Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Tesis de Licenciatura

Un Algoritmo de *Branch&Cut* para el problema de *Ruteo de Vehículos con Capacidades* basado en un nuevo modelo de flujo en redes

Alumno: Daniel Negrotto
Directores: Dr. Julián Aráoz Durand
Dra. Irene Loiseau

Diciembre de 2006

Quiero dedicar esta tesis a mis dos amores, Inés, y nuestra hija Josefina.

Agradecimientos

Quiero agradecerles en primer lugar a Irene Loiseau y a Julián Aráoz Durand por todo el apoyo, el estímulo, el aguante y el tiempo tan valioso que me dedicaron. Muchas gracias de todo corazón!.

Les quiero agradecer también por el apoyo y por la ayuda incondicional a Isabel Méndez Díaz y a Paula Zabala que me introdujeron en el uso de ABACUS y de CPLEX, como también en la discusión de temas relacionados con el problema, el intercambio de papers y rutinas de separación. Además, como jurados de esta tesis, proporcionaron comentarios sobre el trabajo que permitieron mejorar y pulir la redacción final de la misma.

Quiero agradecer especialmente también a Javier Marenco que me transmitió generosamente muchas ideas y conocimientos en un largo e inolvidable viaje a Pucón, Chile.

No quisiera dejar de mencionar mi agradecimiento a Elena Fernández, a quien conozco a través de mails gracias a Julián Aráoz, por el aporte de comentarios e ideas para mejorar los modelos de flujo utilizados.

Agradezco también a mis padres, por el constante apoyo y la confianza que me transmitieron durante toda la carrera. Vieron que algún día llegaría este momento!.

Por último, quiero agradecer especialmente a Néstor Sameghini, a quien debo mi vocación por la computación y el haber elegido esta hermosa carrera.
Muchas gracias a todos!

Resumen

El problema de *Ruteo de Vehículos con Capacidades (CVRP)* es una de las variantes de la vasta familia de problemas de Ruteo de Vehículos. En CVRP se requiere encontrar un conjunto de caminos o rutas que visite una lista de clientes, de forma tal que cada cliente se encuentre en una de las rutas y que los pedidos de los mismos sean satisfechos por los vehículos que los visitan, sin sobrepasar la capacidad máxima de los mismos. Además, cada cliente debe ser visitado por un sólo vehículo. El objetivo es minimizar la distancia total recorrida por la flota de vehículos.

CVRP es un problema NP-Hard y los métodos de *branch&cut* se encuentran dentro de los enfoques más prometedores para resolverlo de manera exacta. Se han propuesto numerosos modelos para representarlo. Pueden destacarse los modelos de flujo de 2 índices que utilizan $O(n^2)$ variables y los de 3 índices que utilizan $O(n^2K)$ variables, siendo n la cantidad de clientes y K la cantidad de vehículos. Estos últimos modelos son más flexibles y se adaptan mejor a diversas extensiones del problema CVRP.

En el presente trabajo, se introduce un nuevo modelo de flujo de 3 índices que utiliza una cantidad de $O(n^2)$ variables, pero que mantiene la misma flexibilidad que los modelos de flujo de 3 índices. Se demuestra su correctitud con respecto a los modelos de flujo de 2 índices y se describe la implementación completa de un algoritmo de *branch&cut* que utiliza de manera exitosa la nueva formulación. Por último, se presentan los resultados obtenidos y se los compara con los obtenidos en trabajos de reciente publicación. La evaluación de los resultados es positiva y promete avances alentadores en trabajos futuros.

Abstract

The problem of the Capacitated Vehicle Routing is one of the varieties among a vast family of problems in Vehicle Routing. In CVRP we need to find a set of routes that visit a list of clients, trying that each client is located in one of the routes and that their requests are satisfied by the visiting vehicles, without exceeding their maximum capacity. Moreover, each client must be visited by only one vehicle. The aim is to minimize the total routing distance of the fleet of vehicles.

CVRP is an NP-Hard problem and the *Branch&cut* methods are within the most promising approaches to solve it in an exact way. Several models have been proposed to represent it such as the two-index flow models that use $O(n^2)$ variables and those of three-indexes that use $O(n^2K)$ variables, being n the amount of clients and K the amount of vehicles. These models are more flexible and better adapt to the different variations of the CVRP problems.

In the present work, we introduce a new model of three-index flow that uses an amount of $O(n^2)$ variables but keeping the same flexibility of traditional three-index models. We prove its correctness in relation to the two-index flow models and we describe the complete implementation of a *branch&cut* algorithm that successfully uses the new formulation. Finally, we present the results obtained and compare them with those obtained in recent publications. The evaluation of the results is positive and represents a promising advance for future works.

Índice general

1. Introducción 1

1.1. Programación Lineal 3

1.2. Programación Lineal Entera 3

1.3. Algoritmos de Plano de Corte 4

1.4. Algoritmos de *branch&cut* 6

2. El Problema de Ruteo de Vehículos con Capacidades 9

2.1. Problemas de Ruteo de Vehículos (VRP) 9

2.2. El problema de Ruteo de Vehículos con Capacidades (CVRP) 10

2.2.1. Modelos de flujo de vehículos 11

2.2.1.1. Modelos de flujo de vehículos de 2 índices 13

2.2.1.2. Modelos de flujo de vehículos de 3 índices 15

2.2.2. Modelos de flujo de *commodities* (commodity flow models) 17

2.2.3. Modelos de partición de conjuntos (Set-Partitioning Models) 19

3. Nuevo modelo para el problema de Ruteo de Vehículos con Capacidades 21

3.1. Nuevo modelo de flujo de vehículos de 3 índices 21

3.1.1. Construcción del nuevo modelo 21

3.1.2. Formulación del modelo 24

3.1.3. Correctitud del modelo presentado 28

3.1.4. Análisis del modelo 29

3.1.5. Extensiones y posibles mejoras 30

3.1.6. Adaptación del nuevo modelo a Ruteo de Vehículos con Ventanas de Tiempo (VRPTW) 31

4. El algoritmo de *branch&cut* **33**

4.1. Desigualdades válidas utilizadas 34

4.1.1. Desigualdades de Capacidad 34

4.1.2. Desigualdades de Lógica de Ruta 1 37

4.1.3. Desigualdades de Rutas Dirigidas 38

4.1.4. Desigualdades de Camiones Ordenados 39

4.1.5. Desigualdades Framed Capacity (FCI) 39

4.1.6. Desigualdades Strengthened Comb 40

4.1.7. Desigualdades MultiStar y Partial MultiStar 41

4.1.8. Desigualdades Generalized Large MultiStar (GLM) 42

4.1.9. Desigualdades Hypotour 43

4.1.10. Estrategia de separación 44

4.2. Estrategia de Branching 46

4.2.1. Branching sobre conjuntos 47

4.2.2. Branching sobre variables 48

4.2.3. Estrategia de Enumeración 48

4.2.4. Detección de problemas no factibles 48

4.3. Heurísticas Primales 49

4.3.1. Búsqueda de solución inicial 49

4.3.2. Heurística de mejoramiento 51

4.4. Fijado y Seteado de variables por implicaciones lógicas 52

4.5. Otras particularidades de la implementación 53

4.5.1. Relajación inicial 53

4.5.2. Pools de Desigualdades 53

4.5.3. Eliminación de Desigualdades Inactivas 54

4.5.4. Identificación de soluciones válidas 54

5. Resultados Computacionales **56**

5.1. Instancias de prueba 57

5.2. Evaluación de las distintas desigualdades válidas utilizadas 57

5.3. Estrategias de Separación 63

5.4. Fijación y seteo de variables por implicaciones lógicas 67

5.5. Branching de variables vs Branching de conjuntos 69

5.6. Comparación con trabajos previos 70

6. Conclusiones **76**

6.1. Trabajos futuros 77

A. Detalles de Implementación **80**

A.1. El framework ABACUS 80

A.2. La estructura de clases de ABACUS 81

A.3. Descripción de la implementación 83

A.3.1. Lectura de las instancias de prueba 83

A.3.2. Variables del modelo 83

A.3.3. Restricciones del modelo 84

A.3.4. Clase Master 86

A.3.5. Clase Subproblema 86

A.3.6. Parámetros específicos 86

A.3.7. Clase Branching 87

Bibliografía **92**

Índice de figuras

1.1. (a) Relajación Lineal. (b) Cápsula convexa. 5

2.1. Resolución óptima de la instancia F-n135-k7 11
2.2. Ejemplo de Red de flujo y asignación de flujo máximo 12
2.3. Ejemplo de una ruta de flujo de commodities (C=15) 18

3.1. Modelo de 2 índices asimétrico 22
3.2. Partición Nodo Cliente 22
3.3. Partición Nodo Vehículo 23
3.4. Nuevo modelo de 3 índices 24

A.1. Jerarquía de clases de variables 84

Capítulo 1

Introducción

La Investigación Operativa, también conocida como Investigación de Operaciones, tiene sus orígenes durante los comienzos de la Segunda Guerra Mundial. Esta área de la matemática aplicada se creó inicialmente con fines militares obteniendo grandes logros en la localización de radares como en la detección de submarinos.
Rápidamente se vislumbró la posibilidad de aplicarla a una innumerable variedad de problemas que exedían a los de aplicación estrictamente militar. Es en ese momento cuando la Investigación Operativa recibe la atención de la comunidad científica internacional y produce su principal avance. La mayor aplicación es en el análisis para la toma de decisiones teniendo en cuenta la mejor utilización de un conjunto acotado de recursos. Sus aportes incluyen grandes mejoras a la productividad de un sinfin de empresas comerciales, avances en la salud, el planeamiento urbano, el transporte, etc.

Un problema de *optimización* tiene como objetivo la búsqueda de un máximo o de un mínimo de una determinada función matemática correctamente definida en un conjunto dado. Dicho problema es de *optimización combinatoria* si el dominio de aplicación es discreto (JRT94). Mas concretamente, se puede definir la *optimización combinatoria* de la siguiente forma:

Dado un conjunto finito $\mathcal{I}$ (el conjunto de soluciones factibles) y una función $f : \mathcal{I} \to \Re$, encontrar un elemento $I^* \in \mathcal{I}$ que verifique que

$$f(I^*) = \text{mín}\{f(I) \mid I \in \mathcal{I}\}.$$

Para resolver un problema de este tipo es necesario tener una caracterización razonable del conjunto de soluciones factibles $\mathcal{I}$ y una manera algorítmica de evaluar la función objetivo para cada $I \in \mathcal{I}$. A menudo, esta caracterización es sencilla. Por ejemplo, en muchos casos corresponde a la estructura de los subgrafos de un grafo dado que satisface una determinada propiedad estructural. Con respecto a la evaluación de la función objetivo, existe un concepto simple que se aplica a muchos problemas difíciles, y que consiste básicamente

en asignarle pesos a los elementos finitos del dominio (Thi95). Sin embargo, muchos de los problemas que surgen mediante esta caracterización pertenecen al grupo de problemas *NP-Hard*, con lo cual su resolución difícilmente pueda ser eficiente a menos que $P = NP$.

Existen diversas maneras de abordar estos problemas difíciles. Uno de los métodos más utilizados suelen ser los *algoritmos heurísticos* que se basan en explotar alguna característica particular del problema y construir, de esa manera, una solución que se espera sea cercana a la óptima (pero puede no serlo).

Existen enfoques exactos que permiten obtener el valor óptimo del problema pero cuando los problemas son *NP-Hard* la complejidad computacional puede ser exponencial. Sin embargo, el éxito de dichos métodos depende del problema en cuestión y los algoritmos que mejores resultados han dado en esta área son los llamados algoritmos de *branch&cut*. Estos últimos utilizan como piedra fundamental la programación lineal y la programación lineal entera.

En este trabajo se aborda el problema de *Ruteo de Vehículos con Capacidades* y se proponen nuevos modelos de programación lineal entera basados en *modelos de flujo en redes*. Se describirá también la implementación de un algoritmo de *branch&cut* específico para los modelos presentados.

En lo que resta de la Introducción se decribirá con mayor detalle los conceptos generales de los algoritmos de plano de corte y de *branch&cut*. En el capítulo 2 se intruducen los problemas de *Ruteo de Vehículos* y se detalla con especial énfasis la variante de *Ruteo de Vehículos con Capacidades*. Se presenta, también, en este capítulo los modelos clásicos utilizados hasta el momento. En el capítulo 3 se introducen los nuevos modelos aportados. Por su parte, en el capítulo 4, se describe el desarrollo del algoritmo de *branch&cut* implementado detallando cada uno de las partes que lo conforman. Luego se exponen las familias de desigualdades utilizadas y los algoritmos de separación respectivos.

Por último, en el capítulo 5 se informan los resultados obtenidos de la aplicación del algoritmo sobre las instancias de prueba TSPLIB95 (Rei95) junto a las compiladas en www.branchandcut.org (Ral95) y se las compara con los resultados reportados en trabajos anteriores.

El algoritmo implementado utiliza el entorno *ABACUS* (JT97a; JT97b; Thi95; Thi96; J̈99) junto con el paquete de optimización *CPLEX* para problemas de programación lineal (Inc94).

1.1. Programación Lineal

La Programación Lineal es una de las herramientas básicas para resolver problemas de optimización matemática. Un *problema de programación lineal* (del inglés linear programming - LP) puede formularse de la siguiente manera:

Dada una matriz $A \in \Re^{(m,n)}$, y los vectores $b \in \Re^m$, $c \in \Re^n$, encontrar un vector $x^* \in \Re^n$, con

$$c^T x^* = \text{mín}\{c^T x \mid Ax \leq b\}$$

La función $c^T x : \Re^n \rightarrow \Re$ es llamada función objetivo y las inecuaciones en el sistema $Ax \leq b$ son llamadas restricciones o desigualdades.

Un problema lineal de minimización puede transformarse en un problema lineal de maximización multiplicando la función objetivo por -1.

Los problemas de optimización lineal han sido muy estudiados. Inicialmente, se utilizó el *método simplex* como algoritmo de resolución. Este algoritmo fue propuesto por George Dantzig en el año 1947. El *método simplex* es el más usado en la programación lineal a pesar de tener complejidad exponencial en el peor caso. En la práctica, este algoritmo se ejecuta en tiempo polinomial, por lo cual, es utilizado con instancias de gran tamaño. Además, desde su creación se dedicó mucho trabajo a mejorar y a optimizar la forma de implementar dicho algoritmo. Muchos esfuerzos también se orientaron a encontrar algoritmos que fueran polinomiales en todos los casos para demostrar que este problema se encontraba dentro de la clase P. En 1979, el matemático ruso Leonid G. Khachiyan propuso el primer algoritmo polinomial para programación lineal. Este algoritmo, denominado *método de elipsoide*, resultó ser ineficiente en la práctica comparado con el *método simplex*. Recién en el año 1984, Narendra K. Karmarkar desarrolla un algoritmo denominado *método de punto interior* que tiene complejidad polinomial, y resulta ser eficiente para aplicarlo a un gran número de problemas reales. Desde entonces, los paquetes de optimización lineal más importantes utilizan variantes de ambos métodos para resolver los problemas lineales.

