativa de la Universidad de La Laguna

C E R T I F I C A N

Que la presente memoria titulada:

"*Problemas de Rutas de Vehículos con capacidades. Métodos exactos de resolución*"

ha sido realizada bajo su dirección por D. **Juan Francisco Pérez Caballero**, con N.I.F. 44749045S.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 15 de julio de 2014

Agradecimientos

a Hipólito y Salazar
por haber aceptado y dirigido este trabajo

a Vicky
por haber hecho todo esto juntos

y a todos los que me han aguantado y apoyado este tiempo.

Resumen

En este Trabajo Fin de Grado se ha abordado uno de los sectores más interdisciplinarios de las matemáticas, la programación matemática y, más concretamente, un problema específico llamado Problema de Rutas de Vehículos con Capacidades (CVRP por sus siglas en ingles). El objetivo del trabajo es conocer los métodos de resolución de este tipo de problemas que se utilizan hoy en día y las líneas de investigación que existen con respecto a ello.

Este trabajo puede ser considerado como una ampliación de los conocimientos adquiridos en las asignaturas de Optimización y Programación combinatoria del Grado en Matemáticas de la Universidad de La Laguna, ya que profundiza en los métodos de resolución de problemas enteros y la modelización de los mismos con el objetivo de ejecutarlos en un programa informático.

Por otro lado, cabe destacar que este trabajo se planteó con el objetivo de ratificar que el método de ramificación y corte es un método eficaz a la hora de resolver problemas considerablemente grandes. Para ello, se describen los algoritmos necesarios para su implementación y el código correspondiente en lenguaje MOSEL. También este trabajo sirve para familiarizarse con el mercado de resolutores de programación matemática actual.

Por último presentaremos, además de los resultados computacionales, unas posibles futuras líneas de investigación para continuar con temas relacionados y profundizar en la optimización combinatoria en futuros Trabajos Fin de Grado o Tesis de Máster y Doctorado.

Palabras clave: CVRP, Ramificación y Corte, Programación Matemática, Mosel, Optimización

Abstract

In this Bachelor Thesis has been addressed one of the most interdisciplinary areas of mathematics, mathematical programming and specifically, a problem called Capacitated Vehicle Routing Problem (CVRP). The aim of the study was to determine the methods of solving these problems that are used and the research line that exist nowadays.

This work may also be considered as an extension of the knowledge acquired in the subjects called Optimization and Combinatorial Programming forming part of Degree in Mathematics from the University of La Laguna, as he delves into the methods of problem solving and modeling in order to execute them in a computer program.

On the other hand, it should also be mentioned that this report was proposed in order to confirm that the branch and cut method is effective in solving problems considerably large. For this, the algorithms needed to implement the model in a programming language called MOSEL and the correspondent code are described in this report. This work also serves to familiarize with the facts involving current lp-solvers market.

Finally we will present, in addition to the computational results, a possible future research lines to continue and deepen in a topic related to combinatorial optimization in incoming Bachelor, Master or PhD Thesis.

Keywords: *CVRP, Brach and Cut, Mathematical Programming, Mosel, Optimization*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y plantemiento del problema

La optimización combinatoria, también llamada programación matemática, programación combinatoria, investigación operativa o optimización, es una de las ramas de la matemáticas a las que más investigadores se dedican hoy en día. Esto es debido a la gran aplicabilidad que tiene y a su interdisciplinariedad, es decir, es un sector no solo investigado por matemáticos, sino también por ingenieros y economistas para mejorar el funcionamiento de sistemas de gestión del día a día y establecer mejoras. De forma general se puede definir la optimización combinatoria el minimizar o maximizar una determinada función objetivo cumpliendo un determinado número de restricciones.

Sin embargo, en este trabajo se tratará la optimización combinatoria desde un punto de vista matemático. La interdisciplinariedad en este caso viene dada por la influencia de diversos ámbitos matemáticos ampliamente conocidos como puede ser el álgebra de poliedros, utilizados en la base del método simplex de resolución de problemas, o la informática y computación para llevar a cabo métodos de resolución para problemas de tamaños que no se podrían resolver fácilmente sin ayuda del ordenador. Si tenemos en cuenta estos diversos campos, es necesario tener un conocimiento suficientemente amplio de las matemáticas en general y, por lo tanto, la aportación de los matemáticos al mundo de la programación combinatoria es imprescindible.

Estos hechos se pueden ver claramente en la constitución de grupos de trabajos en ámbitos aparentemente no relacionados con las matemáticas en sí mismas, como pueden ser la logística, transporte o psicología (en todos ellos es necesaria la programación matemática). Todo esto influyó en la motivación para la elección de la línea de proyecto y la elección del tema en concreto. Además, otra de las motivaciones

determinantes, es la falta de material accesible con información computacional sobre la resolución de problemas de rutas de vehículos.

Tras haber elegido el proyecto, se plantearon ciertos objetivos a cumplir en su elaboración, como son:

- Familiarizarse con software que ayude a la resolución de problemas.
- Proporcionar códigos que se puedan ejecutar en el software elegido.
- Conocer en profundidad algunos métodos de resolución existente en la actualidad.
- Conseguir resultados experimentales que ratifiquen la teoría sobre los métodos de resolución exactos más eficaces.
- Sentar las bases de futuros trabajos que continúen la investigación.

Cabe destacar que, aunque hubieron objetivos que no se plantearon en el inicio de la investigación, sí se consiguieron obtener resultados como son la influencia de los heurísticos utilizados por los programas informáticos en la eficiencia de un algoritmo y la tolerancia que tiene los mismos ante posibles errores en datos. Todo esto influyó en los resultados que mostraremos.

1.1. Definición de problemas de rutas de vehículos

Como ya se ha mencionado, en este trabajo se tratará el problema de las rutas de vehículos (*Vehicule Routing Problem* o *VRP* por sus siglas en inglés) y sus posibles métodos de resolución, para posteriormente intentar determinar cuál de ellos es el más eficiente. Más concretamente, nos centraremos en una variante de los VRP llamada problemas de rutas de vehículos con capacidades, o *CVRP* (*capacitated vehicle routing problem*). Para comenzar a estudiarlos veamos su historia y definición.

El problema de rutas de vehículos tiene sus orígenes en el año 1956 y fue propuesto por Dantzing y Ramser cuando estudiaron una aplicación real en la distribución de gasolina para estaciones de carburante. Este problema manejaba conceptos de rutas y grafos utilizados en problemas estudiados previamente y ampliamente conocidos como el *problema del viajante de comercios (TSP)* y conceptos asociados al almacenamiento de paquetes en contenedores, mas conocido como el *Bin Packing Problem*. De manera general se define el problema VRP como:

"La determinación de la ruta óptima para una flota de vehículos que parten de uno o más depósitos (almacenes) para satisfacer la demanda de varios clientes dispersados geográficamente."

Un esquema básico correspondiente a un VRP y a su solución óptima, que ayuda a entender claramente la definición del problema es el mostrado en la figura 1.1.

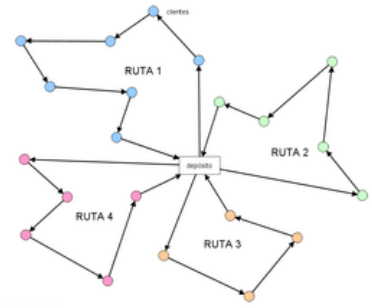

Figura 1.1: Esquema básico de VRP.

Cabe destacar que la optimalidad mencionada puede tener diversas interpretaciones. En la mayoría de los casos teóricos de estudio se utiliza la distancia euclídea total recorrida como coste a minimizar, pero también se puede usar cualquier tipo de distancia, tiempos de duración o diversos costes económicos (gasolina, salarios, etc.).

Además, si partimos de la base de la definición del VRP mencionada, podemos decir que tiene muchas variantes según las posibles consideraciones extra que tengamos que satisfacer. Veamos otros problemas derivados resumidos en la tabla 1.1. Como ya hemos dicho, en este trabajo nos centraremos en el CVRP, por lo cual a partir de ahora nos referiremos al problema directamente de las rutas de vehículos con las capacidades.

1.2. Resolutores. Comerciales y de código libre

Uno de los principales factores que influyen en la elección de la modelización de problemas utilizando programación matemática es la posesión de un software que permita que el ordenador resuelva problemas. Estos softwares, llamados "*solvers*"o *resolutores*, son importantes ya que, al permitir resolver problemas de grandes cantidades de datos, se puede probar la eficiencia de un algoritmo creado previamente y comparar con otros ya existentes para cantidades de datos especificas.

Nombre	**Siglas**	**Suposisciones extra**
Capacitated VRP	CVRP	Capacidad asociada a los vehículos (capacidad homgénea o heterogénea).
Capacitated VRP with time windows	CVRPTW	Capacidad asociada a los vehículos y ventanas de tiempo de llegada a clientes.
Vehicle Routing Problem with Pick-up and Deliveries	VRPPD	Posibilidad de entrega o recogida en los clientes.
Capacitated VRP with Pick-up and Deliveries and Time Windows	CVRPPDTW	Capacidad asociada a los vehículos, ventanas de tiempo de llegada a clientes y posibilidad de entrega o recogida en los clientes.
Multiple Depot VRP	MDVRP	Varios almacenes.
Multiple Depot VRP with Time Windows	MDVRPTW	Varios almacenes y y ventanas de tiempo de llegada a clientes.
Periodic VRP	PVRP	Necesidad de organización periódica.
Periodic VRP with Time Windows	PVRPTW	Necesidad de organización periódica y ventanas de tiempo de llegada a clientes.
Split Delivery VRP	SDVRP	Permite ser servido desde varios vehículos.
Split Delivery VRP with Time Windows	SDVRPTW	Permite ser servido desde varios vehículos y ventanas de tiempo de llegada a clientes.

Tabla 1.1: Variantes del VRP

Estos resolutores pueden venir acompañados de un entorno gráfico que permita, tanto programar de manera fácil y dinámica los algoritmos diseñados, como expresar de manera visualmente cómoda los resultados finales y de cada una de las iteraciones que deseemos. En general podemos decir que la diferencia entre dos resolutores distintos radica en la forma de implementar el algoritmo de resolución y los posibles métodos heurísticos que tengan definidos para ayudar a la resolución de los problemas.

Hemos realizado un pequeño análisis de realidad para intentar elegir el software que utilizamos a lo largo del desarrollo de la investigación. De manera general podemos decir que existen dos tipos de resolutores, resolutores comerciales y resolutores no comerciales o de código libre.

Nota: Decimos que un entorno gráfico es un programa que ayuda a la programación de un cierto problema, su ejecución y visualización de resultados. Los resolutores se pueden ejecutar también desde un programa en un lenguaje de programación, siendo el más común el C++.

- **Resolutores comerciales**

  Los resolutores comerciales se caracterizan por poseer entornos gráficos de gran utilidad, versatilidad y fáciles de aprender. Por otro lado, poseen gran cantidad de heurísticos implementados que ayudan a resolver de manera más rápida los problemas. Además suelen tener versiones para ejecutar el programa

en cualquier sistema operativo.

Las desventajas de este tipo de resolutores viene dada por, como su nombre indica, el coste para adquirirlos. Estos tipos de softwares están desarrollados por entidades privadas que necesitan obtener un beneficio económico de su producto, siendo por ello necesario un gasto monetario para obtenerlos. Cabe destacar que la mayoría de estas compañías poseen versiones de prueba, con capacidades máximas en el número de variables y de restricciones. También se pueden obtener licencias completas y gratuitas si se certifica que se va a usar con fines de investigación a través de una universidad, pero estas licencias son limitadas en el tiempo y no permiten acceder al código del resolutor.

Algunos ejemplos de este tipo de resolutores son el CPLEX (propiedad de IBM), XPRESS (propiedad de FICO), MINOS 5.5 y el solver de Excel (aunque este último no tiene gran capacidad).

- **Resolutores de código libre**

  Los resolutores de código libre son gratuitos y se puede acceder a su código. Desde un punto de vista meramente informático, este acceso permite el cambio y desarrollo de los mismos. Además de esto, no tienen ningún tipo de capacidad máxima para el número de restricciones y de varibles que se asocien al problema.

  Estos resolutores están menos desarrollados que los comerciales, lo que provoca mayor tiempo de ejecución y menos versatilidad a la hora de mostrar los resultados y los análisis de sensitividad.

  Algunos ejemplos de resolutores de código libre son GLPK y lp_solve (desarrollados por el proyecto GNU) y ipopt.

En la figura 1.2 se muestra un ejemplo comparativo de varios resolutores según su tiempo de ejecución.

También existen otros tipos de clasificaciones de los resolutores existentes, como puede ser si resuelven programación no lineal (en cuyo caso que tipo de no linealidad y si se encuentra en la función objetivo o en las restricciones), si resuelven programación entera y binaria, etc.