1.2. Programación Lineal Entera

Desafortunadamente, sólo un conjunto minoritario de problemas pueden ser modelados como problemas de programación lineal con variables continuas. En muchos casos, es necesario la utilización de modelos que incorporen variables cuyos valores deban ser enteros. En consecuencia, al agregar las restricciones de integralidad a un conjunto de variables, obtenemos lo que se denomina un *problema de optimización lineal entera mixta*:

Dada una matriz $A \in \Re^{(m,n)}$, y los vectores $b \in \Re^m, c \in \Re^n$, y un subconjunto $I \subseteq \{1, ..., n\}$, encontrar un vector $x^* \in \Re^n$, con x_i^* entero para todos los $i \in I$ y

$$c^T x^* = \min\{c^T x \mid Ax \leq b, x_i \textit{ entero para todos los } i \in I\} \tag{1.1}$$

En la definición anterior, si se piden que todas las variables sean enteras, es decir, $I = \{1, ..., n\}$, el problema pasa a denominarse **problema de programación lineal entera**. Por otra parte, si las variables del conjunto I sólo toman valores 0 o 1, el problema es llamado **problema de programación lineal 0-1 o binaria**.

Una gran variedad de problemas de optimización combinatoria pueden ser modelados utilizando programación lineal entera. Por ejemplo, las variables 0-1 son usadas para representar una elección binaria. Si se considera que un evento puede o no ocurrir, y se supone que parte del problema es decidir entre esas 2 posibilidades, puede modelarse tal dicotomía usando una variable binaria x que tome

$$x = \begin{cases} 1 \textit{ si el evento ocurre} \\ 0 \textit{ si el evento no occure} \end{cases}$$

Numerosos problemas de optimización combinatoria pertenecen a la clase *NP-Hard*. Gracias al estudio de la llamada *cápsula convexa* de un problema y a la generación de *desigualdades válidas* para ser utilizadas como cortes en un algoritmo de planos de corte, es que dichas modelizaciones han sido útiles para el desarrollo de métodos que resuelven problemas combinatorios difíciles de tamaño cada vez mayores.

1.3. Algoritmos de Plano de Corte

Dado un problema de optimización combinatoria representado por medio de un problema lineal entero, y un conjunto S de puntos factibles tales que:

$$S = \{x \in \Re^n : Ax \leq b, x_i \textit{ entero para todos los } i \in I\}$$

se define su *cápsula convexa* como el poliedro $P_S = conv(S)$. Los puntos extremos de este poliedro representan soluciones factibles del problema combinatorio. Si se conociera la *cápsula convexa* de un problema, se podría resolver su programa lineal asociado y se obtendría inmediatamente el óptimo del problema original. Sin embargo, obtener la caracterización de dicha cápsula no es una tarea sencilla y en los problemas pertenecientes a la clase *NP-Hard* tal descripción incluiría un conjunto exponencial de restricciones salvo que $P = NP$. Es por este motivo que los algoritmos de plano de corte toman especial importancia ya que intentan adecuar, paso a paso, una relajación del problema inicial (acotada

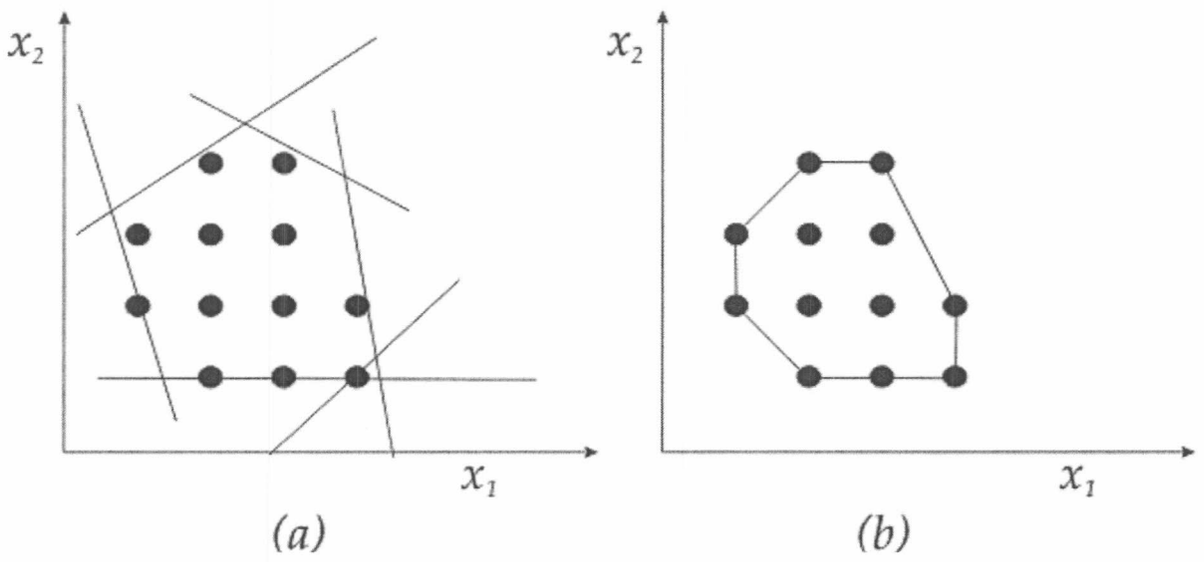

Figura 1.1: (a) Relajación Lineal. (b) Cápsula convexa.

y fácil de calcular mediante paquetes de programación lineal) a su cápsula convexa para obtener el óptimo deseado. En este proceso, sólo se va ajustando la relajación en el sector del poliedro donde pueda ubicarse dicho óptimo.

Las *desigualdades válidas* para un problema de programación entera se define de la siguiente manera:

Dado el problema de programación lineal entera mixta 1.1, una desigualdad $f^T x \leq f_0$ se denomina válida si $f^T \bar{x} \leq f_0$ se cumple para todas las soluciones factibles $\bar{x} \in \{x \mid Ax \leq b, x_i \text{ entero para todos } i \in I\}$.

En la figura 1.1 (a) se ve una relajación lineal de un determinado conjunto S de puntos. En la figura 1.1 (b) se puede observar la cápsula convexa asociada a dicho conjunto S de puntos representantes de un determinado problema de programación lineal entera.

Dado 1.1, la idea básica de los algoritmos de plano de corte puede resumirse en el siguiente pseudocódigo:

1. `Inicializar un sistema de restricciones` (A', b') `con un subconjunto de restricciones del sistema original` (A, b) `que represente una relajación del problema entero modelado.`

2. `Optimizar mediante programación lineal el sistema`
   $c^T x^* = \text{mín}\{c^T x \mid A'x \leq b', x \in \Re^n\}$

3. Si la solución óptima x^* corresponde a una solución factible del problema entero original, entonces mostrar x^* y parar.

4. Buscar una restricción (f, f_0), $f \in \Re^n$, $f_0 \in \Re$ con
   $f^T x^* > f_0$ y
   $f^T y \leq f_0$ para todos los $y \in \{x \mid Ax \leq b, x_i \text{ entero para todos } i \in I\}$.

5. Agregar la desigualdad $f^T x \leq f_0$ al conjunto de restricciones del sistema (A', b') anterior.

6. ir al paso (2).

Es decir, se resuelve mediante programación lineal una relajación del problema original. Si la solución resultante es factible, se encuentra la solución óptima al problema original y finaliza la ejecución. Si no, se debe buscar una nueva restricción, tal que al agregarla a la relajación que se tenía, se elimine la solución no entera recientemente calculada (o entera pero no factible para el problema definido) y se pueda volver a obtener otra solución lineal. De esta manera, el algoritmo itera hasta que se encuentra una solución óptima que lo es también para el problema original.

Las restricciones que definen la cápsula convexa de un poliedro se denominan *facetas del poliedro* y son las desigualdades que deberían, en teoría, ser más apropiadas para ser utilizadas en un algoritmo de planos de corte. Por lo general, una de las tareas fundamentales al intentar aplicar algoritmos de planos de corte es encontrar familias de desigualdades que sean *facetas* y sus correspondientes *algoritmos de separación* eficientes, de manera de poder determinar su posible aplicación en distintas iteraciones del algoritmo principal.

Dependiendo del problema, puede suceder que se arribe a un estado donde no se encuentren más desigualdades válidas para aplicar. Es decir, que no haya más desigualdades conocidas que agregar en el agorítmo de *planos de corte*. En casos donde no se conoce la caracterización completa de la cápsula convexa asociada, es necesario la aplicación de otras herramientas que permitan seguir avanzando en la resolución óptima del problema. Es aquí donde toman importancia los algoritmos de *branch&cut*.

1.4. Algoritmos de *branch&cut*

Los algoritmos de *branch&cut* surgen a partir de la combinación de ideas basadas en los algoritmos de *branch&bound* y los algoritmos de *planos de corte*. Las primeras combinaciones de métodos de *branch&bound* y de desigualdades válidas específicas para un determinado problema fueron presentadas en (Mil76), aplicado al problema del viajante de comercio (Traveling Salesman problem). El uso de *facetas* en algoritmos de *planos de*

corte en combinación con métodos de *branch&bound* fueron inicialmente publicados por Grötschel, Jünger y Reinelt (GJR84). El término *branch&cut* fue introducido por Pardberg y Rinaldi (PR87; PR91), aplicado a un algoritmo para el problema del viajante de comercio.

Básicamente, los algoritmos de *branch&bound* utilizan la idea de dividir y conquistar (divide and conquer) intentando resolver el problema original dividiendolo en subproblemas más pequeños a los que iterativamente se les calcula cotas inferiores y superiores (JRT94; JNS98). En la práctica, este método por sí solo no proporciona buenos resultados para la mayor parte de los problemas, pero da lugar a una de las ideas fundamentales que conforman los algoritmos de *branch&cut*. El funcionamiento de los algoritmos de *branch&cut* para un problema de minimización puede observarse en el siguiente pseudocódigo:

1. Inicializar el árbol de enumeración. Se puede calcular la cota superior global (csg) mediante un algoritmo heurístico.
2. Si las cotas inferior y superior global coinciden, entonces devolver la mejor solución factible encontrada como la óptima del problema y parar.
3. Si no existen nodos abiertos del árbol para procesar, parar e informar que el problema no tiene solución. Si hay algún nodo sin procesar, elegirlo (hay distintas maneras de elegir entre diversos nodos abiertos dependiendo de la estrategia de selección).
4. Resolver la relajación del subproblema que corresponde al nodo elegido. Actualizar la cota inferior global (cig) si corresponde.
   - Si el problema no es factible, eliminar el nodo y volver al paso (2).
   - Si la solución de la relajación supera la cota superior global (csg), ninguna solución de este nodo puede llegar a ser óptimo del problema original. Podar y volver al paso (2).
   - Si la solución de la relajación es solución factible del problema y la cota es inferior a la cota superior global (csg), actualizarla, eliminar el nodo y volver al paso (2).
5. Aplicar el algoritmo de separación para encontrar desigualdades válidas violadas. Si se encuentran desigualdades, agregarlas a la relajación anterior y volver al paso (4). En algunos casos puede ser conveniente ir al paso (5) y hacer branching aunque queden cortes por agregar (depende de la estrategia utilizada).
6. Generar 2 o más nodos del árbol (dependiendo de la regla de branching). Volver al paso (2).

La idea básica de este algoritmo es el cálculo progresivo de cotas inferiores y cotas superiores al óptimo, que se van acercando hasta que se encuentran y constituyen la solución

óptima del problema. Las cotas inferiores se corresponden a los valores de las sucesivas relajaciones lineales que se van resolviendo en cada uno de los nodos del árbol, mientras que las superiores se corresponden con las soluciones factibles encontradas durante dicho recorrido. La velocidad de acercamiento entre ambas cotas es fundamental a la hora de reducir el tamaño del árbol de nodos a analizar. Se utilizan a menudo heurísticas para mejorar rápidamente las cotas obtenidas, tanto en el momento previo a la generación del árbol como durante el análisis de los distintos nodos. Sin embargo, para obtener un algoritmo de *branch&cut* eficiente, es importante lograr un buen balance entre el tiempo destinado a dichas heurísticas y el tiempo destinado a la resolución de sistemas lineales y de generación de cortes.

Hay numerosos detalles adicionales que hacen mucho más compleja la implementación de un algoritmo de *branch&cut*. Entre ellos, se puede mencionar *fijación y seteo de variables* basadas en implicaciones lógicas; aplicación de diversas reglas para la creación y selección de nuevos nodos (branching y selección); métodos de mejoramiento de las soluciones no enteras obtenidas en la iteración, con el fin de obtener soluciones enteras factibles mejores; interrupción de la fase de separación debido a incrementos no significativos en la solución óptima de la relajación analizada; eliminación de desigualdades válidas que se convirtieron en redundantes, con el fin de achicar el tamaño del subproblema a resolver con el optimizador lineal; etc.

Por otra parte, es importante destacar que las primeras definiciones de este tipo de algoritmos sólo permitían cortes que fueran *facetas* (PR87; PR91). Sin embargo, en la actualidad, se utilizan también familias de desigualdades válidas que no son facetas pero que producen igualmente mejoras significativas.

En diversos trabajos publicados hasta el momento, se han desarrollado algoritmos exitosos de *branch&cut* para un gran número de problemas. En muchos casos se ha logrado resolver por optimalidad instancias de gran tamaño, con lo que se posiciona como uno de los métodos exactos más efectivos.

Capítulo 2

El Problema de Ruteo de Vehículos con Capacidades

2.1. Problemas de Ruteo de Vehículos (VRP)

Los problemas de ruteo de vehículos son los problemas relacionados con la distribución o la recolección de mercaderías entre uno o más depósitos y un número finito de clientes.

Los problemas de ruteo de vehículos fueron introducidos inicialmente por Dantzig y Ramser en el año 1959 (DR59). Estos autores introdujeron el problema describiendo una aplicación real relacionada con la distribución de combustible entre estaciones de servicio. Ellos también propusieron la primera formulación matemática y el primer enfoque algorítmico para su resolución. Clarke y Wright, por su parte, propusieron en 1964 (CW64) una heurística golosa que mejoraba el enfoque de Dantzig-Ramser. El problema de ruteo de vehículos es uno de los problemas de la optimización combinatoria que mayor importancia ha ganado en el transcurso de los últimos años.

Entre las aplicaciones más comunes de ruteo de vehículos, encontramos problemas como el transporte escolar, la distribución de productos entre un depósito central y clientes, la recolección de residuos, el transporte de personas discapacitadas, la entrega de correspondencia, etc.

En la mayoría de las aplicaciones suele minimizarse la distancia total recorrida, pero a veces otros objetivos son tenidos en cuenta. Por ejemplo, aquellos que relacionan variables como el número de vehículos, los costos de transporte, penalidades, cargas, tiempo y otras combinaciones de estas variables.

Hay una gran variedad de problemas de ruteo de vehículos y muchos de ellos son anali-

zados en la literatura. Algunos ejemplos son *Ruteo de Vehículos con Capacidades (CVRP)*, *Ruteo de Vehículos con restricción de distancias (DCVRP)*, *Ruteo de Vehículos con ventanas de tiempo (VRPTW)*, *Ruteo de Vehículos con retornos (VRP with Backhauls - VRPB)* y *Ruteo de Vehículos con recolección y entrega (Pickup and Delivery - VRPPD)*.

En general, los problemas de ruteo de vehículos son problemas *NP-Hard* como muchos otros problemas incluídos dentro de la optimización combinatoria.

Dentro de los enfoques propuestos para resolver VRP se encuentran:

- Algoritmos de *Branch&bound* con distinto tipo de relajaciones y cotas
- Algoritmos de *Branch&cut*
- Algoritmos basados en cubrimiento de conjuntos (Set Covering)
- Heurísticas clásicas
- Metaheurísticas

Una descripción más completa y detallada de cada una de las variantes y los distintos métodos de resolución puede encontrarse en la recopilación de Paolo Toth y Daniele Vigo (TV01).

2.2. El problema de Ruteo de Vehículos con Capacidades (CVRP)

Los problemas de Ruteo de Vehículos con Capacidades pueden verse como la conjunción de dos conocidos problemas, a saber, el problema del Viajante de Comercio (Traveling Salesman Problem - TSP) y una variante del problema de la mochila (Bin Packing Problem - BPP), ambos problemas pertenecientes a la clase *NP-Hard*.

Esta es una clase de VRP que tiene un sólo depósito, un número fijo de vehículos idénticos disponibles donde cada uno tiene una capacidad máxima y una restricción adicional que indica que cada cliente debe ser visitado exactamente una vez. Además, los requerimientos de los clientes son conocidos de antemano y deben satisfacerse sin ser partidos, es decir, cada cliente es atendido por un sólo camión. El objetivo de este problema es minimizar la distancia total recorrida por la flota de vehículos. En la figura 2.1 puede verse un ejemplo gráfico de la resolución óptima de una instancia de 135 nodos y 7 vehículos propuesta por Fisher (Fis95).

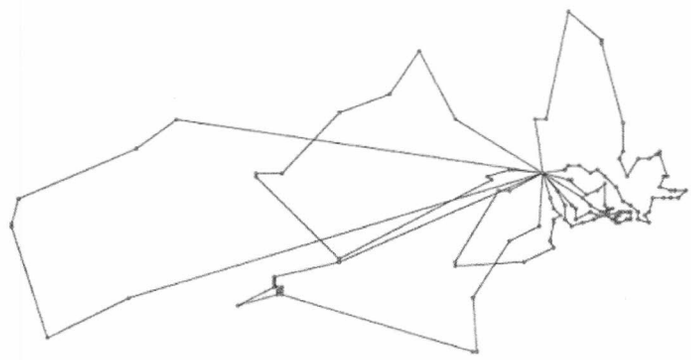

Figura 2.1: Resolución óptima de la instancia F-n135-k7

CVRP es el miembro más estudiado de la familia de problemas de ruteo de vehículos, sin embargo, su complejidad hace que los avances obtenidos dentro de los métodos exactos sean bastante limitados y permitan resolver por optimalidad instancias relativamente pequeñas, comparadas con las resueltas para otros problemas de optimización combinatoria como el TSP.

Teorema. *El problema de Ruteo de Vehículos con Capacidades es* NP-Hard.

Demostración. El problema de Ruteo de Vehículos con Capacidades es una generalización del Problema del Viajante de Comercio (TSP). Si se considera CVRP con un sólo vehículo con capacidad ilimitada, nos encontramos frente al problema TSP. Como TSP es *NP-Hard*, el problema en cuestión también lo será.

Hay muchos modelos de programación lineal entera que pueden usarse para modelar los problemas CVRP. Los modelos básicos propuestos en la literatura pueden agruparse en tres clases bien definidas: Modelos de flujo de vehículos, Modelos de flujo de *commodities* y Modelos de partición de conjuntos (TV01). Este trabajo se circunscribe dentro de la clase de modelos de flujo de vehículos.

2.2.1. Modelos de flujo de vehículos

Los modelos de flujo de vehículos son los modelos más frecuentemente usados para las versiones básicas de CVRP. Estos modelos se basan en los problemas de flujo en redes. El problema básico de flujo en redes (Network Flows en inglés) puede ser representado por un grafo dirigido, con capacidades definidas en sus ejes y con 2 nodos especialmente diferenciados llamados *nodo fuente* y *nodo destino*. La idea fundamental de las redes de flujo

es que en todos los nodos de la red menos en los 2 nodos anteriormente citados, el flujo ingresante debe ser exactamente igual al saliente (conservación de flujo). El *nodo fuente* tiene sólo flujo de salida y el *nodo destino* tiene sólo flujo de entrada. Este grafo configura la red de flujo donde se asignará un flujo válido que satisfaga todas las capacidades y las restricciones de conservación de flujo. En la figura 2.2 (a) puede verse un ejemplo de red de flujo con las capacidades indicadas sobre cada uno de los ejes del grafo.

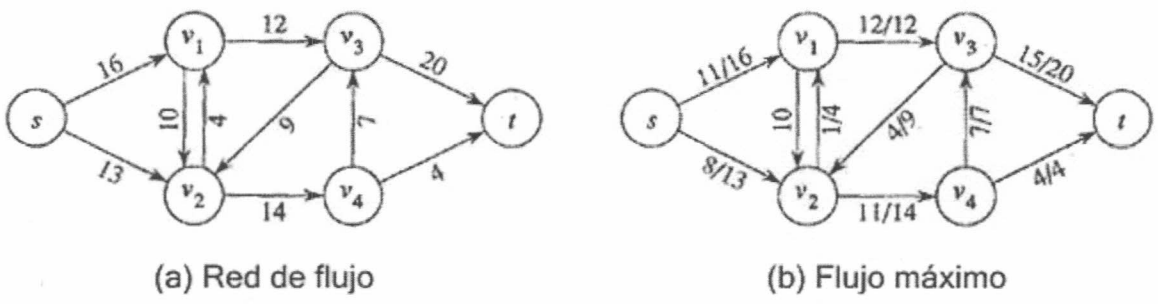

Figura 2.2: Ejemplo de Red de flujo y asignación de flujo máximo

Problema de flujo en redes. Dado un grafo $G(V, E)$, siendo V el conjunto de nodos y E el conjunto de ejes, s el *nodo fuente* y t el *nodo destino*, se define para cada eje (u, v) la capacidad máxima de flujo que puede transitar por dicho eje como $c(u, v)$.
Un flujo en G es una función $f : V \times V \rightarrow \Re$ que cumple las siguientes restricciones:

- **Restricción de Capacidad:** $f(u, v) \leq c(u, v) \quad \forall (u, v) \in E$
- **Simetría:** $f(u, v) = -f(v, u)$
- **Conservación de flujo:** $\sum_{u \in V \setminus \{s,t\}} f(u, v) = \sum_{v \in V \setminus \{s,t\}} f(u, v)$

El valor del flujo es $|f| = \sum_{v \in V} f(s, v)$

Entre las aplicaciones más frecuentes se encuentra el problema de flujo máximo, que busca una asignación de flujo f que maximice el valor del flujo $|f|$. Entre los algoritmos más conocidos para resolver este problema, se encuentra el famoso algoritmo de L. R. Ford y D. R. Fulkerson que basa su estrategia en encontrar *caminos de aumento* entre el nodo s y el nodo t en pasos sucesivos hasta encontrar el flujo máximo. Existen numerosas variantes de este algoritmo (ej: algoritmo Edmonds-Karp) que demuestran que es un problema de complejidad polinomial. En la figura 2.2 (b) puede observarse el flujo máximo del ejemplo

mostrado en (a).

Basados en el problema de flujo máximo, surgen problemas como el de *flujo con costo mínimo* donde además de la red de flujo cada eje tiene asociado un costo por unidad de flujo que lo atraviesa; *problemas de circulación* donde se asocia un costo y un flujo mínimo a cada eje; *problemas de flujo con múltiples commodities* (multi-commodity flow problem) donde deben enviarse simultáneamente múltiples mercaderías a través de la red; y una lista innumerable de problemas de asignación, matching y transporte.

Los problemas de flujo en redes tienen aplicaciones en una gran variedad de escenarios como ser simulaciones de tráfico en caminos, fluídos en cañerías y problemas hidráulicos, corrientes en circuitos eléctricos, etc. Puede encontrarse una explicación detallada de todos los conceptos de flujo en redes en (AMO93).

Los modelos de flujo de vehículos surgen como aplicación directa de las ideas de redes de flujo, especialmente la de flujo con costo mínimo. Este tipo de modelos usa en su formulación variables enteras asociadas a cada eje del grafo de rutas para representar el número de veces que el eje es atravesado por un vehículo. Es decir, la capacidad máxima de cada eje es 1. El grafo de rutas representa a los clientes con nodos y sus conexiones físicas con los ejes. Para convertir el problema de ruteo de vehículos en un problema de flujo tradicional se agregan los nodos s y t para representar la fuente y el destino, que coinciden con la función del depósito. En la mayoría de los modelos de flujo de vehículos, ambos nodos se fusionan en un sólo nodo representando al depósito central. De todos modos, al incluir restricciones adicionales (como la restricción de capacidad) el problema resulta ser *NP-Hard*.

En la literatura se puede encontrar dos clases de modelos de flujo de vehículos: con variables de 2 índices y con variables de 3 índices.

2.2.1.1. Modelos de flujo de vehículos de 2 índices

Los modelos de flujo de 2 índices se atribuyen normalmente a Laporte y Nobert (LN83; LN87). Sea el grafo $G = (V, A)$ un grafo completo dirigido con nodos $V = 0, ..., n$ y ejes A. Los nodos $1, ..., n$ corresponden a los nodos clientes, mientras que el nodo 0 corresponde al nodo depósito. Sea c_{ij} un costo no negativo asociado con cada eje $(i, j) \in A$ que representa el costo de viajar desde el nodo i hasta el nodo j.
Los modelos con variables de 2 índices usan variables binarias vinculadas a los ejes del grafo de rutas, indicando si el eje es atravesado o no en la solución óptima. Generalmente, suele considerarse el grafo de rutas como un grafo completo, es decir que todos los clientes tienen conexión directa con el resto de los clientes y con el depósito central. Esta hipótesis puede realizarse sin inconveniente ya que el grafo de rutas asociado a cualquier problema de ruteo de vehículos puede ser transformado en su equivalente completo agregando los ejes faltantes

con costo infinito.

Los modelos pueden ser simétricos, si cada tramo tiene la misma distancia para una dirección que para la opuesta, o asimétricos en caso contrario. En caso de ser simétricos, es necesario utilizar una variable binaria para representar cada tramo, mientras que para los asimétricos se necesitan dos variables por tramo para indicar la dirección del mismo que se selecciona.

A continuación, se muestra una de las formulaciones de flujo de vehículos de 2 índices más utilizadas para la versión asimétrica de CVRP (ACVRP):

$$min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{2.1}$$

sujeto a

$$\sum_{i\in V} x_{ij} = 1 \qquad \forall j \in V \setminus \{0\} \tag{2.2}$$

$$\sum_{j\in V} x_{ij} = 1 \qquad \forall i \in V \setminus \{0\} \tag{2.3}$$

$$\sum_{i\in V} x_{i0} = K \tag{2.4}$$

$$\sum_{j\in V} x_{0j} = K \tag{2.5}$$

$$\sum_{i\notin S, j\in S} x_{ij} \geq r(S) \qquad \forall S \subseteq V \setminus \{0\}, S \neq \emptyset \tag{2.6}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in V \tag{2.7}$$

En esta formulación la variable x_{ij}, toma el valor 1 si el eje $(i,j) \in A$ pertenece a la solución óptima (indicando que el eje (i,j) es utilizado por algún vehículo), y 0 en caso contrario. La constante K representa al número fijo de vehículos usados.
Por otra parte, $r(S)$ representa la cantidad mínima de vehículos necesarios para servir a los nodos del conjunto S. Su cálculo exacto puede efectuarse resolviendo el problema de *Bin Packing* (BPP) para el conjunto S. Sin embargo, debido a su complejidad (*NP-Hard*), normalmente $r(S)$ es reemplazado por la cota inferior trivial $\lceil d(S)/C \rceil$, representando $d(S)$ la demanda total de los nodos de S y C la capacidad de los camiones.

En primer lugar, lo que se quiere minimizar es la distancia total recorrida por los vehículos. Por eso, la función objetivo minimiza la suma de las distancias utilizadas (2.1). Las restricciones (2.2) y (2.3) imponen que exactamente un vehículo ingrese y otro salga de cada nodo (comúnmente denominadas restricciones de conservación de flujo). Por otro lado,

las restricciones (2.4) y (2.5) exigen que el grado del nodo 0 sea exactamente la cantidad de vehículos K, es decir, K ejes de salida y K de llegada deben valer 1. Las restricciones (2.6) son las llamadas restricciones de capacidad (o restricciones de eliminación de subtour generalizadas) y requieren al mismo tiempo la conectividad de la solución y la correcta utilización de las capacidades de los vehículos involucrados. Este conjunto de restricciones tiene una cantidad exponencial de elementos. Se lo suele implementar, en los algoritmos de *branch&cut*, como cortes agregados durante la ejecución del programa. La separación de dichas restricciones no ha podido ser implementada hasta el momento de manera polinomial, por lo que diversas heurísticas se suelen utilizar para identificarlas. Por último, las restriciones (2.7) representan las condiciones de integralidad necesarias sobre las variables utilizadas.

Este tipo de modelos de 2 índices usa una cantidad de variables binarias del orden de $O(n^2)$ siendo n la cantidad de clientes a servir.

Los modelos con variables de 2 índices no pueden manejar situaciones prácticas donde los vehículos deban ser identificados o donde el costo de la solución dependa del orden de las rutas, ya que en el modelo no hay identificación directa de los vehículos involucrados. Para poder conocer posteriormente las rutas realizadas por cada vehículo, se debe recorrer la solución óptima obtenida, partiendo desde el depósito y visitando a cada uno de los clientes, uno a uno, hasta completar la ruta. Para ello se utiliza la información provista por los ejes que se encuentran en 1. Además, en la formulación mostrada no hay manera de incluir restricciones de capacidad específicas sobre cada vehículo porque los mismos no están individualizados.