Cabe destacar la existencia de entornos gráficos, como el AMPL, que son capaces de integrar multitud de resolutores diferentes y de distintas empresas. Este entorno

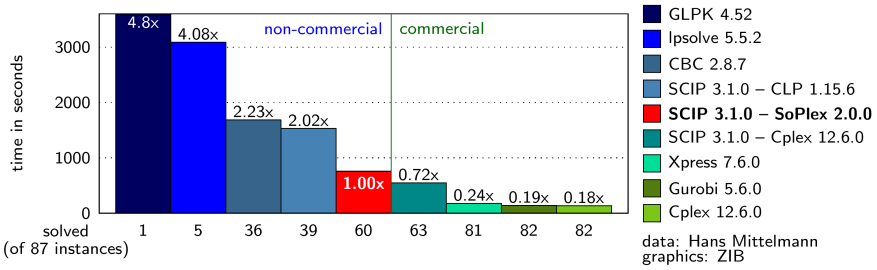

Figura 1.2: Comparación de resolutores con respecto al tiempo de ejecución

gráfico fue una de las principales motivaciones a la hora de elegir la línea de este trabajo, ya que se había estudiado previamente y se le podía sacar mucho partido. Por ello, fue la primera opción como entorno gráfico elegido a la hora de investigar junto con el CPLEX como resolutor. El problema surgió a la hora de intentar evitar el número máximo de restricciones y variables. A través de la página oficial del CPLEX se averiguó que se podía conseguir una licencia anual sin límite de restricciones ni de variables, pero, por el contrario, no se consiguió evitar las restricciones impuestas por el entorno gráfico AMPL, siendo necesario pagar para ello y desechando consecuentemente esta opción.

Posteriormente encontramos otro entorno gráfico llamado GUSEK, que era completamente libre y que utilizaba el GLPK (de código abierto) como resolutor asociado. Cada prueba que hicimos con los resolutores nos sirvió para familiarizarnos con los problemas de programación matemática que existen y que utilizan como ejemplo. Algunos de dichos problemas los veremos en el capítulo 2.

Finalmente decidimos utilizar el entorno gráfico llamado MOSEL, que utiliza el resolutor llamado XPRESS. Este entorno tiene un lenguaje de programación parecido al C++, permitiendo la ejecución secuencial y la adición de restricciones dinámicamente (necesario para el "branch and cut"). Además, presenta los resultados de forma clara, dibujándolos en un gráfico, y especificando los tiempos de ejecución. Otra de las razones por las que nos decidimos por este resolutor y entorno gráfico, es la posibilidad que ofrecía de una versión sin límite de restricciones por un año, dedicada a investigadores.

A modo de resumen podemos decir que existe una gran variedad de resolutores en el mercado y que, si se pretende investigar sobre programación matemática, se debería estudiar los resolutores existentes para elegir el más adecuado.

Capítulo 2

Fundamentos teóricos generales y notación

En este capítulo hablaremos sobre los principios teóricos utilizados en la resolución del problema mencionado. Además, este capítulo servirá para ponernos en situación para la posterior modelización y las necesidades que tendremos que satisfacer para resolver nuestro problema de rutas de vehículos.

Empezaremos dando una breve definición de lo que es la programación matemática y el método del simplex. Se explicarán también los métodos exactos de resolución de problemas lineales enteros (puros o mixtos), para acabar con una breve introducción a la teoría de grafos, la cual supone una herramienta muy útil a la hora de modelizar problemas como el que trataremos. Para más información sobre estos temas, podemos recurrir a uno de los muchos libros sobre programación matemática y teoría de grafos como por ejemplo Salazar [4].

2.1. Programación matemática. Poliedros y Simplex

La programación matemática es la rama de la investigación operativa que trata de identificar un punto extremo (máximo o mínimo) de una función, a la que llamaremos función objetivo o función a optimizar, dentro de un conjunto S de posibles soluciones.

La programación matemática la podemos dividir en dos grandes campos, programación lineal y programación no lineal dependiendo, si la función objetivo es lineal y el conjunto S se puede describir como un conjunto de desigualdades lineales o, por el contrario, esto no es posible.

La parte de la programación lineal en la que existe la restricción adicional de que las variables (o parte de ellas) tienen que ser enteras se conoce como programación lineal entera. En general, esta restricción hace que un problema sea mucho más difícil de resolver.

Un problema de programación lineal puede ser siempre expresado en su forma estándar, añadiendo las variables necesarias (llamadas de holgura) y haciendo los despejes oportunos. Para poder determinar la forma estándar, llamaremos:

- $c \in \mathbb{R}^n \equiv$ vector de costes.
- $b \in \mathbb{R}^m \equiv$ vector de recursos.
- $x \in \mathbb{R}^n \equiv$ vector de variables.
- $A \in M_{m \times n}(\mathbb{R}) \equiv$ matriz de coeficientes tecnológicos.

La forma estándar de un problema de programación lineal queda definida, por tanto:

$$\begin{array}{ll} \text{mín} & c^t x \\ s.a. & Ax = b \\ & x \geq 0 \end{array}$$

Es fácil ver que el conjunto de soluciones factibles de un problema de programación lineal forma un poliedro n-dimensional (poliedro: elemento geométrico de muchas caras) ya que dicho conjunto es de la forma:

$$S = \{x \in \mathbb{R}^n | Ax = b \wedge x \geq 0\}$$

El método simplex es un método propuesto por G.B. Dantzig en 1947 de resolución de problemas de programación lineal que se basa en esta caracterización de los extremos de un poliedro, partiendo de una solución factible inicial y buscando otros extremos en las posibles direcciones de búsqueda dentro de ese poliedro.

Como este método no es objeto de estudio en este trabajo, no nos detendremos en las proposiciones necesarias para demostrarlo, remitimos a uno de los muchos libros, como Salazar [4], simplemente comentar que aunque el método simplex tiene una complejidad algoritmica no polinomial (NP), lo cual significa que la resolución de problemas lineales es considerada "no fácil", se suele considerar que su complejidad es polinomial ya que la mayoría de los problemas resueltos por el simplex son en tiempo polinomial. Existen otros métodos y algoritmos como el algoritmo del elipsoide interior que sí es conocido que tiene complejidad polinomial.

Ejemplos de problemas de programación lineal.

Veamos algunos ejemplos de problemas que se resuelven mediante programación matematica y su expresión en la forma estándar, aunque muchos de ellos podrían suponer una investigación exaustiva. Estos ejemplos sirven también para familiarizarse a la hora de programar problemas de programación lineal, ya que son fácilmente entendibles.

1. **Problema de Producción**
   Este problema se caracteriza por la producción (creación, manipulación, etc.) de varios productos. Dicha producción está dividida en varias etapas, por las que debe pasar los productos durante un tiempo específico y, además, cada una de ellas con una cantidad máxima de horas disponibles. Existen costes asociados y cantidades mínimas a producir. Un ejemplo de datos sería el siguiente, con 3 productos distintos y 2 etapas de producción:

	Producto A	**Producto B**	**Producto C**	**Horas**
Etapa 1/aparato	10	15	20	400
Etapa 2/aparato	4	5	5	120
Costes/aparato	375	325	675	
Demanda/aparato	10	7	5	

Tabla 2.1: Ejemplo de datos problema de producción

   Finalmente, para expresar el problema en su forma estándar, quedaría $c^t = (275, 325, 675)$, $x^t = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$, siendo las tres primeras variables las correspondientes a la producción y las últimas correspondientes a variables de holgura,

$$A = \begin{pmatrix} 10 & 15 & 20 & 1 & 0 & 0 & 0 & 0 \\ 4 & 5 & 5 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 400 \\ 120 \\ 10 \\ 7 \\ 5 \end{pmatrix}$$

2. **Problema de la mochila (Knapsack Problem)**
   El problema consiste en intentar introducir el mayor valor (v_j) de determinados objetos (x_j) en un recipiente, teniendo en cuenta que existe una capacidad máxima (W) del recipiente y cada objeto tiene un volumen (w_j). De forma general, el problema es un problema de programación lineal entera y quedaría establecido como:

$$\begin{array}{lrcl} \text{máx} & \sum_{j=1}^{n} v_j x_j & & \\ s.a. & & & \\ & \sum_{j=1}^{n} w_j x_j & \leq & W \\ & x_j & \in & \{0,1\} \forall j \end{array}$$

2.2. Teoría de grafos

La teoría de grafos es una herramienta muy útil en la descripción de problemas de rutas ya que permite representar problemas aparentemente distintos utilizando un mismo método, es decir, un mismo lenguaje común a todos los problemas. Debido a esto, utilizaremos los conceptos de la teoría de grafos como notación de los problemas y siempre tenderemos a expresar un problema dado utilizando esta notación, para ayudar a decidir el método de resolución.

Dada la importancia mencionada de la teoría de grafos, vamos a dar algunas definiciones que luego ayudarán a la modelización y expresión de problemas en futuros capítulos. Existen dos tipos de grafos:.

- **Grafos dirigidos**

  Definición 2.2.1. *Un Grafo dirigido se denota por $G = (V, A)$, donde V es un conjunto no vacío de elementos denominados vértices y $A \subset V \times V$ es un conjunto de arcos.*

  En este tipo de grafos, a cada elemento $a \in A$ lo llamaremos arco, que tiene asociados dos vértices de V, i y j ($i \neq j$). A i lo llamamos origen del arco y a j destino del arco, el arco a también se denotará por (i, j).

  Definición 2.2.2. *Sea $G = (V, A)$, grafo dirigido y $S \subset V$, se definen:*

  $$\delta^+(S) := \{(i,j) \in A : i \in S, j \notin S\}.$$

  $$\delta^-(S) := \{(i,j) \in A : i \notin S, j \in S\}.$$

  $$A(S) := \{(i,j) \in A : i \in S, j \in S\}.$$

  Para simplificar la notación y hacer más sencilla la representación, si el subconjunto S está formado por un único elemento $i \in A$ en lugar de $\delta^+(\{i\})$ escribiremos $\delta^+(i)$. Análogamente, hacemos el mismo cambio para $\delta^-(\{i\})$, en cuyo caso escribiremos $\delta^-(i)$.

- **Grafos no dirigidos**

  Definición 2.2.3. *Un Grafo no dirigido se denota por $G = (V, E)$, donde V es un conjunto no vacío de elementos denominados vértices y $E \subset V \times V$ es un conjunto de aristas.*

  A cada elemento $e \in E$ lo llamaremos arista y lo denotamos por [i, j] donde $i, j \in V, i \notin j$ son los vértices incidentes de la arista e, entendiendo que [i, j] = [j, i], y por ello sus costes asociados también son iguales.

  Definición 2.2.4. *Sea $G = (V, E)$, grafo no dirigido y $S \subset V$, se define:*

$$\delta(S) := \{[i, j] \in E : i \in S, j \notin S\}$$

$$E(S) := \{[i, j] \in E : i \in S, j \in S\}.$$

  Para simplificar la notación y hacer más sencilla la representación, si el subconjunto S está formado por un único elemento $i \in E$ en lugar de $\delta(\{i\})$ escribiremos $\delta(i)$.

Por otro lado, la importancia de crear un grafo reside en asociar un coste a cada arco. Estos costes pueden venir dados como datos propios de un problema o como una distancia, ya sea euclídea, esférica, etc. A estos tipos de grafos con datos asociados s eles llama grafos pesados o redes.

Existen otras definiciones de la teoría de grafos que son de gran utilidad para nuestros propósitos. Sea un grafo $G = (V, A)$,

Definición 2.2.5. *El grado de un vértice $i = |\delta(i)|$.*

Definición 2.2.6. *Un camino es un conjunto de aristas $P := [i_1, i_2], [i_2, i_3], ...[i_{m-1}, i_m]$ de A, diremos que el camino une a i_1 y i_m. Cuando $i_1 = i_m$ lo llamaremos ciclo o circuito.*

Definición 2.2.7. *Un grafo es conexo si $\forall i, j \in V, \exists P$ camino que los une.*

Definición 2.2.8. *Un ciclo Hamiltoniano $T \subset E$ es un ciclo que incluye todos los vértices una sola vez.*

Definición 2.2.9. *Una arborescencia de raíz a es un grafo dirigido (V,A) en el que se cumple:*

1. $\forall i \in V \backslash \{a\} : |\delta^-(i)| = 1$.
2. $|\delta^-(a)| = 0$.