2.2.1.2. Modelos de flujo de vehículos de 3 índices

Algunos de los problemas mencionados en cuanto a la falta de identidad de los vehículos puede ser solucionado utilizando variables de 3 índices para identificar a los vehículos que atraviesan cada eje. Esta solución usa una cantidad de variables binarias de 3 índices del orden de $O(Kn^2)$ más $O(Kn)$ variables binarias de 2 índices usadas como variables de decisión. A continuación se presenta una formulación de ese tipo basada en los modelos de Fisher y Jaikumar (FJ81), y Fischetti, Salazar y Toth (FGT95):

$$min \sum_{(i,j)\in A} c_{ij} \sum_{k=1}^{K} x_{ijk} \tag{2.8}$$

sujeto a

$$\sum_{k=1}^{K} y_{ik} = 1 \qquad \forall i \in V \setminus \{0\} \tag{2.9}$$

$$\sum_{k=1}^{K} y_{0k} = K \tag{2.10}$$

$$\sum_{j\in V} x_{ijk} = \sum_{j\in V} x_{jik} = y_{ik} \qquad \forall i \in V, k = 1 \dots K \tag{2.11}$$

$$\sum_{i\in V} d_i y_{ik} \leq C \qquad \forall k = 1 \dots K \tag{2.12}$$

$$\sum_{(i\in S, j\notin S)} x_{ijk} \geq 2y_{hk} \qquad \forall S \subseteq V \setminus \{0\}, h \in S, \quad k = 1 \dots K \tag{2.13}$$

$$y_{ik} \in \{0,1\} \qquad \forall i \in V, k = 1 \dots K \tag{2.14}$$

$$x_{ijk} \in \{0,1\} \qquad \forall i,j \in V, k = 1 \dots K \tag{2.15}$$

En este modelo se utilizan las variables de 3 índices x_{ijk} con el mismo significado anterior pero, en lugar de utilizar dos variables por cada eje (una para cada dirección), se pasa a usar $2K$ variables por eje identificando el vehículo que utiliza dicho eje. Por ejemplo, si el vehículo $k \leq K$ atraviesa el eje (i,j) en la solución óptima, entonces x_{ijk} valdrá 1 mientras que el resto de los multiejes x_{ijm} con $m \neq k$ valdrán 0. En este modelo, el tercer índice representa el multieje que identifica al vehículo.

Por otro lado, debe agregarse también un nuevo conjunto de variables de 2 índices y_{ik} para establecer la lógica que permita asegurar que si un eje es atravesado por un vehículo, entonces los nodos que dicho eje involucra están en la misma ruta.
La variable y_{ik} valdrá 1 si el nodo (cliente) i pertenece a la ruta del vehículo k y 0 en caso contrario.
En la formulación anterior se utilizan las siguientes familias de restricciones:

- (2.9) para exigir que cada nodo sea visitado por un solo vehículo.
- (2.10) para indicar que el nodo 0 (depósito) es visitado por todos los vehículos.
- (2.11) para relacionar las variables x con las y, exigiendo que en cada nodo entre un vehículo y salga otro y, además, que el vehículo asignado a los ejes que ingresan y salen del nodo coincida con el número de vehículo asignado al nodo.

- (2.12) para asegurar que cada vehículo no exceda la capacidad máxima respectiva.
- (2.13) para establecer que, si al tomar cualquier subconjunto de nodos S, existe algún nodo de dicho conjunto que pertenece a la ruta k, entonces en el corte de salida de S debe haber por lo menos 1 eje que valga 1 en la ruta k. Estas restricciones junto a las (2.12) reemplazan a las restricciones de capacidad del modelo de 2 índices anterior (2.6).

Los modelos de flujo de 3 índices han sido usados principalmente para modelar versiones con más restricciones o características especiales que CVRP, tales como VRPTW, debido a su gran flexibilidad para incorporar requerimientos sobre los vehículos involucrados y su relación con los clientes servidos. La gran desventaja frente a los modelos de 2 índices es el gran incremento de variables que significa representar en cada eje del grafo una variable para cada vehículo (K variables por eje frente a 1 variable por eje en el modelo de 2 índices) (TV01). En este trabajo se presenta una familia de modelos de flujo de 3 índices que usa solamente una cantidad de variables del orden $O(n^2)$, sin perder además las ventajas y la flexibilidad de las formulaciones de 3 índices mencionadas. Como desventaja, se obtiene un modelo con un número mayor de restricciones.

2.2.2. Modelos de flujo de *commodities* (commodity flow models)

Los modelos de flujo de *commodities* fueron inicialmente propuestos por Garvin (GCJS57) para un problema de repartición de combustibles. Posteriormente fueron extendidos por Gavish y Graves (GG79; GG82) para variantes de TSP y VRP. Estas formulaciones, además de las variables usadas para las formulaciones de flujo de 2 índices mostradas anteriormente, requieren un nuevo conjunto de variables continuas asociadas a los ejes para representar la cantidad de mercaderías que fluyen a través de los mismos. Para una completa disquisición sobre los distintos trabajos publicados con modelos de esta clase referirse a (TV01).

Para presentar la formulación se deberá extender el grafo anterior a un nuevo grafo $G' = (V', A')$ obtenido de agregar el nodo $n+1$ (copia del nodo depósito) al conjunto de nodos V. Las rutas son ahora caminos desde el nodo 0 al nodo $n+1$. Dos variables de flujo no negativas, y_{ij} y y_{ji} se asocian a cada uno de los ejes $(i,j) \in A'$. Si un vehículo viaja de i a j, entonces y_{ij} y y_{ji} dan la carga del vehículo y la capacidad residual del mismo al atravesar el eje. Es decir, $y_{ji} = C - y_{ij}$ con C representando la capacidad del vehículo. Los roles se invierten si el vehículo viaja de j a i. Además, la ecuación $y_{ij} + y_{ji} = C$ se cumple para todos los ejes $(i,j) \in A'$ que formen parte de la solución.

Para cualquier ruta de una solución factible, el flujo de las variables definen dos caminos dirigidos, uno del nodo 0 al nodo $n+1$, cuyas variables representan la carga del vehículo actualizada a lo largo del camino, y otro del nodo $n+1$ al nodo 0, cuyas variables representan

la capacidad residual del vehículo. En otras palabras, se puede pensar como dos vehículos distintos: uno partiendo del nodo 0 hacia el nodo $n+1$ con la suficiente mercadería como para ir satisfaciendo las necesidades de los clientes que recorre antes de abordar vacío al nodo $n+1$. El otro, parte vacío del nodo $n+1$ y recolecta las mercaderías demandadas por cada uno de los clientes hasta llegar al nodo 0 con el total de la demanda de la ruta. Un ejemplo con 3 clientes y capacidad $C = 15$ puede observarse en la figura 2.3 donde las demandas se encuentran notadas al costado de los nodos.

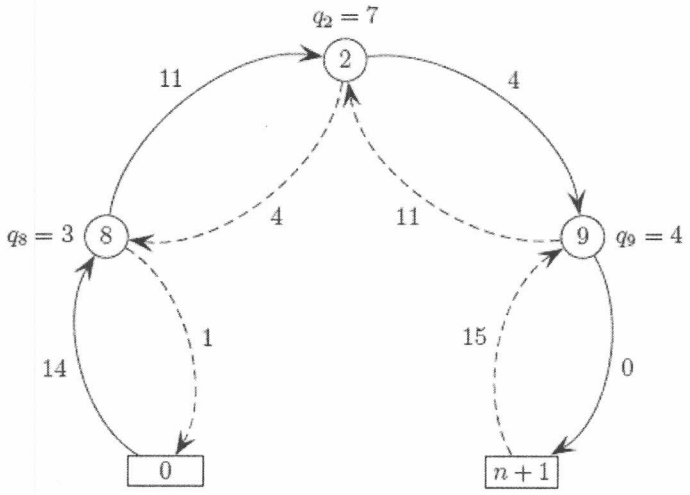

Figura 2.3: Ejemplo de una ruta de flujo de commodities (C=15)

Como en los modelos de flujo de vehículos con 2 índices, a cada eje $(i, j) \in A'$ se asocia una variable binaria x_{ij} que vale 1 si el eje es utilizado en la solución óptima, y 0 en caso contrario. A continuación se presenta la formulación para ACVRP (TV01):

$$min \sum_{(i,j)\in A'} c_{ij}x_{ij} \tag{2.16}$$

sujeto a

$$\sum_{j\in V'}(y_{ji} - y_{ij}) = 2d_i \qquad \forall i \in V' \setminus \{0, n+1\} \tag{2.17}$$

$$\sum_{j\in V'\setminus\{0,n+1\}} y_{0j} = d(V \setminus \{0, n+1\}) \tag{2.18}$$

$$\sum_{j\in V'\setminus\{0,n+1\}} y_{j0} = KC - d(V \setminus \{0, n+1\}) \tag{2.19}$$

$$\sum_{j\in V'\setminus\{0,n+1\}} y_{n+1\ j} = KC \tag{2.20}$$

$$y_{ij} + y_{ji} = C_{x_{ij}} \qquad \forall (i,j) \in A' \tag{2.21}$$

$$\sum_{i\in V'}(x_{ij} + x_{ji}) = 2 \qquad \forall i \in V' \setminus \{0, n+1\} \tag{2.22}$$

$$y_{ij} \geq 0 \qquad \forall (i,j) \in A' \tag{2.23}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A' \tag{2.24}$$

Las restricciones de conservación de flujo (2.17) determinan que la diferencia entre la suma de los *commodities* de las variables de flujo de entrada y las de salida de cada nodo i sea dos veces la demanda de i. Por otro lado, las restricciones (2.18-2.20) exigen los valores válidos para las variables de flujo de *commodities* incidentes al depósito. Por último, las restricciones (2.21) y (2.22) establecen la relación entre las variables de flujo de vehículos y las variables de flujo de *commodities* por un lado, y el grado de los nodos por el otro.

2.2.3. Modelos de partición de conjuntos (Set-Partitioning Models)

Este tipo de formulaciones, propuestas inicialmente por Balinski y Quandt (BQ64), utilizan un número exponencial de variables binarias. Cada variable se asocia a un posible circuito de G. Más específicamente, sea $H = \{H_1, ..., H_q\}$ la colección de todos los circuitos de G, cada uno correspondiendo a una ruta factible con $q = \mid H \mid$. Cada circuito H_j tiene un costo c_j asociado. Además, los coeficientes binarios a_{ij} tienen valor 1 si el nodo i es visitado (o cubierto en la jerga de partición de conjuntos) por una ruta H_j y 0 en caso contrario. Las variables binarias $x_j, j = 1, ..., q$ son iguales a 1 si y solo sí el circuito H_j se selecciona en la solución óptima. De esta manera se puede definir el modelo como:

$$min \sum_{j=1}^{q} c_j x_j \tag{2.25}$$

sujeto a

$$\sum_{j=1}^{q} a_{ij} x_j = 1 \qquad \forall i \in V' \setminus \{0\} \tag{2.26}$$

$$\sum_{j=1}^{q} x_j = K \tag{2.27}$$

$$x_j \in \{0,1\} \qquad \forall j = 1, ..., q \tag{2.28}$$

Las restricciones (2.26) requieren que cada cliente i sea visitado (cubierto) por exactamente uno de los circuitos seleccionados. Por otro lado, la restricción (2.27) establece que se deben elegir K circuitos.

Este es un modelo muy general que permite ser usado en numerosas variantes de CVRP. Con el agregado de algunas restricciones puede utilizarse, por ejemplo, para ruteo de vehículos con ventanas de tiempo.

Debido a la imposibilidad de generar todos los posibles circuitos (columnas) y manejar modelos que rápidamente crecen a millones de variables, se enfoca la resolución de este tipo de formulaciones utilizando técnicas de *generación de columnas* para resolver relajaciones lineales del modelo original. Puede verse una descripción pormenorizada de los métodos y las técnicas utilizadas para atacar estos problemas en (TV01).

Capítulo 3

Nuevo modelo para el problema de Ruteo de Vehículos con Capacidades

3.1. Nuevo modelo de flujo de vehículos de 3 índices

3.1.1. Construcción del nuevo modelo

El objetivo de este trabajo fue plantear, en primer lugar, modelos de flujo para el problema de Ruteo de Vehículos con Capacidades. Las ideas surgidas se circunscriben a la clase de modelos de flujo de 3 índices.

Como introducción, se describirá cómo surge el nuevo modelo de programación lineal entera basado en los modelos de 2 índices tradicionales y luego, se presentará la formulación de programación lineal entera que lo representa.

En los modelos de 2 índices, se utiliza un nodo 0 para representar al depósito y n *nodos clientes* a los cuales servir. En este nuevo modelo se reemplazará el depósito 0 por 2 nodos o y d representando el nodo depósito de salida (origen) y el nodo depósito de llegada (destino). Aquí, se ha utilizado o y d en lugar de la nomenclatura tradicional s y t que representan el origen y el destino en un modelo de flujo en redes (en inglés, s: source - t: target). Con los cambios mencionados se obtiene un modelo de flujo con el aspecto del ejemplo de la Figura 3.1. En dicha figura, q_i representa cada uno de los nodos clientes.

Para obtener el nuevo modelo, se va a partir cada uno de los *nodos clientes* en 3 subnodos. Llámese a , b y c a cada uno de ellos. Puede representarse el nodo anterior q_j como los nodos q_{ja}, q_{jb} y q_{jc} uniéndolos con ejes como se muestra en la figura 3.2. En la figura se representa entre paréntesis la capacidad mínima y máxima que debe satisfacer cada uno de los ejes. Es decir, los nodos q_{ja}, q_{jb} estarán unidos por K multiarcos (o multiejes). Cada multiarco representará una variable $0-1$ e indicará cuál es el vehículo que sirve al

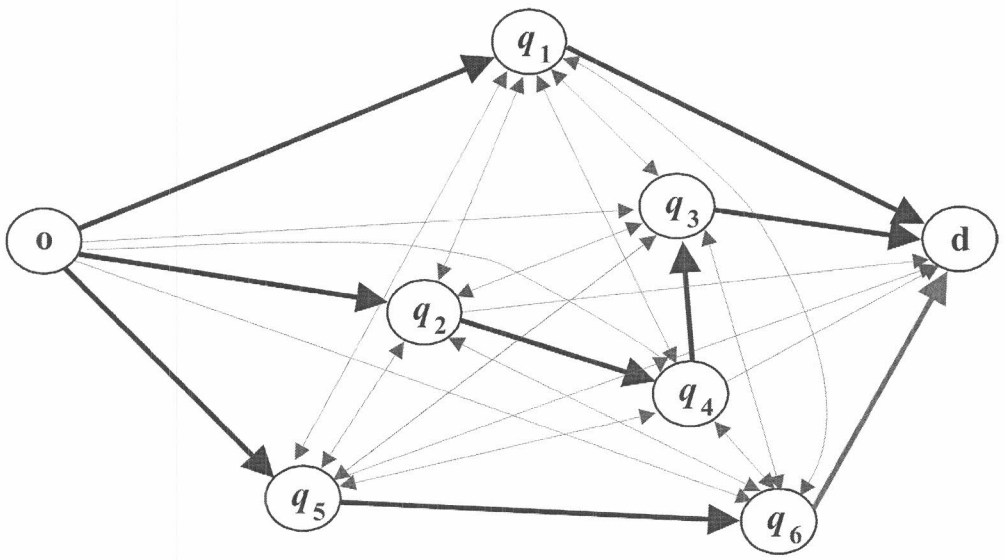

Figura 3.1: Modelo de 2 índices asimétrico

cliente j. Por ejemplo, si el vehículo que sirve al cliente j es el 3, entonces el multiarco 3 deberá contener el valor 1 mientras que todos los otros multiarcos tendrán el valor 0. Por último, se conectará los subnodos q_{jb}, q_{jc} con un eje ficticio que siempre deberá valer 1. Esto permite, al aplicar las anteriores restricciones de conservación de flujo (2.2) y (2.3), que un sólo vehículo pase por cada cliente.