3. El grafo G no contiene ningún circuito.

Llamamos hijos del nodo i a $\delta^+(i)$ *y padre de i a* $\delta^-(i)$

Definición 2.2.10. *Sea un grafo* $G = (V, A)$ *dirigido y sean* $s, t \in V$*, definimos el corte (S,T) asociado al grafo G y los nodos s y t como el conjunto de arcos* $F \subset A$ *tales que en el grafo* $G' = (V, A \backslash F)$ *no existe un camino de s a t, siendo* $S \subset V$ *el conjunto de nodos alcanzables desde s y* $T = V \backslash S$*. Si existen costes asociados a los arcos, el corte se dirá mínimo si la suma de los de los arcos de F, a lo que llamaremos capacidad del corte, es el menor posible.*

Definición 2.2.11. *Sea un grafo* $G = (V, A)$ *dirigido con capacidades máximas asociadas a cada arco y sean* $s, t \in V$*, definimos flujo máximo entre s y t al mayor número de unidades que se pueden transportar desde s a t sin sobrepasar las capacidades de los arcos.*

Teorema 2.2.1 (**Teorema del flujo máximo - corte mínimo**)**.** *En cualquier grafo, el valor del flujo máximo entre dos vértices s y t coincide con la capacidad del corte mínimo entre los mismos nodos.*

El problema del flujo máximo entre s y t se resuelve de manera sencilla y con complejidad polinomial mediante el siguiente modelo de programación lineal, siendo f_{ij} el flujo asociado a cada arco y c_{ij} la capacidad máxima de cada arco (deben ser todas positivas):

$$
\begin{array}{lccll}
 & \text{máx} & f_{ts} & & \\
s.a. & & & & \\
 & \sum_{a \in \delta^-(i)} f_a & = & \sum_{a \in \delta^+(i)} f_a & \forall i \in V \\
 & f_{ij} & \leq & c_{ij} & \forall (i,j) \in A
\end{array}
$$

2.3. Programación lineal entera

La programación lineal entera o simplemente programación entera, como ya definimos previamente en la sección 2.1, es un tipo de programación matemática en la que existe alguna restricción de integralidad, es decir, obligamos a que un subconjuto de variables sean enteras. En el caso de que todas las variables que se involucren en el problema sean enteras, diremos que trabajaremos con programación lineal entera pura, sin embargo, cuando no todas las variables tengan que ser necesariamente enteras, lo llamaremos programación lineal entera mixta o MIP (Mixed Integer Programming). Es más común trabajar con la programación entera mixta debido a que generaliza el otro tipo. Mas adelante en este trabajo, nos centraremos en un tipo de programación entera en la que las variabels solo pueden tomar los valores 0 o 1, y

por ello las llamaremos variables binarias.

Aunque la programación lineal tiene métodos de resolución de complejidad polinomial (P), cuando hacemos frente a la programación entera nos encontramos con un problema con complejidad no Polinomial (NP-duro). Debido a esta complejidad, se ha investigado mucho sobre los métodos de resolución de programación entera y cuál de ellos es mejor en función del problema.

A continuación vamos a explicar algunos de estos métodos de resolución como son el método de ramificación y acotación (Branch and Bound) y el de hiperplanos de corte (Cutting Planes), para luego introducir el método de ramificación y corte (Branch and cut), que es una mezcla de los anteriores y que, por lo general, da mejores resultados. Existen otros métodos como la programación dinámica que también se utilizan para resolver problemas con un mayor número de restricciones que el problema que nos ocupa a nosotros, el CVRP.

2.3.1. Ramificación y acotación

Aunque la programación matemática había sido investigada desde varios años antes, no fue hasta 1960 cuando Land y Boig formularon el primer algoritmo de Ramificación y Acotación e introdujeron esta técnica de manera formal, dando una solución genérica por primera vez al problema de integralidad.

Para definir este método de manera sencilla, podemos decir que se basa en la división progresiva de la región factible del problema, investigando cada división por separado y aprovechando los resultados de las mismas como cotas de la solución. Debido a esto, podemos englobar este método dentro de los llamados métodos enumerativos. Las divisiones de la región factible se van realizando a través de la adición de restricciones que acoten las variables, es decir, añadimos restricciones del tipo $x_i \leq a$ y $x_i \geq b$ al problema y creamos dos problemas nuevos, y así sucesivamente hasta que encontremos la solución. Los valores a y b de las restricciones añadidas corresponden a la parte entera superior e inferior de la solución óptima de una variable (x_i^*) del problema predecesor, quedando definitivamente las restricciones como:

$$\begin{aligned} x_i &\leq \lfloor x_i^* \rfloor \\ x_i &\geq \lceil x_i^* \rceil \end{aligned} \tag{2.1}$$

Debido a esto, podemos representar este método a través de un grafo tipo árbol binario (cada nodo tiene 0 o 2 hijos) en el que cada nodo representa un subproblema del problema de programación entera originario y en el que cada hijo posee una

restricción más que su padre. La raíz del árbol la ocupa el problema relajado sin la restricción de integralidad.

Definición 2.3.1. *Definimos relajación de un problema como el problema resultante a eliminar ciertas restricciones. Los problemas relajados tienen un valor de función objetivo menor que el problema original si la función objetivo es del tipo minimizar.*

En definitiva, el método o algoritmo de ramificación y acotación se basa en ir generando el grafo asociado al problema y considerar como cota superior de la solución óptima al valor menor de las soluciones factibles que se vayan encontrando y como cota inferior a las soluciones de los problemas que no sean factibles considerando la integralidad de las variables necesarias. El algoritmo termina cuando las cotas se encuentran. Cabe destacar que cuando un problema no tiene solución factible, no posee hijos ya que serían también problemas no factibles. También, si al resolver un problema tenemos un coste objetivo mayor que la cota superior, tampoco tiene hijos ya que la soluciones de los mismos siempre es mayor o igual. Este algoritmo también es capaz de detectar al no acotación o la no factibilidad del problema entero.

Un ejemplo de árbol asociado a un problema lo podemos ver en la figura 2.1, en la que vemos claramente el funcionamiento de las cotas y las "podas" árbol.

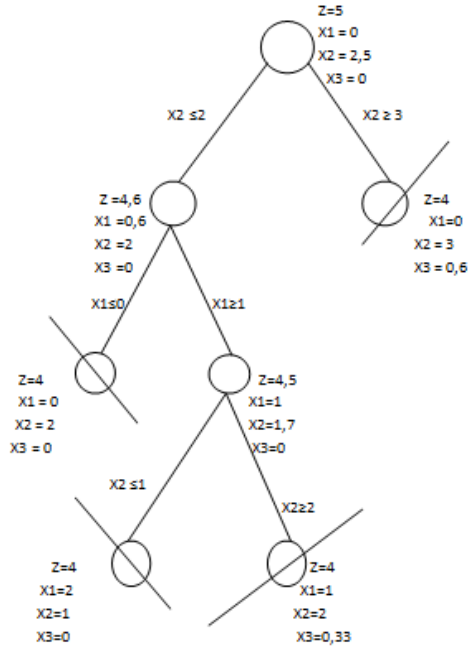

Figura 2.1: Ramificación y acotación. Ejemplo de árbol.

2.3.2. Hiperplanos de Corte

Veamos ahora el llamado método de hiperplanos de cortes, otro método de resolución de problemas de programación entera. Al contrario que el método de ramificación y acotación, este método no se considera de tipo enumerativo. Veamos una definición que nos ayude a entender mejor el método.

Definición 2.3.2. *Dado un poliedro P y un punto extremo x^* fraccionario de P, se llama corte a una desigualdad $d^t x \leq d_0$ válida para todos los puntos enteros de P violada por x^*.*

Definición 2.3.3. *Llamamos problema de separación asociado a un problema de programación matemática al problema de encontrar un corte del poliedro P asociado al mismo.*

El método se basa en, partiendo de un problema relajado del problema original, ir introduciendo cortes al poliedro asociado al problema que lo refine hasta encontrar uno cuya solución óptima tenga valores enteros en las variables necesarias.

El problema surge a la hora de encontrar el corte, es decir, a la hora de resolver el problema de separación, ya que puede suponer grandes esfuerzos computacionales ya que los cortes introducidos, aunque se puede asegurar que converjan hasta la solución entera óptima, puede necesitar un gran número de iteraciones.

Existen 2 tipos de métodos de hiperplanos de corte, de acuerdo con la naturaleza del problema de separación utilizado para encontrar el corte.

- **General:** Este tipo de método es aplicable a cualquier problema de programación lineal entera, es decir, es independiente a la naturaleza del problema y por lo tanto del poliedro. Uno de los métodos que pertenecen a este tipo es el llamado cortes de Gomory, que utiliza la *Derivación de Chvátal* para encontrar el corte. Gracias a este tipo de problemas podemos asegurar que este método es aplicable a cualquier problema, aunque sea necesaria una cantidad grande de tiempo.

- **Específico:** Estos métodos no son aplicables en cualquier caso ya que el problema de separación para encontrar el corte es específico para problemas de una naturaleza específica.

Un ejemplo gráfico de este método en el que se ve el refinamiento lo podemos ver en la figura 2.2.

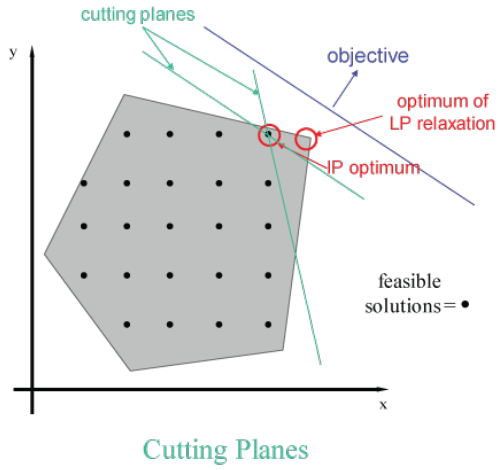

Figura 2.2: Hiperplanos de corte. Ejemplo de cortes.

2.3.3. Ramificación y corte

Introduciremos ahora un nuevo método de resolución de programación entera llamado ramificación y corte (Branch and Cut). Como ya se ha mencionado, esta técnica mejora las anteriores ya que mezcla conceptos de ambas. Este es el método que consideramos como mejor a la hora de resolver el problema del CVRP y lo probaremos desde un punto de vista computacional.

La necesidad de esta nueva técnica se da debido a que durante la utilización del método de hiperplanos de corte, explicado anteriormente, a medida que vamos aumentando el número de iteraciones sin llegar a la solución óptima, generalmente, los cortes comienzan a ser menos profundos, lo que significa que la diferencia entre un poliedro y su predecesor es insignificante. Por su parte el método de ramificación y acotación tiene la desventaja que es preciso el estudio de un gran número de nodos pertenecientes al árbol decisional, del orden de 2^n aproximadamente siendo n la profundidad del árbol. La principal razón por la que esto sucede es que la diferencia entre el valor óptimo del problema lineal relajado y la solución óptima del problema entero suele ser considerablemente grande.

En esencia, esta técnica se basa en, a lo largo del avance en el árbol originado por las ramificaciones de la misma manera que en el método de ramificación y acotación, ir introduciendo cortes en el nodo actual y sus posibles hijos que mejoren de manera significativa el gap (diferencia entre la cota inferior y superior) antes de cambiar de nodo, reduciendo así la profundidad del árbol y, por ello, el número total de nodos

a visitar.

Para introducir cortes es necesario resolver un problema de separación. Dicho problema de separación suele ser de tipo específico, es decir, cada problema de programación lineal suele tener un problema de separación asociado particular. Para que podamos considerar que este método es más rápido que los anteriores, es necesario encontrar un problema de separación de complejidad polinomial, es decir, capaz de resolverse en tiempo relativamente pequeños.

En resumen, el algoritmo se basa en, resolver el problema relajado, añadir un corte oportuno tras resolver el problema de separación y ramificar. Cada vez que resolvamos un problema relajado, es decir, cada vez que avancemos por el árbol de búsqueda, se añadirá un corte si es posible. Además, si es posible, se añadirán más de un corte en cada nodo del árbol. El procedimiento de poda del árbol de búsqueda es el mismo que para el caso de ramificación y acotación, es decir, jugamos con las cotas del problema.

NOTA: Al ir introduciendo cortes es recomendable revisar los anteriores porque puede que existan algunos que no tengan efectos, lo que se llama holgura positiva, y puede que existan algunos eliminados que puedan tener importancia en el futuro. Para ello, los sistemas informáticos guardan en memoria las restricciones, lo que puede provocar problemas de memoria a la hora de la ejecución.

Cabe destacar que existe un método parecido al método descrito pero que solo añade cortes en el primer nodo, es decir, en el problema relajado. A este método se le llama método de corte y ramificación (cut and branch) y se suelen usar los cortes de Gomory como método de introducción de cortes.

Capítulo 3

Modelos de Rutas de vehículos (VRP)

En este capítulo procederemos definitivamente al modelado matemático de los problemas de rutas de vehículos con capacidades (CVRP), teniendo en cuenta los métodos de resolución y la notación en forma de grafos explicados en el capítulo 2. Estudiaremos varios modelos para el problema de rutas de vehículos considerando el caso de capacidades, es decir, el CVRP (Capacitated Vehicle Routing Problem). Consideraremos que los vehículos recorrerán los puntos de demanda recogiendo un tipo exclusivo de objetos, es decir, englobaremos los modelos que representaremos a continuación dentro de los one-commodity flow formulations.

Empezaremos estudiando modelos que utilizan la eliminación de subciclos que no pasen por el depósito y de ciclos que no cumplen con las restricciones de capacidad de los vehículos. Dentro de este tipo, veremos dos formas de resolverlos, la primera de ellas resolviendo el problema muchas veces de manera iterativa, añadiendo en cada iteración las restricciones necesarias violadas hasta encontrar el óptimo, y la segunda de ellas utilizando el método de ramificación y corte, apoyándonos en la resolución del problema del corte mínimo como problema de separación. También formularemos el modelo de 3 índices, aunque no lo usaremos a la hora de comparar los tiempos de ejecución.

Seguidamente consideraremos los modelos con variables de flujo de una mercancía, empezando por el caso de los modelos de restricciones llamadas potenciales, para seguidamente tratar en los modelos considerados puramente de flujos, con dos índices y con tres índices. El caso de tres índices se puede usar para la situación de tener una flota no homogénea de vehículos, es decir, vehículos con distintas capacidades.

Es de especial atención también mencionar que solo estudiaremos los modelos

de costes asimétricos, es decir, el costo asociado de ir de una ciudad i a otra j no tiene porqué ser igual al de ir de j a i. Se han elegido estos modelos que generalizan todos los posibles casos. Existen modelos exclusivos para casos simétricos, pero son exclusivos para los mismos.

Convenciones para la notación

Vamos a dejar en claro algunas notaciones que utilizaremos en todos los modelos. Asociaremos los Problemas a un grafo dirigido $G = (V, A)$ (ya que habíamos dicho que íbamos a considerar los costes asimétricos), en el que V es el conjunto de ciudades o localizaciones y A es el conjunto de las conexiones existentes entre dichas ciudades o localizaciones, es decir, cada nodo representa un lugar a visitar y cada arista una conexión entre ellos. Podemos considerar que la solución representa un tipo de grafo Hamiltoniano en la que todo nodo es visitado una sola vez excepto el depósito. Otras notaciones que usaremos serán:

- $n \in \mathbb{N} \equiv$ Número de ciudades. $\Rightarrow V = \{1..n\}$
- $d \in V \equiv$ depósito o nodo raíz.
- $V_c := V \backslash \{d\} \equiv$ conjunto de clientes.
- $c_{ij} \in \mathbb{R}^+, \forall (i,j) \in A \equiv$ Costo asociado al arco $(i,j) \in A$.
- $q_i \in \mathbb{R}^+ \forall i \in V_c \equiv$ Demanda del nodo $i \in V_c$. El depósito, en general, no tiene demanda.
- $Q \in \mathbb{R} \equiv$ Capacidad máxima de los vehículos.
- $m \in \mathbb{N} \equiv$ Número de vehículos disponibles.
- $x_{i,j} \in \{0,1\}, \forall (i,j) \in A \equiv$ variable binaria de asignación asociada a cada arco. 1 si el arco (i,j) es usado en la solución y 0 en otro caso.
- $u_i \in \mathbb{R}, \forall i \in V \equiv$ carga en el vehículo a la salida de cada nodo.
- $f_{i,j} \in \mathbb{R}, \forall (i,j) \in A \equiv$ mercancía transportada por el vehículo a lo largo del arco $(i,j) \in A$.
- Sea $F \subset A \Rightarrow x(F) := \sum_{(i,j)\in F} x_{ij}$.
- Sea $S \subset V \Rightarrow q(S) := \sum_{i \in S} q_i$.
- Sea $F \subset A \Rightarrow f(S) := \sum_{(i,j)\in F} f_{ij}$.

3.1. Formulación con flujo de vehículos

El primer modelo que trataremos aprovecha las características de los subconjuntos de nodos del grafo, con respecto al numero de aristas incidentes.

3.1.1. Formulación de dos índices

Este modelo es atribuido normalmente a Laporte y Nobert [6]

$$\text{mín} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{3.1}$$

Sujeto a

$$x(\delta^+(i)) = 1 \quad \forall i \in V_c \tag{3.2}$$
$$x(\delta^-(i)) = 1 \quad \forall i \in V_c \tag{3.3}$$
$$x(\delta^+(d)) = x(\delta^-(d)) \leq m \tag{3.4}$$
$$x(\delta^+(S)) \geq \lceil q(S)/Q \rceil \quad \forall S \subset V_c \tag{3.5}$$
$$0 \leq x_{ij} \leq 1 \quad \forall (i,j) \in A \tag{3.6}$$
$$x_{ij} \in \mathbb{Z} \quad \forall (i,j) \in A \tag{3.7}$$
$$\tag{3.8}$$

Las familias de restricciones (3.2) y (3.3) son las llamadas restricciones de grado de salida y grado de entrada. Estas restricciones aseguran que cada nodo que no sea el nodo depósito sea visitado exactamente una vez ya que solo permite que tenga un arco entrante y uno saliente.

Por otro lado, las restricciones (3.4) hacen referencia a los arcos entrantes y salientes del nodo depósito, obligando a que sean los mismos que entren y los que salgan y que el numero de estos arcos sean, como mucho, el número de vehículos disponibles. Estas restricciones deberán ser representadas a través de dos diferentes a la hora de programar ya que no se permite dos igualdades o desigualdades en una misma sentencia.

Las desigualdades del tipo (3.6) y (3.7) hacen referencia a la binariedad de las variables decisión. La primera de las restricciones acotan las variables y las segundas las obligan a ser enteras. también es posible ponerlas como $x_{ij} \in \{0,1\}, \forall (i,j) \in A$, pero de la otra forma da lugar a relajaciones más precisas y ayuda al entendimiento de ciertos aspectos más adelante.

Las restricciones tipo (3.5) son las que evitan la presencia de subciclos no permitidos, es decir, subciclos que no pasen por el nodo depósito o que sobrepasen al capacidad máxima de los vehículos. Esto lo consigue mediante la obligación de sacar tantas aristas de un subconjunto $S \subset V$ dado como vehículos sean necesarios para satisfacer las demandas de los vértices del conjunto S sin sobrepasar al capacidad. Por ejemplo, si $q(S) = 15$ y $Q = 10 \Rightarrow x(\delta^+(S)) \geq \lceil 1{,}5 \rceil = 2$, es decir ,será necesario como mínimo dos vehículos que atraviesen este conjunto de nodos. La siguiente equivalencia es utilizada en algunos casos para resolver:

$$x(\delta^+(S)) \geq \lceil q(S)/Q \rceil \Leftrightarrow \sum_{i \in S, j \in S, i \neq j} x_{ij} \leq |S| - \lceil q(S)/Q \rceil$$

Para resolver el problema debemos tener en cuenta ciertos aspectos. A la hora de programar la adición de las restricciones de subciclo (3.5), nos encontramos con que no existen comandos en los lenguajes de programación que los permita implementar. Además, son un número excesivamente grande de restricciones, concretamente $2^n - 1$. Por ello, debemos buscar formulas alternativas de implementar este tipo de restricciones y resolver el problema.

Resolución Múltiple

Este método se basa en resolver el problema relajado resultante de no tener en cuenta las restricciones del tipo (3.5), es decir, las restricciones de eliminación de subciclos no válidos. Una vez resuelto, se procede a encontrar una restricción del mismo tipo violada y añadirla al conjunto de restricciones. Tras ser añadida, volver a resolver el problema con la nueva restricción. Este proceso se repite iterativamente hasta encontrar un óptimo que cumpla todos los requisitos de capacidad y subciclos permitido (todos deben contener al depósito como nodo), lo que significa que cumple todas las restricciones del tipo (3.5).

Un pseudocódigo recursivo de este algoritmo los podemos encontrar en el apéndice A, y una explicación rápida de dicho código sería:

1. Resolver el problema relajado correspondiente al modelo (3.1) - (3.4), (3.6) y (3.7).

2. En la solución óptima de dicho problema (que es entero), definir nodos siguientes a cada nodo en el grafo y encontrar los nodos siguientes al depósito.

3. Recorrer los ciclos que comienzan en los nodos siguientes al depósito. Si existe un ciclo que no cumpla con la restricción de capacidad ir al paso 4. Si no ir al paso 5.

4. Añadir al restricción del tipo (3.5) siendo S el conjunto de nodos del subciclo. Resolver e ir al paso 2.

5. Si existen nodos que no estén conectados con el depósito, tomar uno de ellos, recorrer el ciclo al que pertenezcan y añadir una restricción del tipo (3.5) con los nodos que los compongan, para posteriormente resolver nuevamente e ir al paso 2. Si no, hemos encontrado el óptimo.

Nota: En cada resolución se utiliza el método de ramificación y corte, porque es el que viene por defecto en los resolutores informáticos.

Resolución por ramificación y corte

Este método de resolución se basa en el método de ramificación y corte descrito en la sección 2.3.3. A la hora de encontrar cortes válidos para el poliedro, resulta que existen muchos tipos de desigualdades que cumple el poliedro asociado a las restricciones (3.3)-(3.6). Algunos de ellos son los siguientes, aunque no hay lugar para examinarlos todos y referenciamos los libros [1], [5] y [8].

- Desigualdades de capacidad fraccionaria:

$$x(\delta^+(S)) \geq \frac{q(S)}{Q}\,,(S \subset V_c). \tag{3.9}$$

- Desigualdades de eliminación de subciclos:

$$x(\delta^+(S)) \geq 1\,,(S \subset V_c). \tag{3.10}$$

- Desigualdades de Grandes Multiestrellas Generalizadas (*Generalized Large Multistar (GLM)*):

$$x(\delta^+(S)) \geq \frac{q(S)}{Q} + \frac{1}{Q}\sum_{j\in S}\sum_{j\in V_c\setminus S} q_j(x_{ij}+x_{ji})\,,(S \subset V_c). \tag{3.11}$$

- Desigualdades tipo *hipotour*:

$$x(A\setminus F) \geq 1 \tag{3.12}$$

  Donde $F \subset A$ es un conjunto de arcos para los que no hay solución factible para el CVRP utilizando únicamente los arcos de F.

Es fácil ver que las restricciones originarias dominan las de tipo capacidad fraccionaria y las de tipo eliminación de subciclos. Por otro lado, también es fácil ver que las desigualdades GLM dominan las de capacidad fraccionaria y también se puede ver que las desigualdades hypotour, generalizan las de eliminación de subciclos.

Sin embargo, el problema surge al intentar encontrar el subconjunto S o F que no cumpla una desigualdad una vez resuelto el problema relajado (modelo (3.1) - (3.4) y (3.6)). Para ello, se ha descubierto que, a través de un problema de corte mínimo, podemos encontrar dicho subconjunto necesitado. Este problema será capaz de encontrar tanto restricciones de tipo (3.5) como de capacidad fraccionaria. Veamos la formalización de dicho problema y como se ha llegado hasta él. Denotaremos por $\overline{x} \in \mathbb{R}$ al óptimo encontrado en el momento de aplicar el problema de separación.

Problema de separación: Sea $G_{\overline{x}} = (V, \overline{A})$ el grafo inducido por las componentes estrictamente positivas de $\overline{x}$ y por todos los arcos incidentes al depósito, es decir, $A = \{a \in A / \overline{x}_a > 0\} \bigcup \{(d,i) | i \in V_c\} \bigcup \{(i,d) | i \in V_c\}$. Asociamos costes a los arcos de la siguiente forma:

- $w_{ij} = \overline{x}_{ij} \in (0,1]$, $\forall i,j \in V_c / i \neq j$
- $w_{dj} = w_{jd} = \overline{x}_{ij} - q_i/Q \in [-1,1)$, $\forall j \in V_c$

Es fácil ver que si el corte mínimo (S,T), asociado al grafo $G_{\overline{x}}$ siendo $s = d$ tiene coste negativo para algún $t \in V_c$, entonces el conjunto T correspondiente al corte define una desigualdad fraccionaria (y por lo tanto también del tipo 3.5) violada.

Proposición 3.1.1. *Sea $G_{\overline{x}}$ el grafo definido anteriormente y sea $s = d$. Si existe un nodo en V que verifique que el corte mínimo (S,T) tiene costo negativo $\Rightarrow$ El conjunto T define una desigualdad fraccionaria violada.*

Demostración. Tenemos que ver que:

$$x(\delta^+(T)) < \frac{q(T)}{Q}$$

Sea $F \subset A$ el conjunto de arcos que produce el corte mínimo (S,T) y C su correspondiente capacidad, se verifican las siguientes relaciones.[1]

$$\left.\begin{array}{cc} x(F) = & x(\delta^-(T)) = x(\delta^+(T)) \\ x(F) = ^1 & C + \sum_{i \in T} q_i/Q = C + q(T)/Q \end{array}\right\} \Rightarrow x(\delta^+(T)) = C + q(T)/Q \Rightarrow$$

$$\Rightarrow x(\delta^+(T)) - q(T)/Q = C \tag{3.13}$$

[1]Debido a la definición de los nuevos costes.