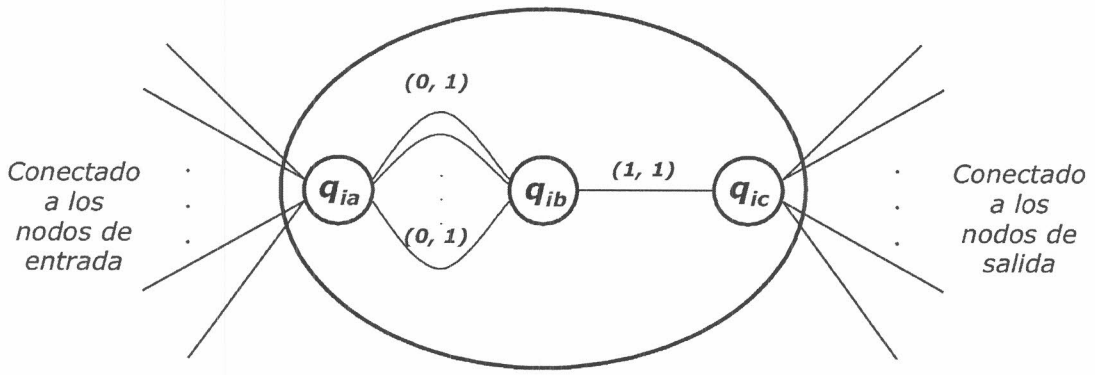

Figura 3.2: Partición Nodo Cliente

A continuación, se van a agregar K nuevos nodos llamados *nodos vehículo* o nodos ca-

mión que representarán a los vehículos usados. Cada uno de dichos nodos estará dividido, a su vez, en 2 subnodos a y b como se muestra en la Figura 3.3, conectados por K multiarcos distintos. Como en el caso de los *nodos cliente*, cada multiarco indicará cuál será el vehículo activado. Por ejemplo, en el *nodo vehículo* que representa al vehículo 4, el multiarco 4 podrá variar entre los valores $0-1$ mientras que el resto de los multiarcos siempre tendrán el valor 0. A diferencia del modelo de 2 índices mostrado inicialmente, aquí se permitirá el uso de 1 *a* K vehículos, es decir que la cantidad de vehículos no será constante y deberá tomarse como una variable más a optimizar en el problema. Esto tendrá la consecuencia de que determinados vehículos pueden no ser utilizados en la solución óptima y otorga al modelo mayor flexibilidad ya que permite adaptarse a un número más grande de variantes.

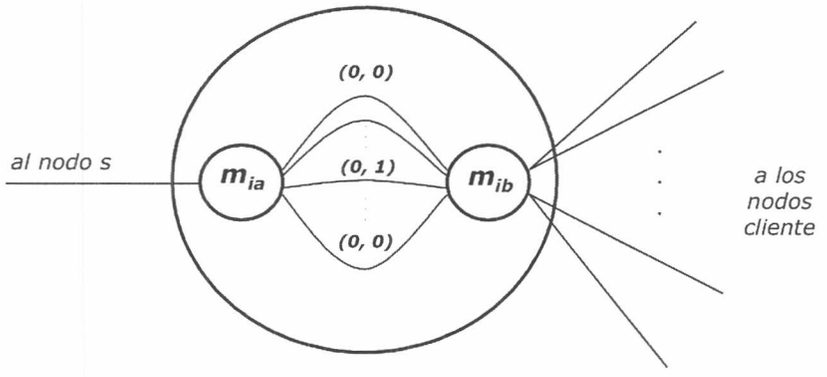

Figura 3.3: Partición Nodo Vehículo

En la Figura 3.4 se muestra un esquema de cómo queda el modelo anterior después de las particiones y del agregado de los nuevos nodos vehículo.

De esta manera, se define la modificación principal que da lugar al nuevo modelo de flujo de vehículos de 3 índices.

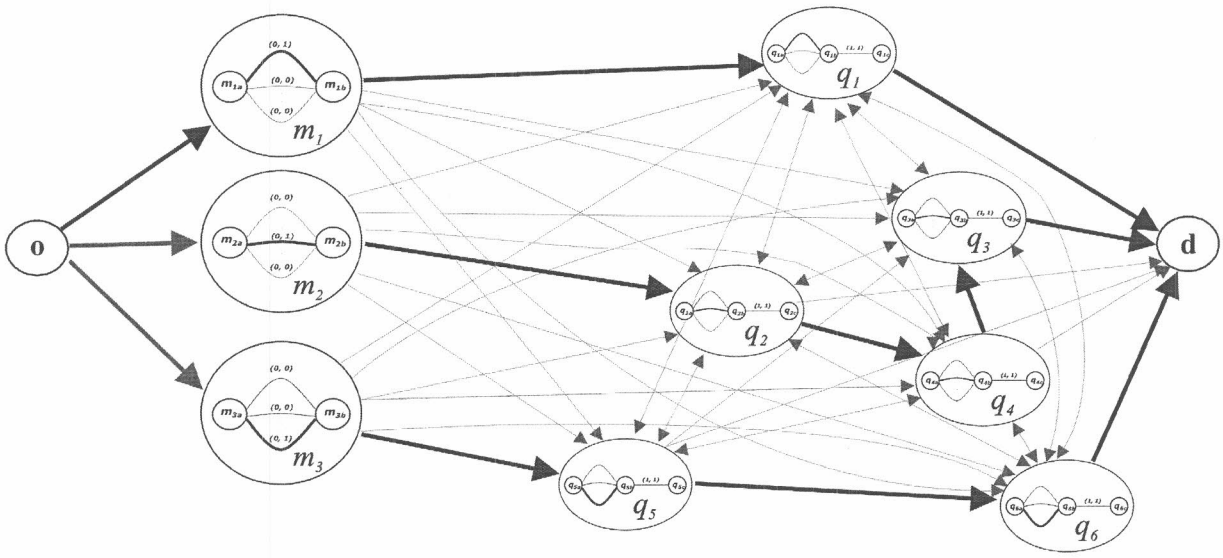

Figura 3.4: Nuevo modelo de 3 índices

3.1.2. Formulación del modelo

Sea el grafo $G = (V, A)$ del modelo de 2 índices mostrado anteriormente, se realizará una transformación y se obtendrá el grafo $G' = (V', A')$. El conjunto de los nodos V anterior será modificado en un nuevo conjunto de nodos V' donde se reemplazará el nodo depósito 0 por 2 nodos o y d representando el origen y la finalización de cada una de las rutas. Además, se agregarán $2K$ nuevos nodos para identificar a cada uno de los vehículos, como se explicó anteriormente (un subnodo m_a y otro m_b para cada uno). Llámese M al conjunto de dichos nodos. Cada uno de los clientes $i \in V \setminus \{0\}$ se convertirá en 3 nuevos nodos q_{ia}, q_{ib} y q_{ic}. Se llamará Q al conjunto de los nodos clientes formados de esta manera. Así, se obtiene el nuevo conjunto de nodos $V' = \{o, t\} \cup Q \cup M$ donde $Q = \{q_{ia}, q_{ib}, q_{ic} \mid \forall i \in V \setminus \{0\}\}$ y $M = \{m_{ia}, m_{ib} \mid i = 1...K\}$.
El nuevo conjunto de ejes A' queda constituido de la siguiente manera:

$A' = A_2 \cup A_3$

$A_2 = \{(o, m_{ka}) \mid k = 1...K\} \cup \{(m_{kb}, q_{ia}) \mid k = 1...K, \forall i \in V \setminus \{0\}\} \cup \{(q_{ib}, q_{ic}) \mid \forall i \in V \setminus \{0\}\} \cup \{(q_{ic}, q_{ja}) \mid \forall i, j \in V \setminus \{0\}, i \neq j\} \cup \{(q_{ic}, d) \mid \forall i \in V \setminus \{0\}\}$

$A_3 = \{(m_{ka}, m_{kb}, l) \mid k, l = 1...K\} \cup \{(q_{ia}, q_{ib}, k) \mid \forall i \in V \setminus \{0\}, k = 1...K\}$

A_2 representa a los ejes de 2 índices y A_3 a los de 3 índices.

Por otra parte, se redefine c_{ij} de manera que mantenga su valor original cuando represente a ejes de 2 índices que conecten a clientes entre sí, o conecten clientes al depósito (c_{oi} y c_{id}). Para todos los otros nuevos ejes creados (incluso los de 3 índices) el costo será 0.

Antes de presentar la formulación, es necesario explicar la notación adoptada. Las variables x_{ij} representarán la conexión del nodo i al nodo j (ejes (q_{ic}, q_{ja}) y ejes (q_{ic}, d)). En este caso, j podrá ser *nodo cliente* o *nodo vehículo* (d). Las variables de 3 índices x^k_{ii} representarán al multieje k del *nodo cliente* i (ejes (q_{ia}, q_{ib}, k)). Las variables x^k_{oo} representarán al multiarco k del *nodo vehículo* k (ejes (m_{ka}, m_{kb}, k)). Debido a que el resto de los multiejes de los *nodos vehículo* valen siempre 0, fueron eliminados de la formulación.
Los ejes que interconectan el nodo depósito de origen (o) con los *nodos vehículo* serán notados como x_{ok}, siendo k el número de vehículo al que se conecta el nodo depósito (ejes (o, m_{ka})).
Se utilizará x^k_{oi} para representar la interconexión entre los *nodos vehículo* k y los *nodos cliente* i (ejes (m_{kb}, q_{ia})). Además, se representará con x_{ii}, $i \in Q$, a los ejes (q_{ib}, q_{ic}) definidos anteriormente. Como ya se explicó, dichos ejes son innecesarios y se han agregado para cumplir con la idea de un modelo clásico de flujo. Estos ejes, ya que siempre tomarán el valor 1, podrán ser reemplazados por restricciones en futuros modelos.

Con estas definiciones, la formulación correspondiente al nuevo modelo queda de la siguiente manera:

$$min \sum_{i,j \in Q} c_{ij} x_{ij} + \sum_{\substack{i \in Q \\ k \in M}} c_{oi} x_{oi}^k + \sum_{i \in Q} c_{id} x_{id} \tag{3.1}$$

sujeto a

$$\sum_{a \in \delta^+(\{i\})} x_a - \sum_{a \in \delta^-(\{i\})} x_a = 0 \qquad \forall i \in V \setminus \{o, d\} \tag{3.2}$$

$$\sum_{i \in Q} d_i \, x_{ii}^k - x_{ok} \, cap(k) \leq 0 \qquad k = 1...K \tag{3.3}$$

$$x_{ii}^k - x_{jj}^k + x_{ij} + x_{ji} \leq 1 \qquad \forall i, j \in Q \; i \neq j; k = 1...K \tag{3.4}$$

$$x_{oo}^k + x_{ii}^k - 2x_{oi}^k \geq 0 \qquad \forall i \in Q; k = 1...K \tag{3.5}$$

$$\sum_{\substack{i,j \in S \\ i \neq j}} x_{ij} \leq \mid S \mid -1 \qquad 2 \leq \mid S \mid \leq n - 1 \tag{3.6}$$

$$x_{oi}^k + x_{jd} + x_{jj}^k \leq 2 \qquad \forall i, j \in Q \; i > j; k = 1...K \tag{3.7}$$

$$x_{oi}^k + x_{oj}^{k+1} \leq 1 \qquad \forall i, j \in Q \; i > j; \quad k = 1 \ldots K - 1 \tag{3.8}$$

$$x_{ii} = 1 \qquad \forall i \in Q \tag{3.9}$$

$$x_e \in \{0, 1\} \qquad \forall e \in A' \tag{3.10}$$

donde:
d_i =Demanda del cliente i
$cap(k)$ = capacidad del vehículo k
$\delta^-(\{S\})$ ejes que parten de S y terminan fuera de S
$\delta^+(\{S\})$ ejes que parten fuera de S y terminan en S

La función objetivo 3.1 es la distancia total recorrida por la flota de vehículos. Es decir, es la suma de las distancias asociadas a las variables activas que interconectan clientes entre sí, a las que interconectan vehículos con clientes y a las que conectan clientes con el nodo depósito (d).

Las restricciones 3.2 son las ya mencionadas *restricciones de conservación de flujo*. Estipulan que la suma de todo el flujo que ingresa a cada nodo debe coincidir con la suma del flujo que egresa del mismo (exceptuando a los nodos depósito o y d). Estas restricciones son la base de un modelo de flujo en redes.
Se ha denominado *restricciones de control de capacidad máxima* a la familia 3.3. Estas desigualdades establecen que la suma de las demandas de todos los clientes servidos por un

mismo vehículo debe ser a lo sumo la capacidad máxima asignada a dicho vehículo. En estas restricciones se pide, asimismo, que los multiejes de los *nodos cliente* correspondientes al vehículo analizado, tomen el valor 0 si el vehículo no ha sido usado en la solución óptima del problema (recordar que a diferencia de los modelos de 2 índices presentados anteriormente, se eligió optimizar también el número de vehículos usados).
A las restricciones 3.4 y 3.5 se las ha denominado *restricciones de lógica de rutas*. Las restricciones 3.4 determinan que si 2 *nodos cliente* están conectados, entonces los multiejes que informan el vehículo que los atraviesa, deben coincidir en todos sus valores. Por otro lado, las restricciones 3.5 piden lo mismo para los *nodos cliente* y los *nodos vehículo*.

Las restricciones 3.6 son las conocidas *restricciones de eliminación de subtours*. Fueron introducidas en los problemas del viajante de comercio (TSP) y establecen que si se toma cualquier conjunto de *nodos cliente* con cardinalidad mayor o igual a 2 y menor o igual a $n-1$, entonces se debe cumplir que la suma de todas las variables que representan ejes con ambos extremos en el conjunto elegido, debe ser menor o igual al tamaño de dicho conjunto menos 1. Si esto no se cumple, en el conjunto elegido hay al menos un eje sobrante que forma un subtour.
Otra manera de escribir esta familia de restricciones es pedir que, dado un conjunto como el anterior, la suma de las variables que representan ejes con un extremo en el conjunto y otro fuera de él (corte), deba ser mayor o igual a 2. La cantidad de posibles conjuntos es $O(2^n)$ por lo que la utilización de dichas restricciones no se realiza de manera explícita, sino que se las utiliza como la familia de cortes principal a la hora de agregar cortes en el algoritmo de *branch&cut*.