Pero como, por hipótesis, se verifica que $C < 0$, entonces utilizando la relación 3.1.1:

$$x(\delta^+(T)) - q(T)/Q = C < 0 \Rightarrow x(\delta^+(T)) - q(T)/Q < 0 \Rightarrow x(\delta^+(T)) < q(T)/Q$$

□

Una vez sabemos este hecho, nos bastaría encontrar dicho corte. Sabemos por el teorema del flujo máximo corte mínimo que si redefinimos los costes del grafo mencionado anteriormente como capacidades, podríamos resolver el problema del flujo máximo en tiempo polinomial para encontrar el corte. el problema surge al existir capacidades negativas, por lo cual no podemos aplicar el problema del flujo.

Para solucionar este problema, creamos un nuevo grafo auxiliar $G'_{\overline{x}}$ igual obtenido de añadir a $G_{\overline{x}}$ un nodo extra $(n+1)$ conectado a todos los nodos de V_c. En este nuevo grafo los costes asociados a los arcos quedan definidos de la siguiente manera.

- $c_{ij} = \overline{x}_{ij} \in (0,1]$, $\forall i,j \in V_c / i \neq j$
- $c_{dj} = c_{jd} = \text{máx}\{0, \overline{x}_{ij} - q_i/Q\} \in [0,1)$, $\forall j \in V_c$
- $c_{(n+1),j} = c_{j,(n+1)} = \text{máx}\{0, q_i/Q - \overline{x}_{ij}\} \in [0,1)$, $\forall j \in V_c$

En este nuevo grafo $G'_{\overline{x}}$ tenemos todas los costes positivos, y por lo tanto capacidades para el problema del flujo máximo, por lo cual podemos resolver el problema del flujo máximo en tiempo polinomial. Para encontrar el subconjunto T correspondiente usamos un algoritmo de nodos alcanzables , cuyo código lo podemos encontrar en el apéndice A.

En McCormick, Rao y Rinaldi [9] se muestra que el corte mínimo (F') del grafo $G'_{\overline{x}}$ lleva asociado un corte minimo (F) del grafo $G_{\overline{x}}$ si quitamos los arcos que tengan como origen o destino al nodo n+1. Además, la capacidad del corte mínimo F' excede al capacidad del corte mínimo F en exáctamente $Z = \sum_{i \in V_c} \text{máx}\{0, q_i/Q - \overline{x}_{di}\}$. Debido a esto podemos concluir que:

- Si capacidad de $F' < Z \Rightarrow$ capacidad de F $< 0 \Rightarrow$ Hemos encontrado una desigualdad violada.
- capacidad de $F' \leq Z + 2 \Rightarrow$ la desigualdad fraccionaria asociada al corte está satisfecha, pero puede ser posible que la desigualdad redondeada no lo esté.
- Si la capacidad de $F' \geq Z + 2 \Rightarrow$ la restricción (3.5) asociada es satisfecha.

3.1.2. Formulación de 3 índices

Existe otra formulación basada en los mismos conceptos que la anterior pero considerando las variables de decisión x con tres índices, es decir, x_{ij}^k. Este nuevo índice indica qué vehículo es utilizado en el arco. Además debemos crear una nueva variable de paso. En conclusión:

- $x_{ij}^k \in \{0,1\}, \forall (i,j) \in A, k \in \{1,...,m\} \equiv 1$ si el vehículo k utiliza el arco (i,j), 0 en otro caso.
- $y_i^k \in \{0,1\}, \forall i \in V, k \in \{1,...,m\} \equiv 1$ si el vehículo k visita el vértice i, 0 en otro caso..

Cabe destacar que este modelo permite tener en cuenta el caso en el que las flotas de vehículos tienen capacidades heterogéneas, es decir, no todas las capacidades de los vehículos son iguales. Por ello, en lugar de tener una única capacidad Q, podemos decir que tenemos un conjunto de capacidades $Q_k, \forall k \in \{1,...,m\}$. El modelo quedaría por tanto:

$$\text{mín} \sum_{(i,j)\in A} c_{ij} \sum_{k=1}^{m} x_{ij}^k \tag{3.14}$$

Sujeto a

$$x^k(\delta^+(i)) = x^k(\delta^-(i)) = y_i^k \qquad \forall i \in V, k \in \{1,...,m\} \tag{3.15}$$

$$x^k(\delta^+(S)) \geq y_i^k \quad \forall k \in \{1,...,m\},\ S \subset V_c, i \in S \tag{3.16}$$

$$\sum_{i\in V_c} q_i y_i^k \leq Q_k \qquad \forall k \in \{1,...,m\} \tag{3.17}$$

$$\sum_{k=1}^{m} y_i^k = 1 \qquad \forall i \in V_c \tag{3.18}$$

$$x_{ij}^k \in \{0,1\} \qquad \forall k \in \{1,...,m\}, (i,j) \in A \tag{3.19}$$

$$y_i^k \in \{0,1\} \qquad k \in \{1,...,m\}, i \in V \tag{3.20}$$

Las restricciones tipo (3.15) significan que, si un vehículo entra en un nodo, también debe salir para conservar el flujo de vehículos. Las restricciones tipo (3.16) evitan subciclos no deseados en lo referente a subciclos que no pasen por el depósito. Esta restricción es una adaptación del conocido problema del viajante de comercio (TSP), el cual lleva una investigación aparte.

Por otro lado, las restricciones tipo (3.17) evitan que se sobrepasen las capacidades de los vehículos, ya que significa que la suma de las demandas de los nodos recorridos por un vehículo k debe ser menor que la capacidad de dicho vehículo Q_k.

Por último, las restricciones tipo (3.18) indican que los nodos son visitados solo por un vehículo, ya que no se permite la división de cargas entre varios vehículos y las restricciones (3.19) y (3.20) son las llamadas restricciones de binariedad.

A la hora de resolver este problema existen relajaciones pero que son muy débiles. Como el objetivo de este trabajo es estudiar el método de ramificación y corte, no nos centraremos en este modelo pero simplemente decir que a través de determinadas relaciones existen similitudes entre el poliedro asociado a este problema y el asociado al problema con dos índices, además de existir métodos de resolución relacionados con el TSP.

3.2. Formulación con flujo de mercancías

En esta sección trataremos de formalizar modelos en los que tenemos en cuenta la carga transportada por los vehículos. En este trabajo se tendrán en cuenta el modelo de potencial, el de flujo y el de multiflujo (o flujo de 3 índices) para el caso de un único tipo de mercancía, pero a la hora de resolver para obtener los resultados computacionales solo comprobaremos el potencial y el de flujo.

3.2.1. Modelo Potencial

En este caso utilizaremos, además de las variables decisión, las variables de tipo $u_i, \forall i \in V$, que representan la carga del vehículo a la salida de cada nodo. El modelo es el siguiente:

$$\text{mín} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{3.21}$$

Sujeto a

$$x(\delta^+(i)) = 1 \qquad \forall i \in V_c \tag{3.22}$$

$$x(\delta^-(i)) = 1 \qquad \forall i \in V_c \tag{3.23}$$

$$x(\delta^+(d)) = x(\delta^-(d)) \leq m \tag{3.24}$$

$$u_j \geq u_i + q_j - M(1 - x_{ij}) \quad \forall i \in V, j \in V_c : i \neq j \tag{3.25}$$

$$u_i \leq Q \qquad \forall i \in V \tag{3.26}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i,j) \in A \tag{3.27}$$

El significado de las restricciones (3.22) - (3.24) y (3.27) son el mismo que para el caso de flujo de vehículos con dos índices. Las restricciones tipo (3.26), por su lado son las que permiten que se cumplan las restricciones de capacidad ya que la variable u_i representa la carga del vehículo y las estamos mayorando por al capacidad.

Por otro lado, las restricciones tipo (3.25) son las que eliminan los subciclos no deseados (que no pasen por el depósito) y mantienen la conservación de flujo en los nodos. El valor M representa un valor suficientemente grande. En el caso de estar entregando mercancía en lugar de recogerla, simplemente cambiaríamos la desigualdad $\geq$ por $\leq$ y el signo de q_j y M.

Proposición 3.2.1. *Si $\overline{x}_{ij}$ es una solución a un problema de CVRP que cumple la restricción*

$$u_j \geq u_i + q_j - M(1 - x_{ij});\ \forall i \in V, j \in V_c : i \neq j$$

Para una M suficientemente grande $\Rightarrow \nexists$ ciclos que no pasen por el depósito.

Demostración. Para demostrarlo usaremos el método de reducción al absurdo. Para simplificar el problema, supondremos que existe un ciclo de 3 nodos que no contiene al depósito. Para ciclos de cardinal mayor, la demostración se procedería igual.

Sean $i, j \wedge k$ los nodos que forman el subciclo no permitido en el sentido $i \to j \to k \to i$. Como se cumple la desigualdad (3.2.1), y las demandas (q_i) son estrictamente positivas, resulta que:

$$\left.\begin{array}{l} u_j \geq u_i + q_j \Rightarrow u_j > u_i \\ u_k \geq u_j + q_k \Rightarrow u_k > u_j \\ u_i \geq u_k + q_i \Rightarrow u_i > u_k \end{array}\right\} \Rightarrow u_i > u_k > u_j > u_i \Rightarrow u_i > u_i \#$$

□

Para hallar el menor valor de M que permita cumplir esta restricción procedemos de la siguiente forma, teniendo en cuenta que el caso extremo se produce cuando $x_{ij} = 1$, u_j es el menor posible y u_i es el mayor posible:

$$\left.\begin{array}{l} u_j \geq 0, \forall j \in V \\ u_i \leq Q, \forall i \in V \end{array}\right\} \Rightarrow 0 \geq Q + q_j - M \Rightarrow M \geq Q + q_j$$

En conclusión, nos bastará con tomar un M, dependiente del indice j con valor $M_j = Q + q_j$. En el caso de que no exista el arco (i,j) pero si existe el arco (j,i), podemos añadir un nuevo factor al lado derecho de las desigualdades (3.25) que reducen la posible holgura relacionada con la M. El factor añadido es Cx_{ij}, con un valor de C concreto. Dicho valor viene de la relación, ya que existe el arco (j,i):

$$\left.\begin{array}{c} u_i = u_j + q_i \\ u_j = u_i + q_j - M_j(1 - x_{ij}) + Cx_{ij} \end{array}\right\} \Rightarrow u_j = u_j + q_i + q_j - Q - q_j + C \Rightarrow$$

$$\Rightarrow 0 = q_i - Q + C \Rightarrow C = Q - q_i$$

Es decir, para cada i tenemos un valor de $C_i = Q - q_i$. De esta forma podemos, por tanto, cambiar en el modelo (3.22) - (3.27) la restricción (3.25) por:

$$u_j \geq u_i + q_j - M_j(1 - x_{ij}) + C_i x_{ij} \quad \forall i \in V, j \in V_c / i \neq j \tag{3.28}$$

para $M_j = Q + q_j$ y $C_i = Q - q_i$.

3.2.2. Modelo de flujo

Para representar este modelo utilizaremos la variable flujo (f_{ij}) que representa la cantidad de mercancía transportada por el vehículo a lo largo de cada arco (i,j), es decir, de i a j.

$$\text{mín} \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{3.29}$$

Sujeto a

$$x(\delta^+(i)) = 1 \quad \forall i \in V_c \tag{3.30}$$
$$x(\delta^-(i)) = 1 \quad \forall i \in V_c \tag{3.31}$$
$$x(\delta^+(d)) = x(\delta^-(d)) \leq m \tag{3.32}$$
$$f(\delta^+(i)) - f(\delta^-(i)) = q_i \quad \forall i \in V_c \tag{3.33}$$
$$0 \leq f_{ij} \leq Q x_{ij} \quad \forall (i,j) \in A \tag{3.34}$$
$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{3.35}$$

Las restricciones (3.30) - (3.32) y las (3.35), son las mismas que en el caso de flujo de vehículos. Por otro lado, las restricciones (3.33) y (3.34) son las nuevas restricciones asociadas al flujo de mercancías. Las primeras representan que la cantidad de mercancías que salen de cada nodo menos la cantidad que entra es igual a la demanda de dicho nodo. Las segundas representan la cota superior de la capacidad y la incapacidad de transportar flujo por un arco si este no es usado, gracias a la multiplicación de Q por x_{ij}.