A continuación, se pueden observar las restricciones 3.7. Estas desigualdades han sido bautizadas como *restricciones de rutas dirigidas*. La idea principal de las mismas es evitar permutaciones de soluciones factibles. Si, por ejemplo, se considera una solución O_1 y se modifica la dirección de una de las rutas de la misma, se obtendrá otra solución factible distinta O_2. Si esto se generaliza, se puede observar que para cada solución existen 2^K soluciones factibles, donde K es la cantidad de vehículos. Esto puede llevar a que se analice diferentes permutaciones de la misma solución a la hora de realizar la optimización del modelo. Para solucionar este problema, es que se utilizan las restricciones 3.7. El funcionamiento de las mismas es muy simple: solicita que si el nodo i es el primer nodo de la ruta k, entonces el índice del último nodo de la misma ruta debe ser siempre mayor a i. De esta manera, se exige que las rutas de la solución óptima sean consideradas en un sólo sentido.
Además de poder intercambiar la dirección de las rutas, pueden permutarse los números de vehículos que las transitan. Es decir, el modelo no distingue si una determinada ruta la realiza el vehículo 1 o el vehículo 3, por ejemplo. Esto ocasiona nuevamente el problema de tener múltiples soluciones equivalentes por cada solución considerada. La cantidad de posibles permutaciones de una solución es $K!$, donde K es la cantidad total de vehículos. Para solucionar este problema y eliminar la posibilidad de permutar vehículos, se intro-

duce la familia de restricciones 3.8 denominadas *restricciones de camiones ordenados*. En estas restricciones se exige que si un *nodo cliente* i comienza una ruta k, entonces todos los vehículos con índice mayor a k deben cursar rutas que inicien en clientes con índice mayor a i. Esta simple restricción elimina el problema anteriormente citado.
Por último, en 3.9 se pide que las variables x_{ii} (que representan a los ejes (q_{ib}, q_{ic})) valgan siempre 1 y en 3.10 se exige que todas las variables involucradas sean binarias.

3.1.3. Correctitud del modelo presentado

Proposición *Cualquier solución válida del modelo presentado con vehículos de capacidad homogénea puede ser convertida en una solución válida para el modelo de flujo de 2 índices.*

Demostración. Debido a la manera en que se construyó el nuevo modelo, cualquier solución válida deberá asignar rutas que comiencen en el nodo depósito o y terminen en el nodo depósito d. Por otra parte, dadas las *restricciones de conservación de flujo* y las *restricciones de subtour* se sabe que dichos caminos serán disjuntos y recorrerán clientes distintos. Satisfaciendo las *restricciones de control de capacidad máxima* se sabe también que la capacidad máxima de los vehículos utilizados no ha sido sobrepasada. Además, como se han utilizado variables ficticias para representar los ejes (q_{ib}, q_{ic}) con valor constante 1, se puede asegurar que todos los clientes son visitados por algún vehículo.
Debido a que la solución cumple todos estos requisitos, es posible fusionar los nodos depósito o y d con los *nodos vehículo* en un nuevo nodo 0, eliminar todos los ejes intermedios y dejar los ejes de salida de los *nodos vehículo* como ejes de salida de 0. Los ejes de entrada al nodo depósito d se convierten en ejes de entrada al nodo 0. Por último, se fusionan cada uno de los subnodos a, b y c de cada *nodo vehículo* en un sólo nodo, eliminando también aquí los ejes intermedios (multiejes entre a y b, y eje ficticio de b a c). Todos los ejes eliminados tienen asociado costo 0, por lo que la función objetivo a optimizar no varía.
Si se utilizan todos los ejes restantes como variables binarias para el modelo de flujo de 2 índices, se puede verificar que cualquiera sea el valor de dichas variables, la solución continuará siendo válida ya que se satisfacen de manera trivial todas las restricciones formuladas en el modelo de 2 índices. De esta manera, se demuestra que cualquier solución válida del modelo presentado también lo es para el modelo de flujo de 2 índices.

Observación: Es importante destacar que no es posible afirmar que cualquier solución válida del modelo de 2 índices es también válida para el modelo de 3 índices presentado ya que el nuevo modelo contiene más información que el modelo de 2 índices (número de vehículo que atraviesa el nodo). En otras palabras, los modelos no son equivalentes.

Proposición *Cualquier solución válida óptima del modelo presentado con vehículos de*

capacidad homogénea puede ser convertida en una solución óptima para el modelo de flujo de 2 índices.

Demostración. En base a la proposición anterior, se sabe que cualquier solución válida puede transformarse. Ahora, si se obtiene una solución óptima O_1 en el nuevo modelo de 3 índices, dicha solución se puede transformar en una solución válida O'_1 para el modelo de 2 índices. Puede suponerse que dicha solución es válida pero no es óptima. Esto quiere decir que puede encontrarse otra solución distinta al modelo de 2 índices O'_2 que es óptima y que mejora la función objetivo. Pero O'_2 puede ser utilizada en el nuevo modelo de 3 índices completando la información del número de vehículo que recorre cada cliente de manera conveniente. Es decir, puede completarse de manera arbitraria las variables correspondientes a los ejes que indican información de vehículos, cuidando de satisfacer todas las restricciones del modelo. Debido a que los ejes completados tienen costo 0 en la función objetivo, la nueva solución O'_2 es también mejor solución que la óptima O_1 para el nuevo modelo. Esto quiere decir que la solución O_1 no era óptima y se ha arribado a un absurdo. Por lo tanto, cualquier solución óptima para el nuevo modelo con vehículos de capacidad homogénea lo es también para el modelo de flujo de 2 índices.

Después de probar la validez de las soluciones encontradas por el nuevo modelo y, que la solución óptima también lo es para los modelos de flujo de 2 índices, se puede aseverar que el nuevo modelo de 3 índices presentado es correcto en relación con los problemas tratables por el modelo de flujo de 2 índices.

3.1.4. Análisis del modelo

En esta sección se analizará el número de variables y de restricciones del modelo presentado y se las comparará con las formulaciones de 3 índices presentadas en el capítulo 2.

La cantidad de variables binarias de 3 índices es la siguiente: K^2 para representar los ejes (m_{ka}, m_{kb}, l) y Kn correspondientes a los ejes (q_{ia}, q_{ib}, k) (K multiejes por *nodo cliente*).

A continuación, se enumera la cantidad de variables binarias de 2 índices: Hay K variables que interconectan el nodo depósito (d) con los *nodos vehículo*, n variables representando los ejes ficticios (q_{ib}, q_{ic}), Kn variables para los ejes unidireccionales (m_{kb}, q_{ia}), $n(n-1)$ variables representando los ejes que interconectan *nodos cliente* considerando 2 ejes (ambas direcciones) para cada par de nodos y n ejes dirigidos conectando los *nodos cliente* con el nodo depósito de llegada (d).
Entonces, el número total de variables utilizadas es $\mid A' \mid = K^2 + Kn + K + n + Kn + n(n-1) + n = n^2 + K^2 + 2(Kn) + K + n$.
La cantidad de vehículos es significativamente inferior a la cantidad de clientes en mayoría

de los problemas prácticos, por lo tanto, puede decirse que la cantidad de variables binarias del modelo presentado es $O(n^2 + Kn) \approx O(n^2)$.

Volviendo al modelo de flujo de 3 índices presentado en el capítulo 2, se puede observar que se utilizan $O(n^2K)$ variables de 3 índices para representar los K multiejes que interconectan a cada par de clientes. Además, dicho modelo utiliza $O(nK)$ variables binarias y_{ik} para representar al vehículo que sirve a cada cliente. Si bien dicho modelo contiene la información del vehículo que atraviesa cada uno de los tramos y el nuevo modelo presentado contiene sólo la información del vehículo que visita cada cliente, es importante remarcar que el modelo presentado ofrece una reducción importante en el número de variables utilizadas ($O(n^2)$ frente a $O(n^2K)$ del modelo de flujo de 3 índices clásico).
Si bien el rango de aplicación del nuevo modelo está limitado a problemas que puedan modelarse conociendo sólo la identificación de los vehículos que atraviesan cada nodo, la mayoría de las extensiones a CVRP conocidas pueden modelarse de esta manera. En la sección 3.1.6 se ejemplificá su aplicación a la extensión Ruteo de Vehículos con Capacidades con Ventanas de Tiempo (VRPTW)

Es importante destacar que el número de restricciones es considerablemente mayor en el modelo presentado que en el modelo clásico de flujo de 3 índices. Esto no es considerado un inconveniente importante ya que la mayoría de las familias de desigualdades presentadas se utilizan en el algoritmo de *branch&cut* como desigualdades válidas agregadas durante las distintas etapas del proceso de separación. Otra ventaja de este modelo, a pesar de su número mayor de restricciones, es haber conseguido algoritmos de separación con complejidad polinomial para las nuevas familias de restricciones.

3.1.5. Extensiones y posibles mejoras

Debido al número considerable de variables que contiene el nuevo modelo, hay algunas alternativas que pueden aplicarse para disminuir dicho número. Estas modificaciones pueden hacer que el modelo deje ser un modelo clásico de flujo, por lo tanto, las mismas no serán tenidas en cuenta a la hora de implementar el algoritmo de *branch&cut* ya que uno de los objetivos principales de esta tesis fue el de avocarse a modelos de flujo. Por otro lado, se confía que la *fijación* de las variables que contienen valores constantes, permita que el paquete de optimización lineal utilizado (CPLEX) optimice la formulación de manera inteligente para que su utilización no produzca un impacto considerable en los tiempos de ejecución.

En primer lugar, las variables correspondientes a los ejes (m_{ka}, m_{kb}, l) con $l \neq k$ han sido eliminadas de la formulación debido a que valen siempre 0. De esta manera, quedan sólo K variables que representan a los ejes (m_{ka}, m_{kb}, k) que pueden tomar valores $0-1$. En

el modelo, se contaba también con variables binarias que representaban a los ejes (o, m_{ka}). Estas variables, al eliminar las variables anteriormente explicadas, cumplen la misma función que las variables (m_{ka}, m_{kb}, k). Una posible modificación es eliminar alguna de las dos y disminuir, de esta manera, el número de variables en K unidades.

Otra modificación posible es eliminar las variables ficticias (q_{ib}, q_{ic}) que valen siempre 1 y agregar un nuevo conjunto de restricciones de la forma siguiente:

$\sum_{k \in K} x_{ii}^k = 1 \quad \forall i \in Q$

Estas restricciones piden que la suma de todos los multiejes de cada cliente totalice el valor 1, exigiendo de esta manera que todos los clientes sean visitados por algún vehículo. Para poder efectuar esta modificación, los ejes (q_{ic}, q_{ja}) deben ser reemplazados por ejes de la forma (q_{ib}, q_{ja}). Con este cambio, se toma distancia de los modelos clásicos de flujo. El ahorro es de n variables con el agregado de n nuevas restricciones.

Si se considera la modificación del problema, donde se permite sólo un número fijo de vehículos K, es decir que la optimización no elige la cantidad de dichos vehículos a utilizar, se puede realizar otra modificación que permita reducir el número de variables en K unidades. La misma consiste en eliminar los *nodos vehículo* completamente del modelo, y conectar los ejes clientes directamente al nodo depósito o con multiejes, representando el número de vehículo que sale para cada cliente. Si se supone constante el número de vehículos, también sería factible relacionar el nodo depósito o con los *nodos cliente* mediante un sólo eje. Para realizar esta modificación, es necesario también modicar las restricciones 3.3 y eliminar las restricciones 3.5. Además, con este cambio se pierde la posibilidad de utilizar las restricciones de eliminación de soluciones equivalentes 3.8 denominadas *restricciones de camiones ordenados.*

3.1.6. Adaptación del nuevo modelo a Ruteo de Vehículos con Ventanas de Tiempo (VRPTW)

Se presenta en esta sección un ejemplo de cómo extender el modelo presentado a variantes que requieran el agregado de determinadas restricciones adicionales. Se pretende mostrar la flexibilidad y su adaptabilidad a diferentes escenarios.

Como se describe en (TV01), el VRPTW es generalmente formulado mediante *modelos de flujo de Commodities.* En dicho libro se modifica la formulación presentada en la sección 2.2.2 para incorporar las restricciones de tiempo.
Para mostrar como se puede modificar el nuevo modelo para que soporte ventanas de tiempo, se utilizará la misma notación utilizada en este libro. Se define w_{ik}, $i \in Q, k \in K$ como

las variables de tiempo asociadas a cada uno de los *nodos cliente* i cuando son ser servidos por el vehículo k. Esta variable representa la hora de llegada al *nodo cliente* i. Se define también los tiempos t_{ij} asociados a la duración en tiempo que requiere realizar el viaje entre los nodos i y j. Por su parte, s_i define el tiempo que demandará servir al cliente i. Se definen además los intervalos $[a_i, b_i]$ indicando el tiempo mínimo y máximo durante el cual se puede arribar al nodo i. Se define el intervalo $[E, L]$ como el tiempo mínimo y máximo de salida del depósito de los vehículos. Por lo tanto, existirán soluciones factibles, si y solo sí $E \leq min_{i \in Q}\ b_i - t_{oi}$ y $L \leq min_{i \in Q}\ a_i + s_i + t_{io}$.

A la formulación 3.1.2 se le debe agregar las siguientes restricciones:

$$w_{ik} + s_i + t_{ij} - w_{jk} \leq (2 - x_{ij} - x_{ji} - x_{ii}^k)M_{ij} \qquad \forall i, j \in Q, k \in K \qquad (3.11)$$

$$a_i x_{ii}^k \leq w_{ik} \leq b_i x_{ii}^k \qquad \forall i \in Q, k \in K \qquad (3.12)$$

$$0 \leq w_{ik} \leq L \qquad \forall i \in Q, k \in K \qquad (3.13)$$

Las restricciones 3.11 señalan que la hora de llegada al nodo i, más el tiempo que insume su servicio, más el tiempo que llevará transitar el tramo (i, j) para llegar hasta el cliente j, debe ser a lo sumo el tiempo de arribo al cliente j. Todo esto debe satisfacerse sólo en caso de que la variable que indica la conexión entre ambos nodos esté activada y el número de vehículo que pasa por i sea el correcto. En caso contrario, se utilizará M_{ij} como una constante arbitraria muy grande asociada al tramo para asegurar que lo pedido anteriormente deje de exigirse.
En las restricciones 3.12 se establece que la hora de llegada a cada *nodo cliente* debe encontrarse dentro de la ventana de tiempo asociada a dicho cliente. Por último, las restricciones 3.13 determinan que las horas de llegada a cada cliente deben ser no negativas y deben encontrarse dentro del rango máximo estipulado por la constante L. En lugar de la no negatividad, en el libro anteriormente citado se establecía que $E \leq w_{ik} \leq L$, lo cual se considera un error ya que los vehículos que no arriban a un determinado cliente tendrán su variable w asociada en 0, y pedir que sea mayor o igual que E es incorrecto a menos que $E = 0$.