Existen cortes relacionados con el problema relajado de este tipo para ayudar a la hora de resolver, pero como no pertenecen a nuestro objetivo no los describiremos. Sin embargo, utilizarémos resultados computacionales de resolver este problema utilizando los métodos propios del resolutor, basados principalmente en el método de ramificación y corte, para compararlos con los tiempos de ejecución de lso otros modelos y métodos.

3.2.3. Modelo de Multiflujo (o flujo de 3 índices)

Es posible crear un modelo de flujo de mercancías con 3 índices basado en el modelo de flujo de vehículos de 3 índices. volveremos a considerar las variables de decisión x con tres índices, es decir, x_{ij}^k y además le incluirémeos este tercer índice también a la variable flujo, quedando entonces f_{ij}^k. Este nuevo índice indica qué vehículo es utilizado en el arco y flujo. Además debemos crear una nueva variable de paso. En conclusión:

- $x_{ij}^k \in \{0,1\}, \forall (i,j) \in A, k \in \{1,...,m\} \equiv 1$ si el vehículo k utiliza el arco (i,j), o en otro caso.

- $y_i^k \in \{0,1\}, \forall i \in V, k$
  $in\{1,...,m\} \equiv 1$ si el vehículo k visita el vértice i.

- $f_{ij}^k \in \mathbb{R}^+, \forall (i,j) \in A, k \in \{1,...,m\} \equiv$ cantidad de mercacnía transoprtada por el vehículo k por el arco (i,j).

Cabe destacar que este modelo también permite tener el cuenta el caso en el que las flotas de vehículos tienen capacidades heterogéneas, es decir, no todas las capacidades de los vehículos son iguales. Por ello, en lugar de tener una única capacidad Q, podemos decir que tenemos un conjunto de capacidades $Q_k, \forall k \in \{1,...,m\}$. El modelo quedaría por tanto:

$$\text{mín} \sum_{(i,j)\in A} c_{ij} \sum_{k=1}^{m} x_{ij}^k \tag{3.36}$$

Sujeto a

$$x^k(\delta^+(i)) = x^k(\delta^-(i)) = y_i^k \quad \forall i \in V, k \in \{1, ..., m\} \tag{3.37}$$

$$f^k(\delta^+(i)) - f^k(\delta^-(i)) = q_i y_i^k \quad \forall i \in V_c, k \in \{1, ..., m\} \tag{3.38}$$

$$0 \leq f_{ij}^k \leq Q x_{ij}^k \quad \forall (i,j) \in A, k \in \{1, ..., m\} \tag{3.39}$$

$$\sum_{k=1}^{m} y_i^k = 1 \quad \forall i \in V_c \tag{3.40}$$

$$x_{ij}^k \in \{0, 1\} \quad \forall k \in \{1, ..., m\}, (i,j) \in A \tag{3.41}$$

$$y_i^k \in \{0, 1\} \quad k \in \{1, ..., m\}, i \in V \tag{3.42}$$

Las restricciones (3.37) y (3.40) - (3.42) son las mismas e indican los mismos conceptos que para el modelo de flujo de vehículo con 3 índices. Las restricciones (3.38) y (3.39) hacen referencia al flujo de mercancías por cada arco transportado por cada vehículo. Las variables y_i^k podrán ser eliminadas pero las mantenemos porque ayuda la entendimiento y a posibles cambios de casos particulares.

Capítulo 4

Implementación en MOSEL

La implementación de los modelos explicados anteriormente ha sido una de las principales dificultades de este trabajo. Como ya hemos explicado, hemos elegido el resolutor XPRESS con su entorno gráfico MOSEL para implementar y programar los modelos. Veremos algunos de los comandos más útiles a la hora de programar, aunque no prestaremos atención a comandos básicos, para ello adjuntamos en el apéndice el programa correspondiente a la resolución del modelo de flujo de vehículos por el método de ramificación y corte.

4.1. Comandos y modelado en MOSEL

Existen varios comandos y conceptos en MOSEL imprescindibles a la hora de programar el método de ramificación y corte de manera adecuada. En el apéndice A, se muestra el código para el caso del método de ramificación y corte. Los más importantes son:

Setcallback

Las setcallback son las llamadas iterativas que utiliza el MOSEL para llamar a funciones y procedimientos a lo largo del proceso de búsqueda en un árbol del método de ramificación y acotación. Su sintaxis es del tipo:

```
setcallback(\...tipo de llamada ...", \...nombre de la función llamada...")
```

Dentro de los tipos de llamadas iterativas existen muchas que son llamadas en situaciones distintas. Para conocerlas, acudir al manual de MOSEL y al de XPRESS [2] y [3]. Veamos las usadas definitivamente:

- `setcallback(XPRS_CB_CUTMGR, cb_node:boolean)`

  Esta función es el controlador de cortes recursivo necesario para poder implementar el método de ramificación y corte. El resolutor llama a la función `\cb_node "` tras llegar a un nodo del árbol, es decir, a un problema relajado, y resolverlo. En esta función es en la que se implementa el problema de separación.

  Además, la función lleva asociada un parámetro booleano que devuelve al programa general y nos indica si se ha añadido algún nuevo corte. Si es añadido, no ramifica hacia otro nodo, sino que vuelve a resolver el problema y llamar a la función. Este proceso se repite iterativamente hasta que el parámetro booleano indique que no se ha añadido ningún corte, en cuyo caso, el programa ramifica hacia otro nodo.

- `setcallback(XPRS_CB_PREINTSOL, cb_entera(isheur:boolean, cutoff:real))`

  Este procedimiento recursivo llama a una función una vez se ha encontrado una solución entera. Esta solución entera puede ser encontrada mediante una heurística propia del resolutor o en un nodo del árbol de búsqueda. Este procedimiento recursivo es útil y necesario ya que cuando se encuentran las soluciones enteras no se llama al procedimiento de ramificación y corte, por lo cual puede que la solución no sea factible realmente y la llamada permite desecharla.

  Cabe destacar que esta función es especialmente útil para el caso de las soluciones halladas por heurísticos, ya que al estar tratando con el problema relajado, es muy raro que la heurística obtenga una solución factible.

addcuts

Este comando es de especial interés porque es el responsable de añadir los cortes al problema. Se sitúa dentro del procedimiento llamado `cb_node`. Además de añadir los cortes, este procedimiento es encargado de gestionar al cutting pool, que es la base de datos en la que se encuentra todos los cortes y que gestiona los cortes que pierden sentido al tener holgura positiva, es decir, existe otro que lo domina.

Tolerancia

La tolerancia es un aspecto importante a tener en cuenta a la hora de programar ya que el programa trabaja muchas veces con redondeos. Por ello es necesario la suma y/o resta de un $\epsilon > 0$ cuando tratamos con desigualdades.

Capítulo 5

Resultados computacionales

Nos disponemos ahora a mostrar algunos resultados computacionales de la ejecución de los diferentes modelos propuestos. Como ya hemos comentado, usamos el resolutor XPRESS, junto con el entorno gráfico llamado MOSEL, que nos permite programar los recursos que necesitamos. A la hora de elegir problemas para resolver existen dos métodos: elegir problemas en librerías públicas o crearlos. En primer lugar hemos elegido crearlos de manera aleatoria, ya que al resolverlos estamos abarcando todo tipo de problemas, sin ninguna manipulación previa, por lo que nos permite encontrar mejor posibles fallos a la hora de implementar. Sin embargo, tras comprobar el correcto funcionamiento del código del programa, hemos resuelto algunas de los problemas propuestos en la librería `VRPlib` (hemos elegido los problemas que utilizan distancias euclídeas como costes de utilizar una conección), gracias a los cuales podremos comparar nuestros estudios con los realizados por otros investigadores de manera concreta.

Debido a que nos interesa la comparación entre los distintos métodos, a la hora de crear problemas aleatorios nos basta con coger datos significativos con los que se aprecien las diferencias entre los tiempos de resolución de los distintos modelos. Para ello hemos elegido fijar como número de ciudades los casos de 10, 20, 30 y 40 ciudades. la disposición de las ciudades en el plano la realizamos gracias a la función `random`, la cual genera valores aleatorios entre 0 y 1 con distribución probabilística uniforme. Este valor aleatorio es multiplicado por 100 y redondeado, de esta manera situamos los nodos dentro del cuadrado [0,100]x[0,100]. Los costes asociados a los arcos que conectan los nodos se calcula mediante una "pseudo"distancia euclídea, obtenida tras redondear el resultado de la distancia euclídea. La demanda de cada nodo también es generada de la misma manera con valores entre 1 y 10.

Otro de los parámetros necesarios que influyen en la variación de los tiempos de ejecución es la cantidad de vehículos (m). Hemos elegido como cantidades represen-

tativas 3, 4 y 5 vehículos. Por otro lado, las capacidades de los vehículos las hemos variado en función del número de vehículos y de la demanda total, debido a la necesidad de que los problemas que creemos tengan solución óptima y los podamos comparar en todo caso. El problema surge debido a que, al generar las demandas de manera aleatoria, no conocemos a priori las demandas. Por ello hemos utilizado el hecho de que si sumamos a la media ($\overline{q_i} = n \cdot 5$) al doble de la desviación típica ($\sigma = \sqrt{n} * 3$), podemos asegurar en un 95 % que la demanda será inferior a la propuesta. Tras hallar la demanda, la dividimos por el número de vehículos, y redondeamos. Por último, aumentamos la cantidad obtenida para permitir cierta holgura.

Finalmente, para tener varios problemas y no considerar casos particulares como habíamos comentado, hemos resuelto el problema con cada dato para 5 semillas distintas, es decir, los datos que dependen de la función `random` son distintos para 5 problemas ya que varían en función de la semilla.

En las tablas que mostraremos en adelante nos referiremos a los parámetros de la siguiente forma: la primera columna n representa el número de ciudades, la segunda el número de vehículos m, la tercera la capacidad Q de los vehículos, en la cuarta la semilla (o el nombre de la instancia del `VRPLIB`) y en la quinta el coste óptimo del problema.

En lo referente al equipo informático, hemos usado un ordenador con un procesador `intel core i5-2450M CPU @ 2.50GHz x 4`, una memoria `RAM DDR3 de 8 GB`, de los cuales 2 son dedicados al sistema operativo (`Windows 7 32-bit`) y dos a la tarjeta gráfica (`NVIDIA GEFORCE GT 630M`).

Problema relajado

En primer lugar, calculamos los costes de los problemas relajados correspondientes a eliminar las restricciones de integralidad. Estos datos nos servirán como cota inferior del valor de la función objetivo correspondiente a cada modelo. Cuanto mayor sea dicha cota inferior, más se acerca la función a su óptimo. Este hecho puede ser usado en investigaciones futuras para casos de grandes nodos, mediante relajación sucesiva. Las diferencias entre el coste relajado y el óptimo obtenidos se muestran en las tablas (5.2) y (5.4) en forma de porcentaje.

Nota: El problema relajado para el método de ramificación corresponde al nodo raíz de árbol de búsqueda tras realizar la adición de las desigualdades de eliminación de subciclos correspondientes a dicho nodo, es decir, antes de la primera ramificación.

Claramente podemos ver que el método de ramificación y corte otorga las mejores cotas inferiores, llegando en muchos casos incluso a dar exactamente el valor óptimo. En segundo lugar nos encontramos que la cota proporcionada por el modelo de flujo con dos índices y el de flujo con tres índices son los siguientes mejores y con el mismo valor.

Tiempos de resolución

Veamos ahora los tiempos de resolución. Debemos tener en cuenta que, a partir de la media hora de ejecución, se ha considerado que le problema ha tardado "demasiado", y por ello se ha establecido dicho tiempo como límite. El tiempo en las tablas (5.6) y (5.8)vienen dados en segundos. Claramente vemos como el método de ramificación y corte resuelve los problemas en tiempos muchos más rápidos, seguido del modelo de flujo de mercancías con dos índices. El resto de modelos prácticamente son descartables por su gran tardanza, excepto el caso del modelo de flujo con 3 índices, el cual es útil en el caso de capacidades no homogéneas.

También podemos ver claramente como, al disminuir las capacidades de los vehículos y aumentar el número de vehículos, los tiempos de ejecución aumentan de forma significativa, por lo que nos permite considerar los problemas con bajas capacidades como "más difíciles". Cabe destacar que si consideramos la capacidad suficientemente grande, es decir, mayor o igual que la suma de las demandas, la solución será la correspondiente a resolver el problema del viajante de comercios (TSP).

En general, tras investigar la distribución de los nodos en el plano, sabemos que se resuelven de manera más rápida los problemas con el depósito situado en la región central de la envoltura convexa de los nodos, como es el caso de los problemas de la VRPlib, cuyos depósitos están situados en ese lugar lo que nos permite resolver problemas mayores. Por otro lado, los sistemas heurísticos propios de los resolutores perjudican los tiempos de resolución, ya que nunca consiguen soluciones factibles al no tener en cuenta las capacidades y los subciclos para el método de ramificación y acotación. También cabe destacar que, en la mayoría de los problemas, conociendo a posteriori la solución óptima, esta se encuentra rápido, pero el problema surge a al hora de incrementar las cotas inferiores.

Nota: No hemos calculado las medias de los tiempos de ejecución con respecto a las semillas ya que no las consideramos representativas por el hecho de que, si se pasan del tiempo límite establecido, no la deberíamos considerar y además la varianza es considerablemente grande.

n	m	Q	Seed	ópt.	Res. it.	Pot.	Flujo	Multiflujo	R. y C.
			111	308	14,6 %	25,1 %	13,5 %	13,5 %	0,0 %
			222	394	44,7 %	53,6 %	29,0 %	29,0 %	0,0 %
	3	25	333	374	39,8 %	44,1 %	13,4 %	13,4 %	0,0 %
			444	439	37,8 %	41,0 %	13,2 %	13,2 %	5,5 %
			555	424	32,9 %	46,7 %	22,5 %	22,5 %	0,0 %
			111	362	27,3 %	36,3 %	21,8 %	21,8 %	0,0 %
			222	439	50,3 %	57,2 %	26,2 %	26,2 %	8,9 %
10	4	20	333	440	48,9 %	51,9 %	17,0 %	17,0 %	0,0 %
			444	538	49,3 %	51,5 %	20,1 %	20,1 %	9,5 %
			555	475	40,1 %	51,3 %	21,7 %	21,7 %	1,8 %
			111	391	32,7 %	40,9 %	19,8 %	19,8 %	0,0 %
			222	528	58,7 %	62,8 %	23,6 %	23,6 %	0,0 %
	5	15	333	521	56,8 %	58,5 %	15,9 %	15,9 %	1,2 %
			444	624	56,3 %	57,5 %	17,8 %	17,8 %	0,2 %
			555	540	47,3 %	54,6 %	16,8 %	16,8 %	0,0 %
			111	517	23,8 %	30,1 %	18,0 %	18,0 %	4,7 %
			222	516	33,9 %	54,0 %	28,8 %	28,8 %	5,0 %
	3	50	333	484	34,2 %	50,3 %	23,0 %	23,0 %	2,6 %
			444	569	42,4 %	47,6 %	18,8 %	18,8 %	2,6 %
			555	528	30,3 %	33,6 %	11,2 %	11,2 %	0,0 %
			111	542	27,3 %	32,6 %	15,3 %	15,3 %	2,8 %
			222	583	41,5 %	58,7 %	28,9 %	28,9 %	2,3 %
20	4	40	333	527	39,6 %	53,1 %	20,4 %	20,4 %	4,9 %
			444	612	46,4 %	51,0 %	15,5 %	15,5 %	4,2 %
			555	626	41,2 %	43,8 %	16,5 %	16,5 %	1,0 %
			111	627	37,2 %	40,7 %	16,8 %	16,8 %	4,7 %
			222	691	50,7 %	64,4 %	27,6 %	27,6 %	5,8 %
	5	30	333	600	46,9 %	57,0 %	16,8 %	16,8 %	1,3 %
			444	756	56,6 %	60,0 %	18,7 %	18,7 %	5,6 %
			555	768	52,1 %	53,9 %	19,6 %	19,6 %	6,7 %
			111	575	22,4 %	32,7 %	16,8 %	16,8 %	3,2 %
			222	660	41,5 %	52,4 %	20,4 %	20,4 %	4,0 %
	3	75	333	506	23,0 %	39,3 %	21,5 %	21,5 %	1,2 %
			444	659	34,4 %	39,1 %	22,8 %	22,8 %	4,6 %
			555	540	23,2 %	28,7 %	10,5 %	10,5 %	3,6 %
			111	613	27,2 %	36,3 %	16,1 %	16,1 %	3,2 %
			222	725	46,8 %	56,2 %	18,1 %	18,1 %	5,0 %
30	4	60	333	571	31,8 %	45,8 %	24,2 %	24,2 %	5,7 %
			444	694	37,8 %	41,9 %	19,4 %	19,4 %	3,9 %
			555	634	34,6 %	39,2 %	16,8 %	16,8 %	6,6 %
			111	698	36,1 %	43,3 %	17,5 %	17,5 %	5,6 %
			222	832	53,6 %	61,1 %	14,4 %	14,4 %	0,4 %
	5	45	333	646	39,7 %	51,4 %	22,5 %	22,5 %	5,0 %
			444	794	45,6 %	49,0 %	17,7 %	17,7 %	5,3 %
			555	685	39,5 %	43,5 %	12,0 %	12,0 %	3,4 %
			111	599	16,7 %	26,2 %	16,9 %	16,9 %	2,4 %
			222	721	39,1 %	45,8 %	20,0 %	20,0 %	5,6 %
	3	100	333	567	19,4 %	30,9 %	18,5 %	18,5 %	3,8 %
			444	633	23,2 %	33,9 %	17,9 %	17,9 %	5,1 %
			555	586	19,9 %	26,3 %	10,1 %	10,1 %	0,9 %
			111	631	20,9 %	29,6 %	17,1 %	17,1 %	2,2 %
			222	820	46,5 %	52,2 %	21,2 %	21,2 %	11,0 %
40	4	80	333	603	24,2 %	34,8 %	18,8 %	18,8 %	2,8 %
			444	704	31,0 %	40,2 %	20,0 %	20,0 %	7,4 %
			555	649	27,7 %	33,3 %	13,8 %	13,8 %	3,6 %
			111	696	28,3 %	35,8 %	17,8 %	17,8 %	3,4 %
			222	945	53,5 %	58,3 %	18,6 %	18,6 %	5,0 %
	4	60	333	675	32,3 %	41,5 %	20,0 %	20,0 %	3,0 %
			444	808	39,9 %	47,5 %	21,0 %	21,0 %	4,7 %
			555	717	34,5 %	39,4 %	13,7 %	13,7 %	3,5 %

Tabla 5.2: Coste relajado. Problemas aleatorios

n	m	Q	Nombre	ópt.	Res. it.	Pot.	Flujo	Multiflujo	R. y C.
16	3	90	E016-03m.dat	278,7	25,4 %	27,3 %	13,2 %	13,2 %	7,6 %
16	5	55	E016-05m.dat	335,0	37,9 %	38,7 %	14,3 %	14,3 %	9,2 %
21	4	85	E021-04m.dat	358,4	28,7 %	30,8 %	13,5 %	13,5 %	5,8 %
21	6	58	E021-06m.dat	430,9	40,7 %	41,9 %	14,6 %	14,6 %	9,4 %
22	4	6000	E022-04g.dat	375,3	30,2 %	39,4 %	18,1 %	18,1 %	2,8 %
22	6	4000	E022-06m.dat	495,8	47,2 %	52,2 %	21,2 %	21,2 %	6,4 %
23	3	4500	E023-03g.dat	568,6	21,1 %	31,4 %	17,1 %	17,1 %	1,7 %
23	5	4500	E023-05s.dat	568,6	21,1 %	31,4 %	17,1 %	17,1 %	1,7 %
26	8	48	E026-08m.dat	607,7	52,8 %	55,6 %	16,7 %	16,7 %	5,2 %
30	3	4500	E030-03g.dat	535,8	43,3 %	47,5 %	22,0 %	22,0 %	8,2 %
30	4	4500	E030-04s.dat	505,0	39,8 %	44,3 %	17,2 %	17,2 %	2,6 %

Tabla 5.4: Coste relajado. Problemas VRPlib

n	m	Q	Nombre	ópt.	Res. it.	Pot.	Flujo	Multiflujo	R. y C.
16	3	90	E016-03m.dat	278,7	1800,0	586,8	5,6	162,2	9,0
16	5	55	E016-05m.dat	335,0	1800,0	85,5	6,2	1800,0	21,8
21	4	85	E021-04m.dat	358,4	1800,0	1800,0	139,0	1800,0	30,2
21	6	58	E021-06m.dat	430,9	1800,0	1800,0	16,0	1800,0	82,9
22	4	6000	E022-04g.dat	375,3	1800,0	1800,0	16,5	1800,0	0,7
22	6	4000	E022-06m.dat	495,8	1800,0	1800,0	1800,0	1800,0	211,7
23	3	4500	E023-03g.dat	568,6	1800,0	1800,0	14,9	1800,0	0,7
23	5	4500	E023-05s.dat	568,6	1800,0	1800,0	23,0	1800,0	0,8
26	8	48	E026-08m.dat	607,7	1800,0	1800,0	182,5	1800,0	547,5
30	3	4500	E030-03g.dat	535,8	1800,0	1800,0	1800,0	1800,0	46,2
30	4	4500	E030-04s.dat	505,0	1800,0	1800,0	1800,0	1800,0	0,7

Tabla 5.6: Tiempos de ejecución. Problemas VRPlib

n	m	Q	Seed	ópt.	Res. it.	Pot.	Flujo	Multiflujo	R. y C.
			111	308	0,1	0,2	0,1	0,4	0,0
			222	394	14,3	2,7	1,5	2,3	0,0
	3	25	333	374	3,1	0,7	0,5	1,1	0,0
			444	439	12,3	6,5	0,7	1,2	0,3
			555	424	3,4	0,6	0,8	1,3	0,0
			111	362	0,3	0,5	0,4	0,7	0,0
			222	439	46,6	1,9	1,0	3,5	0,4
10	4	20	333	440	28,3	0,8	0,8	2,9	0,0
			444	538	119,0	4,8	1,9	3,6	0,3
			555	475	12,7	0,5	0,5	1,8	0,2
			111	391	0,2	0,3	0,4	1,0	0,0
			222	528	23,0	0,4	0,5	2,7	0,0
	5	15	333	521	10,9	0,2	0,5	2,2	0,4
			444	624	56,5	1,3	1,2	2,4	0,1
			555	540	8,7	0,4	0,4	0,9	0,1
			111	517	1800,0	1800,0	278,1	1800,0	0,9
			222	516	1800,0	1800,0	341,7	1800,0	6,4
	3	50	333	484	1800,0	1800,0	1409,3	1800,0	2,7
			444	569	1800,0	1800,0	190,3	1800,0	2,9
			555	528	1800,0	1800,0	3,5	1800,0	0,5
			111	542	1800,0	1800,0	14,6	1800,0	1,9
			222	583	1800,0	1800,0	1800,0	1800,0	2,4
20	4	40	333	527	1800,0	1800,0	1800,0	1800,0	3,6
			444	612	1800,0	1800,0	30,6	1800,0	5,3
			555	626	1800,0	1800,0	1800,0	1800,0	0,4
			111	627	1800,0	1800,0	1210,1	1800,0	21,8
			222	691	1800,0	1800,0	1800,0	1800,0	21,9
	5	30	333	600	1800,0	1800,0	1247,6	1800,0	0,5
			444	756	1800,0	1800,0	1800,0	1800,0	160,5
			555	768	1800,0	1800,0	1800,0	1800,0	38,1
			111	575	1800,0	1800,0	1800,0	1800,0	5,2
			222	660	1800,0	1800,0	1800,0	1800,0	5,5
	3	75	333	506	1800,0	1800,0	71,6	1800,0	1,0
			444	659	1800,0	1800,0	1800,0	1800,0	122,8
			555	540	1800,0	1800,0	83,1	1800,0	8,7
			111	613	1800,0	1800,0	1800,0	1800,0	33,7
			222	725	1800,0	1800,0	1800,0	1800,0	133,4
30	4	60	333	571	1800,0	1800,0	1800,0	1800,0	49,0
			444	694	1800,0	1800,0	1800,0	1800,0	62,9
			555	634	1800,0	1800,0	1800,0	1800,0	27,1
			111	698	1800,0	1800,0	1800,0	1800,0	51,9
			222	832	1800,0	1800,0	1800,0	1800,0	25,5
	5	45	333	646	1800,0	1800,0	1800,0	1800,0	96,7
			444	794	1800,0	1800,0	1800,0	1800,0	489,8
			555	685	1800,0	1800,0	1800,0	1800,0	41,4
			111	599	1800,0	1800,0	1800,0	1800,0	23,6
			222	721	1800,0	1800,0	1800,0	1800,0	13,8
	3	100	333	567	1800,0	1800,0	1800,0	1800,0	93,6
			444	633	1800,0	1800,0	1800,0	1800,0	3,5
			555	586	1800,0	1800,0	1800,0	1800,0	0,8
			111	631	1800,0	1800,0	1800,0	1800,0	9,8
			222	820	1800,0	1800,0	1800,0	1800,0	1800,0
40	4	80	333	603	1800,0	1800,0	1800,0	1800,0	39,2
			444	704	1800,0	1800,0	1800,0	1800,0	131,7
			555	649	1800,0	1800,0	1800,0	1800,0	49,5
			111	696	1800,0	1800,0	1800,0	1800,0	104,2
			222	945	1800,0	1800,0	1800,0	1800,0	1800,0
	5	60	333	675	1800,0	1800,0	1800,0	1800,0	208,5
			444	808	1800,0	1800,0	1800,0	1800,0	1800,0
			555	717	1800,0	1800,0	1800,0	1800,0	76,7

Tabla 5.8: Tiempos de ejecución. Problemas aleatorios

Capítulo 6

Conclusiones finales

A lo largo de este trabajo hemos introducido un problema de gran importancia en lo referente a la investigación operativa. Esta importancia no solo reside en el puro conocimiento computacional y matemático del mismo, sino que es ampliamente aplicable en muchas situaciones de la vida como puede ser el reparto de mercancías o la optimización del carburante de un vehículo. Además existen otros problemas que, al modelizarlos, obtenemos situaciones similares a este problema, con lo cual la resolución es idéntica aunque su interpretación final sea diferente.