Estos cambios respetan las convenciones adoptadas por las formulaciones más frecuentemente utilizadas.

De la misma manera, puede modificarse el nuevo modelo para un gran número de otras situaciones.

Capítulo 4

El algoritmo de *branch&cut*

Con el propósito de poner a prueba el modelo planteado, se desarrolló un algoritmo de *branch&cut* que incluye todas las características básicas que un algoritmo de esta clase debe contener para ser eficiente.

La base fundamental del éxito de estos algoritmos es el cálculo simultáneo de cotas superiores e inferiores a la solución óptima. La velocidad de convergencia estará íntimamente relacionada con la rapidez en que dichas cotas se acerquen al óptimo buscado. Las cotas superiores se obtienen mediante algoritmos heurísticos y mediante la enumeración proporcionada por el proceso de branching. Por otra parte, las cotas inferiores se ajustan mediante la generación repetida de planos de corte.
Las desigualdes válidas constituyen la herramienta fundamental del algoritmo de planos de corte. Por ello, vislumbrar familias de desigualdades válidas que incrementen significativamente la función objetivo de la relajación lineal activa, se convierte en el objetivo número uno a la hora de diseñar un buen algoritmo de *branch&cut*. Para encontrar y analizar familias de desigualdades es muy importante y conveniente realizar un buen estudio poliedral del problema en cuestión. Conociendo el poliedro asociado se podrá saber si las desigualdades válidas encontradas poseen buenas cualidades teóricas, como así también, saber si las mismas son o no facetas del poliedro. Vislumbrar desigualdades válidas que, a su vez, prueben ser facetas del poliedro se convierte en el anhelo máximo que puede esperarse a la hora de intentar resolver en forma exacta problemas combinatorios mediante algoritmos de *branch&cut*.

En este trabajo, no se realizará un estudio poliedral del problema de ruteo de vehículos con capacidades. Sin embargo es importante mencionar que el poliedro asociado al problema de ruteo de vehículos con capacidades no tiene una caracterización sencilla y se lo suele estudiar analizando la cápsula convexa de una modificación del problema. En la bibliografía, hay estudios realizados de la cápsula convexa del *Ruteo Gráfico de Vehículos con capacidades* (GCVRP). Este poliedro tiene mejores propiedades que el poliedro asociado al

problema original y ha permitido grandes avances en el estudio de la cápsula convexa de CVRP. La metodología es similar a la efectuada para el problema del viajante de comercio (TSP). Incluso, muchos de los resultados en el problema TSP fueron generalizados para el problema CVRP. En (TV01) se prueba que cualquier desigualdad válida para TSP es también válida para CVRP.
Puede encontrarse una excelente introducción a las técnicas fundamentales del análisis poliedral en (NW88). Estudios poliedrales del problema CVRP pueden encontrarse en (LN84; AHM90; Ara90; CCM91; CH93; Aug95; ACH98; LEL02).

A continuación, se describirán las desigualdades utilizadas por el algoritmo de *branch&cut* implementado y sus rutinas de separación asociadas. Por último, se describirán los distintos elementos y detalles que componen el algoritmo de *branch&cut*.

4.1. Desigualdades válidas utilizadas

Las nuevas desigualdades que fueron introducidas para este modelo, y que son utilizadas como planos de corte, poseen un número polinomial de elementos y su correspondiente algoritmo de separación consiste en buscar por enumeración las desigualdades violadas. Estos algoritmos han demostrado ser muy eficientes en el desenvolvimiento global del algoritmo principal. Por otra parte, se utilizan también varias familias de desigualdades válidas propuestas en la literatura que, contienen por lo general, un número exponencial de elementos. Para dichos casos se ha utilizado el paquete de rutinas CVRPSEP (Lys05) que contiene algoritmos de separación muy eficientes explicados en los trabajos (LEL02; LLE04). Este paquete de rutinas es una implementación de código abierto de las desigualdades válidas propuestas en distintos algoritmos de *branch&cut* para el CVRP encontrados en la bibliografía. Los autores, son además los creadores de los últimos algoritmos publicados para dicho problema dónde hacen uso de todas las desigualdades implementadas en CVRPSEP.

4.1.1. Desigualdades de Capacidad

Después de haber efectuado una vasta experimentación implementando algoritmos polinomiales de separación, para las desigualdades presentadas en el modelo como *restricciones de eliminación de subtours*, se llegó a la conclusión de que las mismas no incrementaban de manera significativa la cota inferior de la relajación. El problema quedó resaltado al comparar los incrementos obtenidos con los resultados publicados en los trabajos de *branch&cut* sobre problemas de ruteo de vehículos con capacidades (ABB$^+$95; BH02; ACH03; LLE04). Si bien estos trabajos se basan principalmente en la formulación clásica de 2 índices, las cotas inferiores obtenidas en la raíz del árbol superaban ampliamente a las mismas cotas obtenidas con el modelo presentado junto con las *desigualdades de eliminación de subtours*. A partir del análisis de la situación se observó que el optimizador lineal tendía a completar

los multiejes correspondientes a los *nodos cliente* y a los *nodos vehículo* con valores fraccionales entre 0 y 1. De esta manera, queda asignado un nodo a varios vehículos en forma partida. Como ello no es posible, y además, no se consiguió encontrar familias de desigualdades que eliminaran de manera efectiva dichos valores fraccionales, se optó por reemplazar las *restricciones de eliminación de subtours* por las conocidas *desigualdades de capacidad*. Estas desigualdades fueron introducidas con la formulación clásica de flujo de 2 índices 2.6. Al igual que las *desigualdades de eliminación de subtours*, las *desigualdades de capacidad* poseen un número exponencial de elementos. Sin embargo, para esta nueva familia no se ha encontrado algoritmos polinomiales de separación, y se utilizan en su reemplazo diferentes heurísticas que realizan la tarea de manera eficiente.

Existen numerosas maneras de expresar las desigualdades de capacidad. En general, se prefieren las desigualdades que generen sistemas con un número mayor de ceros ya que dicha particularidad permite acelerar la reoptimización del sistema lineal. Dadas las *restricciones de conservación de flujo*, las siguientes tres formas de cortes de capacidad son equivalentes (Lys05):

1. $X(S : S) \leq \mid S \mid -k(S)$
2. $X(\delta(S)) \leq 2k(S)$
3. $X(\bar{S} : \bar{S}) + \frac{1}{2}X(\{0\} : \bar{S}) - \frac{1}{2}X(\{0\} : S) \leq \mid \bar{S} \mid -k(S)$

En las formas mostradas se utilizó la notación de corte explicada anteriormente. Se agregó una nueva notación $X(A : B)$ que denota la suma de los ejes del corte entre los conjuntos A y B y. Por último, $\bar{S}$ representa el complemento de S, es decir, a todos los clientes que se encuentran fuera del conjunto S.

Según el análisis realizado en diferentes trabajos y específicamente en (Lys05), conviene utilizar la forma 1 cuando $\mid S \mid \leq n/2$, y la forma 3 en caso contrario. Para el modelo presentado, se ha optado por utilizar la forma que genere menos variables. Para ello se ha incluido también la forma 2 modificada debido a que el modelo es dirigido y permite utilizar la mitad de los ejes que los mostrados. Esa particularidad no se cumple ni para la forma 1 ni para la forma 3 ya que, al utilizar los ejes internos que interconectan los conjuntos, es necesario utilizar los ejes en ambas direcciones. La forma utilizada es la siguiente:

2'. $X(\delta^{+}(S)) \leq k(S)$

En base al conjunto de nodos que determine la *desigualdad de capacidad* se calcula la cantidad de ejes que requerirá representarlo y se elige de las formas 1, 2' o 3 la que determine la menor cantidad de variables.

El algoritmo de separación finalmente utilizado es el provisto por el paquete CVRPSEP. Inicialmente, antes del conocimiento de dicho paquete, se implementó una heurística golosa de separación inspirada en las ideas de los trabajos (ABB+95; ABB+98; LLE04; CH93). Este algoritmo demostró resultados similares a los publicados en los trabajos mencionados anteriormente. Sin embargo, al probar el paquete CVRPSEP se observó que este obtenía mejores resultados en tiempos no muy superiores (Cuadro 4.1). Esta fue la razón por la cual se culminó implementando el algoritmo principal con la rutina de separación de capacidades provista por CVRPSEP. Debido a que las *restricciones de capacidad* son las que más incrementan la cota inferior del algoritmo principal de *branch&cut*, se prefirió utilizar las rutinas de separación que mejores resultados reportaron.

La idea básica del algoritmo goloso que se había implementado originalmente, y que coincide con los lineamientos principales con los que se implementó en CVRPSEP es la siguiente:

1. Con el grafo de entrada $G_c^* = (V_c, E_c^*)$ donde se consideran solo los ejes $E^* = \{e \in E : x_e^* > 0\}$ y los nodos cliente sin el depósito, calcular las componentes conexas $S_1, ..., S_p$ de G_c^*. Para cada S_i se verifica las desigualdades de capacidad como también para $V_c \backslash S_i$. Por último se verifica la violación de las desigualdades de capacidad para la unión de componentes que no están conectadas al depósito. Si se encuentra alguna violación se sale devolviendo él o los conjuntos que la producen.

2. Se realiza un proceso que se denomina *proceso de Shrinking* y se obtiene un nuevo grafo G_s. La idea de dicho proceso consiste en fusionar conjuntos de nodos en supernodos cuando el valor del eje que conecta dos nodos cumple $x_e^* \geq 1$ y ambos nodos no son incidentes al depósito. Se ha demostrado que encontrar desigualdades de capacidad en el grafo G_s equivale a encontrarlas en el grafo G^* original.

3. Se itera un número de veces definido por la constante *maxiter* (fijada por experimentación en función del número de nodos del grafo G_s):

   a) Se itera para cada uno de los nodos i de G_s:

      1) Se arma un conjunto inicial S con el nodo i más un conjunto aleatorio de nodos.
      2) Se verifica si el conjunto S armado no fue creado anteriormente. Se utiliza para ello una tabla de Hash que memoriza los conjuntos analizados.
      3) Se verifica si el conjunto S genera una restricción de capacidad violada. Si es así se la agrega a la lista de nuevas restricciones.
      4) Mientras haya algún nodo de G_s que no esté en el conjunto S, se selecciona un nodo v tal que $X(S : \{v\})$ sea máximo y repetir:

   a' Verificar si el nuevo S no fue creado anteriormente usando la tabla de Hash. Si es así descartarlo y volver a 3a4.

   b' Si el nuevo conjunto S genera una restricción de capacidad violada, agregarla a la lista de nuevas restricciones.

4. Devolver el conjunto de nuevas restricciones violadas (puede ser un conjunto vacío).

Observaciones: según lo indicado en el trabajo (LLE04) que marcó las bases para CVRPSEP, hay ciertas diferencias y agregados a la implementación del algoritmo mostrado. En primer lugar, se generalizó la idea del proceso de *Shrinking* a conjuntos $\mid S \mid > 2$ y se demostró que en tales casos el proceso seguía siendo seguro (mantenía las mismas desigualdades de capacidad violadas que el grafo original). En segundo lugar, se utilizó un algoritmo de flujo máximo para encontrar desigualdades de capacidad violadas mediante la búsqueda de las conocidas *desigualdades fraccionales de capacidad*. Por último, en dicho trabajo se finalizó el proceso utilizando una heurística que revisa todas las desigualdades encontradas en ejecuciones anteriores del proceso de separación. En el proceso, reemplaza el conjunto S respectivo por algún supernodo GS_s que contenga a S e intenta agregar o eliminar otros supernodos en búsqueda de nuevas desigualdades violadas. De alguna manera, este último algoritmo generaliza la heurística de búsqueda Tabú explicada en el trabajo (ABB+98).
Todos estos agregados determinan las diferencias observadas en el Cuadro 4.1. Requieren un mayor tiempo de ejecución que el algoritmo implementado, pero dicho costo se premia con mejores resultados.

4.1.2. Desigualdades de Lógica de Ruta 1

En la formulación se presentaron 2 tipos de restricciones de lógica de ruta. Las *restricciones de lógica de ruta* 3.4 establecen que si dos *nodos cliente* están conectados en la solución final, entonces los valores asignados a cada uno de los multiejes correspondientes debe coincidir. Por otra parte, las *restricciones de lógica de ruta* 3.5 establecen lo mismo pero entre los *nodos depósito* y los *nodos cliente*. A continuación, se vuelven a mostrar ambos conjuntos de desigualdades:

Desigualdades de lógica de ruta 1: $x^k_{ii} - x^k_{jj} + x_{ij} + x_{ji} \leq 1 \quad \forall i, j \in Q \; i \neq j; k = 1...K$

Desigualdades de lógica de ruta 2: $x^k_{oo} + x^k_{ii} - 2x^k_{oi} \geq 0 \quad \forall i \in Q; k = 1...K$

Debido a que existe una cantidad de Kn *desigualdades de lógica de ruta 2*, se ha decidido incorporarlas en la formulación inicial. Por otra parte, como la cantidad de *restricciones de lógica de ruta 1* es $Kn(n-1)$, es decir, una cantidad de $O(n^2)$ restricciones, se las agregó durante el proceso de separación a medida que las mismas eran necesitadas. Luego

Instancia	Imple. Propia		con CVRPSEP		% dif.
	LB	Tiempo	LB	Tiempo	
E-n22-k4	375	0,13	375	0,24	0,00 %
E-n23-k3	569	0,9	569	0,11	0,00 %
E-n30-k3	508,5	0,25	508,5	0,23	0,00 %
E-n33-k4	833	0,47	**833,5**	0,87	0,06 %
E-n51-k5	513,188	0,99	**514,524**	2,38	0,26 %
E-n76-k7	656,763	7,56	**661,349**	16,9	0,69 %
E-n76-k8	705,165	9,91	**711,207**	26,43	0,85 %
E-n76-k10	782,006	18,82	**789,451**	48,06	0,94 %
E-n76-k14	940,634	51,55	**948,064**	75,92	0,78 %
E-n101-k8	789,802	9,81	**796,423**	38,4	0,83 %
E-n101-k14	998,134	59,83	**1008,19**	165,83	1,00 %
M-n101-k10	818,5	27,43	**819,5**	75,33	0,12 %
F-n45-k4	724	0,6	724	1,04	0,00 %
F-n72-k4	232,4	2,33	**232,5**	1,9	0,04 %
F-n135-k7	1153,81	62,76	**1158,22**	133,21	0,38 %

Cuadro 4.1: Rutina de separación de capacidades implentada vs. rutina de CVRPSEP

de un proceso de experimentación en el que se probó incluirlas en la formulación inicial frente a utilizarlas como cortes, se llegó a la conclusión que esta última alternativa ofrecía resultados similares con tiempos de ejecución muy inferiores, ya que optimizar el sistema lineal con tantas restricciones aletargaba mucho el proceso. La misma experimentación fue efectuada con las *desigualdades de lógica de ruta 2* pero en este caso se llegó a la conclusión contraria debido al número reducido de las mismas.
El proceso de separación utilizado es un proceso polinomial de separación exhaustivo. Se recorren todas las posibles desigualdades basándose en el resultado del último sistema lineal optimizado, chequeando cuando existen desigualdades violadas. Este proceso tiene un tiempo teórico de ejecución de orden $O(Kn^2)$ y en las pruebas demostró ser extremadamente rápido.