Aprovechando esta importancia, se han introducido conceptos fundamentales de la programación matemática necesarios para la modelización y resolución de problemas de programación lineal entera, con los que hemos cumplido parte de los objetivos marcados al inicio del proyecto.

El objetivo más importante a lo largo del proyecto, quizás, ha sido la familiarización con los conceptos asociados a los modelos y a las proyecciones de los mismos, y la programación de ellos en un lenguaje que permita la resolución de diversas maneras. El aprendizaje no ha venido únicamente a la hora de aprender el lenguaje MOSEL, sino también todos los conocimientos asociados a los ficheros extra necesarios (entrada y salida de datos) y el formato de los mismos que facilitan la ejecución automática de los problemas. Todo esto ha contribuido también ha cumplir los objetivos de aprendizajes marcados al inicio.

Desde el punto de vista de los resultados computacionales, hemos conseguido demostrar de manera concreta y ejemplos específicos que el método de ramificación y corte como método de resolución de problemas de programación lineal entera es mejor que el método de ramificación y acotación. También hemos visto que la modelización del CVRP considerando flujos de vehículos se resuelve de manera más rápida que con la modelización a través flujo de mercancías para el caso de flotas

de vehículos con capacidades homogéneas. Cabe destacar también el hecho de la mejora de los tiempos de resolución si desactivamos los procedimientos heurísticos propios de los resolutores informáticos. Es de gran utilidad el hecho de conocer la diferencia entre las cotas inferiores y superiores de la solución, ya que podríamos dar una solución factible, con una diferencia concreto posible con la solución óptima real sin conocerla (aunque, como se ha comentado, dicha solución es rápidamente encontrada pero la dificultad reside en aumentar al cota inferior).

Por último, comentar que existen muchas líneas de investigación más relacionadas con los problemas de rutas de vehículos y que podrían ser objeto de investigaciones futuras en trabajos fin de grado de años venideros, como es el caso de considerar flotas heterogéneas de vehículos y comparar sus modelos y resoluciones, modelos para el transporte de más de un tipo de mercancías (y gracias al cual podríamos considerar conjuntamente el problema de la carga y del combustible) o, teniendo en cuenta la tabla introducida en el capítulo 1, hacer frente a la modelización de otros casos de VRP, como pueden ser el transporte particionado o las ventanas de tiempos.

Apéndice A

Scripts

A.1. Código recursivo para la resolución multiple del modelo de flujo de vehículos

```
procedure break_capacity

        declarations
               STARTS, TOUR,SMALLEST,ALLCITIES: set of integer
               NEXTC: array(CITIES) of integer
               size_smallest, size_tours, cap, minus: integer
               end-declarations

        forall(i in 2..NCITIES) do
               NEXTC(i):= integer(round(getsol(sum(j in V) j*assig(i,j) )))
        end-do
        STARTS:= {}
        forall(i in 2..NCITIES) do
               if round(getsol(assig(1,i))) = 1 then
                       STARTS += {i}
               end-if
        end-do

        ! Encontarar el subtour que contien el depósito

        size_tours :=0
        size_smallest := 1000
        minus := 0
        forall(i in STARTS) do
               TOUR:={}
               first:=i
               cap := 0
               repeat
                       TOUR+={first}
                       cap +=DEMANDS(first)
                       if(cap > CAPACITY) then
                               ! hemos encontrado un subconjunto recorrido consecutivamente que
                               ! supera la capacidad del vehículo
                               if(size_smallest > getsize(TOUR)) then
                                       ! Si es el de un tamaño más pequeño lo actualizamos
                                       SMALLEST := TOUR;
                                       size_smallest := getsize(TOUR)
```

```
                                        minus := 2
                                end-if
                        end-if
                        first:=NEXTC(first)
                until first=1
                size_tours += getsize(TOUR)
                ALLCITIES += TOUR
        end-do

        ! Si minus > 0 hemos encontrado un subconjunto que supera la capacidad
        ! y si size_tours < NCITIES hay clientes que no están conectados con el depósito (hay subciclos)
        ! Vamos a buscar el menor de los subconjunto que viola la restricción
        if (minus > 0 or size_tours < (NCITIES-1)) then
                forall(i in 2..NCITIES) do
                        if(i not in ALLCITIES) then
                                TOUR:={}
                                first:=i
                                cap := 0
                                repeat
                                        TOUR+={first}
                                        cap += DEMANDS(first)
                                        if(cap > CAPACITY) then
                                                if(size_smallest > getsize(TOUR)) then
                                                        SMALLEST := TOUR;
                                                        size_smallest := getsize(TOUR)
                                                        minus:= 2
                                                end-if
                                        end-if
                                        first:=NEXTC(first)
                                until first=i
                                ALLCITIES+=TOUR
                                if getsize(TOUR)< size_smallest then
                                        SMALLEST:=TOUR
                                        size_smallest:=getsize(SMALLEST)
                                        minus:= 1
                                end-if
                                if size_smallest=2 then
                                        break
                                end-if
                        end-if
                end-do
        end-if
        if(minus > 0) then
                sum(i,j in SMALLEST | i<>j ) assig(i,j) <= size_smallest - minus
                write(mips,") Cost: ", getobjval,"   ; minus=", minus,"  size=", size_smallest," ={")
                forall(i in SMALLEST) write(i,",")
                writeln("}")
                mips := mips+1
                draw
                !IVEpause(""+mips)

                ! Volver a resolver el problema
                minimize(TotalDist)

                ! Recursividad
                break_capacity
                end-if
end-procedure
```

A.2. Código para la detección de los subconjuntos asociados a un corte mínimo

```
forall(i in CITIES_maxflow) inside(i):=false
TOUR(1):= SOURCE
inside(SOURCE):= TRUE
last :=1
while (last>0) do
        u := TOUR(last)
        last := last-1
        forall ( v in CITIES_maxflow | v<>u ) do
                if ( Capacity(u,v) - Flow(u,v) > epsilon or Flow(v,u) > epsilon )
                    and (not inside(v)) then
                        last := last+1
                        TOUR(last):= v
                        inside(v):=true
                end-if
        end-do
end-do
```

A.3. Código completo de la función correspondiente al método de ramificación y corte

```
(!*****************************************************************!)
public function cb_node:boolean
   declarations
               Capacity,Flow:array (CITIES_maxflow,CITIES_maxflow) of real
               first,last, SINK,SOURCE :integer
               TOUR  : array(CITIES_maxflow) of integer
               inside: array(CITIES_maxflow) of boolean
               maxflow: mpproblem
               amount,Z:real
               flow: array(CITIES_maxflow,CITIES_maxflow) of mpvar
   end-declarations

   addcut_in_cb_entera:= true


  !Creamos un grafo con cargas, las cuales se corresponden con

  !la solución parcial


   forall(n,m in CLIENTS|n<>m) Capacity(n,m) := getsol(assig(n,m))

   forall(j in CLIENTS) Capacity(depot,j) := (getsol(assig(depot,j))-
       Demand(j)/CAPACITY)
   forall(j in CLIENTS) Capacity(j,NCITIES+1) := - Capacity(depot,j)
```

```
   forall(i,j in CITIES_maxflow|i<>j and Capacity(i,j)<epsilon )
        Capacity(i,j) := 0

   Z:=sum(j in CLIENTS)Capacity(j,NCITIES+1)

   returned:=false

   !Resolvemos el problema de flujo máximo
   SOURCE:=depot
    SINK:=NCITIES +1
    with maxflow do
        reset( maxflow )
        forall(n,m in CITIES_maxflow |n<>m and Capacity(n,m)>epsilon )
                   create(flow(n,m))
        forall(n in CLIENTS)
sum(m in CITIES|m<>n and exists(flow(m,n))) flow(m,n)
                 = sum(m in CITIES_maxflow|m<>n and exists(flow(n,m))) flow(n,m)
        forall(n,m in CITIES_maxflow | n<>m and exists(flow(n,m)) )
               flow(n,m)<=Capacity(n,m)

        maximize(sum(n in CITIES|n<>SOURCE and exists(flow(SOURCE,n)))
        flow(SOURCE,n))

        amount := getobjval
        forall (n,m in CITIES_maxflow | n<>m) Flow (n,m):= getsol(flow(n,m))
    end-do

    !Identificamos si el flujo es menor que Z, para posteriormente encontrar una
            restricción de subciclos violada
    if amount < Z-epsilon then
       forall(i in CITIES_maxflow) inside(i):=false
       TOUR(1):= SOURCE
       inside(SOURCE):= TRUE
       last :=1
       while (last>0) do
         u := TOUR(last)
         last := last-1
         forall ( v in CITIES_maxflow | v<>u ) do
           if ( Capacity(u,v) - Flow(u,v) > epsilon or Flow(v,u) > epsilon )
                    and (not inside(v)) then
             last := last+1
             TOUR(last):= v
             inside(v):=true
           end-if
          end-do
        end-do

        forall(i in CITIES|inside(i)=true) write(i, " ")
```

```
        writeln("")

        !Añadimos el corte oportuno

        DEMAND_S:= ceil((sum (k in CITIES| inside(k)=false)Demand(k)/CAPACITY))

        cut:=(sum(i,k in CITIES|inside(i)=false and inside(k)=true)assig(i,k))
              -DEMAND_S
        addcut(1, CT_GEQ, cut)

        writeln("Corte añadido: (profundidad= ",getparam("XPRS_NODEDEPTH"),",
            nodo= ",getparam("XPRS_NODES"), ", valor objetivo= ",
            getparam("XPRS_LPOBJVAL"), ")")
        draw
        !IVEpause("")

        returned:=true
   end-if

   if returned = false then
     writeln("Ramificando")
   end-if
end-function
```

Bibliografía

[1] P. Augerat. *Approche Polyèdrale du Problème de Tournées de Véhicles*. PhD thesis, Institute National Politechnique de Grenoble, France, 1995.

[2] $FICO^{TM}$ Xpress Optimization Suite. *Xpress-Optimizer Reference manual*, 2009.

[3] $FICO^{TM}$ Xpress Optimization Suite. *Xpress-Mosel (User guide)*, 2013.

[4] Juan José Salazar González. *Programación Matemática*. Díaz de Santos, 2001.

[5] L. Gouveia. A result for projection for the vehicle routing problem. *European Journey Operational Research*, 85:610–624, 1995.

[6] G. Laporte and Y. Nobert. Exact algorithms for the vehicle routing problem. *Ann. Discr. Math*, 31:147–184, 1987.

[7] A.N. Letchford and J.J. Salazar-González. Projection results for vehicle routing. *Math. Program., Ser. B*, 105:251 – 274, 2006.

[8] D. Naddef and G. Rinaldi. *Branch and cut algorithms for the capacitated VRP*, chapter 3. SIAM, 2001.

[9] M.R. Rao T. McCormick and G.Rinaldi. When is min cut with negative edgeseasy to solve? easy and dificult orthants. In preparation, 2000.

Branch and cut

Algorithm Complexity

Modeling

Mathematical Programming

In this Bachelor Thesis has been addressed one of the most interdisciplinary areas of mathematics, mathematical programming and specifically, a problem called Capacitated Vehicle Routing Problem (CVRP). The aim of the study was to determine the methods of solving these problems that are used and the research line that exist nowadays.

This report was proposed in order to confirm that the branch and cut method is effective in solving problems considerably large. For this, the algorithms needed to implement the model in a programming language called MOSEL and the correspondent code. This work also serves to familiarize with the facts involving current solvers market.

VEHICLE ROUTING PROBLEMS

EXACT SOLVING METHODS

Juan Francisco Pérez Caballero july, 21st 2014

Faculty of Mathematics

University of La Laguna