4.1.3. Desigualdades de Rutas Dirigidas

Como se explicó en el capítulo 3, el modelo presentado muestra la particularidad de permitir múltiples soluciones equivalentes. Para poder solucionar este problema se presentaron 2 familias de desigualdades de eliminación de simetría: *desigualdades de rutas dirigidas* y *desigualdades de camiones ordenados*. La primera 3.7 tiene el objetivo de eliminar las 2^K soluciones equivalentes por cada solución factible del modelo. Como se vió, tal situación se origina al invertir la dirección de cada una de las rutas de la solución. A continuación, se

presenta nuevamente la familia referida:

$$x_{oi}^k + x_{jd} + x_{jj}^k \leq 2 \quad \forall i, j \in Q \; i > j; k = 1...K$$

Como puede observarse, existen $(n-1)(n)K/2$ posibles restricciones, es decir que la cantidad de desigualdades de este tipo es del orden de $O(n^2K)$. Se decidió agregarlas en el algoritmo de separación a medida que las mismas iban siendo requeridas para no sobrecargar la formulación inicial.
El algoritmo de separación consiste en revisar cada una de las rutas verificando si el índice del primer nodo cliente es menor que el índice del último nodo cliente. Este algoritmo de exploración exhaustiva es polinomial y tiene un costo teórico de ejecución del orden de $O(Kn^2)$. Al igual que el algoritmo de separación de las *restricciones de lógica de ruta 1*, este algoritmo resultó ser muy eficiente en la práctica.

4.1.4. Desigualdades de Camiones Ordenados

La segunda familia de desigualdades válidas presentadas para eliminar simetría es la denominada familia de *restricciones de camiones ordenados* 3.8. Estas desigualdades eliminan la posibilidad de permutar los números de los vehículos que recorren cada una de las rutas. Como se explicó al presentar el modelo, tal irregularidad permitía $K!$ soluciones equivalentes por cada solución factible analizada. Las *restricciones de camiones ordenados* son las siguientes:

$$x_{oi}^k + x_{oj}^{k+1} \leq 1 \quad \forall i, j \in Q, i > j; k = 1...K-1$$

Al igual que las *restricciones de rutas dirigidas*, esta familia de restricciones tiene $O(n^2K)$ posibles desigualdades. Como es de esperar, estas restricciones también han sido utilizadas como cortes agregados en el procedimiento de separación. El algoritmo de separación respectivo también realiza búsqueda exhaustiva de desigualdades violadas. Recorre cada uno de los ejes que interconectan *nodos vehículo* con *nodos cliente* y verifica que todos los *nodos cliente* con índice menor al analizado, no sean iniciadores de la ruta siguiente. De esta manera, se recorren todas las variables necesarias del último sistema optimizado y se verifica que no se viole la desigualdad. El costo computacional de dicho proceso es del orden de $O(Kn^2)$ y también ha mostrado resultados empíricos muy satisfactorios.

4.1.5. Desigualdades Framed Capacity (FCI)

Las desigualdades de capacidad enmarcada (en inglés, Framed Capacity Inequalities) han sido propuestas por Pochet (Poc) y Augerat (Aug95). La definición es la siguiente: Para

algún $S \subseteq V_c$, se toma $\Omega = \{S_1, ..., S_p\}$ como una partición de S. Se define $k(S, \Omega)$ como el mínimo número de vehículos necesarios para servir S sabiendo que las *desigualdades de capacidad* se cumplen por igualdad para cada uno de los S_i. La desigualdad de capacidad enmarcada (FCI) es:

$$X(\delta(S)) + \sum_{i=1}^{p} X(\delta(S_i)) \geq 2k(S, \Omega) + 2\sum_{i=1}^{p} k(S_i)$$

Una explicación más detallada del significado de dicha desigualdad, como así también la idea intuitiva que le da sentido puede encontrarse en (TV01; LLE04).
El algoritmo de separación utilizado es el proporcionado por el paquete CVRPSEP. Según se detalla en (LLE04), dicha implementación calcula el grafo G_c^* de componentes conexas y toma cada una de las componentes como candidatos para representar a S. En base a cada uno de dichos conjuntos, se genera un árbol basado en las particiones Ω y se realiza una búsqueda DFS (depth first search - búsqueda en profundidad primero) para encontrar restricciones FCI violadas. Para el cálculo de la función k se resuelve BPP utilizando técnicas propuestas en (MT90). En el capítulo de resultados se mostrará los testeos efectuados con estas desigualdades y su comparación con trabajos anteriores.
De acuerdo a lo analizado en (Lys05), existen numerosas formas de expresar las desigualdades FCI. Sin embargo, en dicho trabajo se llega a la conclusión de que la representación más conveniente por ser la más generadora de 0's (genera una matriz esparsa) es la siguiente:

$$-\tfrac{1}{2}X(\delta(S)) + \sum_{i=1}^{p} X(S_i : S_i) \leq -r(S, \Omega) + \sum_{i=1}^{p}(\mid S_i \mid -k(S_i))$$

Esta forma es la implementada en el algoritmo de *branch&cut*.

Si bien no se reporta la complejidad del algoritmo en los trabajos referidos, la experimentación realizada demuestra que el algoritmo de separación es bastante costoso e incide fuertemente en el costo total del tiempo utilizado por los algoritmos de separación.

4.1.6. Desigualdades Strengthened Comb

Las desigualdades de peine, comúnmente referidas en la literatura en inglés como *comb inequalities*, fueron propuestasa para el problema del viajante de comercio (TSP) (GP79a; GP79b). Muchos autores intentaron su adaptación a CVRP, como puede verse en (LN84; Ara90; ABB+95; BH02). En el presente algoritmo, debido a que se han utilizado las rutinas de separación provistas por el paquete CVRPSEP, se adoptará el enfoque efectuado en (Ara90), tal como lo indica el trabajo (LLE04) a partir del cual se origina el dicho paquete. Las desigualdades que se utilizan son las denominadas *desigualdades Strengthened Comb*.

Se define un *comb* como un conjunto de vértices $H \subset V_c$ llamado *handle*, y $t \geq 2$ conjuntos de vértices denominados *teeth*, de manera que:

- $H \cap T_j \neq \emptyset$ y $T_j \setminus H \neq \emptyset$ para $j = 1, ..., t$.
- para cada par $\{i, j\} \subset \{1, ..., t\}$, o $T_i \cup T_j \subset H$ o $T_i \cup T_j \cup H = \emptyset$ (o ambos).

Esta definición es más general que la que se utiliza en TSP.
Ahora, para cualquier conjunto $S \subset V$, se toma $\tilde{k}(S)$ igual a $k(S)$ si $0 \notin S$, y $k(V \setminus S)$ en caso contrario. Se define la cantidad:

$$S(H, T_1, ..., T_t) = \sum_{j=1}^{t} (\tilde{k}(T_j \cup H) + \tilde{k}(T_j \setminus H) + \tilde{k}(T_j))$$

Si $S(H, T_1, ..., T_t)$ es impar, entonces la siguiente desigualdad *strengthened comb* es válida para CVRP:

$$X(\delta(H)) + \sum_{j=1}^{t} X(\delta(T_j)) \geq S(H, T_1, ..., T_t) + 1$$

Para más detalle de la formulación y de la heurística de separación referirse a (LLE04). En el capítulo de resultados se mostrará los experimentos realizados con esta familia de desigualdades y la comparación de los resultados obtenidos, frente a los obtenidos en trabajos anteriores. Es importante destacar, cómo también fué mostrado en (LLE04), que el incremento de la cota inferior obtenido al utilizar estas desigualdades es considerable y que los tiempos consumidos por el algoritmo de separación no son elevados.

La forma que se utiliza en la implementación es la mostrada anteriormente.

4.1.7. Desigualdades MultiStar y Partial MultiStar

Las *desigualdades multiestrella* (del inglés Multistar) y *multiestrella parciales* fueron analizadas inicialmente en un trabajo no publicado por (AHM90) y luego extendidas en (LEL02). Este último trabajo presenta un procedimiento para mostrar la validez de las desigualdades Multistar y Partial Multistar, como también ideas para desarrollar un algoritmo de separación que las identifique. Además, se generalizan las desigualdades anteriores en las llamadas *homegeneous* multistar y partial multistar. En (LLE04) se utilizan estas desigualdades en un algoritmo de *branch&cut* y se consiguen muy buenos resultados con las mismas.
Las desigualdades (homogeneous) multistar tienen la siguiente forma:

$$\alpha X(E(N)) + \beta X(E(N : S)) \leq \gamma$$

El conjunto de nodos $N \subset V_c$ es denominado *núcleo*, $S \subseteq V_c \setminus N$ es el conjunto de los elementos denominados *satélites*, y α, β y γ son constantes que dependen de $\mid N \mid$ y $\mid S \mid$. Las desigualdades (homogeneous) partial multistar tienen la siguiente forma:

$$\alpha X(E(N)) + \beta X(E(C : S)) \leq \gamma$$

En estas desigualdades se introduce un nuevo conjunto de vértices $C \subset N$ denominado *conectores*. De la misma manera que en la familia mostrada anteriormente, α, β y γ son constantes que en este caso dependerán de $\mid N \mid$, $\mid S \mid$ y $\mid C \mid$.

Los algoritmos de separación presentados en (LEL02) son un conjunto de heurísticas golosas y métodos basados en ideas de flujo máximo que permiten seleccionar el conjunto núcleo N. Otra batería de heurísticas golosas están destinadas a encontrar los conjuntos de satélites S y los conectores C para el caso de las desigualdades partial multistar. La separación de estas desigualdades se realiza utilizando el paquete CVRPSEP. Según lo indicado en (Lys05), las desigualdades multistar pueden representarse de 3 formas equivalentes:

- $BX(\alpha(N)) - AX(C : T) \geq L$
- $2BX(N : N) + AX(C : T) \leq 2B \mid N \mid -L$
- $2BX(\tilde{N} : \tilde{N}) + BX(\{0\} : \tilde{N}) - BX(\{0\} : N) + AX(C : T) \leq 2B \mid \tilde{N} \mid -L$

Se ha decidido implementar cualquiera de las 2 primeras formas dependiendo de la cantidad de variables que cada una genere. En el proceso de separación se calcula la cantidad necesaria de variables que requerirán ambas presentaciones y en base a ello, se elige cuál usar (siempre se prefieren las representaciones que contienen menor cantidad de variables).

4.1.8. Desigualdades Generalized Large MultiStar (GLM)

Otro conjunto de desigualdades similares a las anteriores son las llamadas *desigualdades Generalized Large MultiStar* propuestas por Gouveia (Gou94) y analizadas en (ACH98; LEL02). En dichos trabajos, también se prueba que las desigualdades generalized large multistar dominan las desigualdades de capacidad fraccionales y pueden ser separadas en tiempo polinomial. Esta familia de desigualdades toma la siguiente forma:

$$QX(E(N)) + \sum_{j \in V_c \setminus N} d_j X(E(N : \{j\})) \leq Q \mid N \mid -d(N)$$

Donde Q es la capacidad homogénea de los vehículos, d_i es la demanda del cliente i y N especifica el conjunto de nodos que conforman el *núcleo*.

Sin embargo, al igual que el resto de las desigualdades presentadas, esta desigualdad puede reescribirse de manera de generar la menor cantidad de variables. La forma que se implementa en el algoritmo de *branch&cut* es la siguiente:

$$QX(N:N) + \sum_{j \in \tilde{N}} d_j X(N:\{j\}) \leq Q \mid N \mid - \sum_{i \in N} d_i$$

Donde $\tilde{N}$ es el conjunto de clientes fuera del conjunto N.

La rutina de separación utilizada para identificar estas desigualdades es la provista por el paquete CVRPSEP. A diferencia del resto de las rutinas de separación de CVRPSEP, la separación de las desigualdades GLM es polinomial y exacta (si existe una desigualdad GLM violada, el algoritmo la detecta).

4.1.9. Desigualdades Hypotour

Las *desigualdades Hypotour* fueron propuestas para CVRP en (Aug95). Diferentes variaciones de las mismas fueron presentadas en (BMMN95) incluyendo las llamadas *desigualdades extended hypotour*. Otras variaciones han sido propuestas en (ABB+95). Las desigualdades de hypotour se basan en la simple idea de que: si se tiene un conjunto $F \subset E$ tal que cualquier solución factible de CVRP debe usar al menos un eje (variable) de F, entonces la desigualdad de hypotour $X(F) \geq 1$ debe satisfacerse.

Las desigualdades de hypotour que fueron utilizadas en el algoritmo de *branch&cut* implementado, están basadas en el trabajo (LLE04). El algoritmo de separación correspondiente, provisto por el paquete CVRPSEP, sólo considera una de las variantes denominada *desigualdades 2-edges extended hypotour* (2EH). La idea básica de tales restricciones es que dado el conjunto $W \subset V_c$ y dos ejes distintos $e_1, e_2 \in \delta(W)$, las desigualdades 2EH son:

$$X(\delta(W)) + 2X(F) \geq 2x_{e_1} + 2x_{e_2}$$

donde se cumple que $F \subset E$ y $e_i = \{u_i, v_i\}$ para $i = 1, 2$ con $v_1, v_2 \notin W$. En adelante, v_1 y v_2 serán llamados los *terminales* de 2EH.

La desigualdad 2EH expresa que para cualquier solución factible de CVRP que cumpla $X(\delta(W)) = 2$ y $x_{e_1} = x_{e_2} = 1$, debe usarse al menos un eje de F. Es decir, cualquier ruta factible que visite en forma seguida todos los clientes de $W \cup \{v_1, v_2\}$, comenzando y terminando en los *terminales*, debe usar al menos un eje de F.

La identificación de F se basa en que el camino hamiltoniano que pasa por $W \cup \{v_1, v_2\}$ debe extenderse por dos caminos al depósito, de manera que esos caminos sean disjuntos, nodo a nodo, exceptuando la intesercción de ambos en el depósito y que la demanda total determinada por el circuito, no exceda la capacidad del vehículo.

El procedimiento de separación implementado por CVRPSEP se explica en detalle en (LLE04). Según se indica en dicho trabajo, el procedimiento obtenido es polinomial y está acotado por una complejidad de $O(n^2)$.

4.1.10. Estrategia de separación

Para poder implementar un algoritmo de *branch&cut* eficiente, es importante encontrar buenos algoritmos de separación para las desigualdades válidas utilizadas como cortes. Es también una tarea fundamental decidir la política con la que los mismos serán ejecutados. Entre otras cosas, el orden en que se llama a cada uno de los algoritmos de separación es un factor muy decisivo a la hora de lograr buenos rendimientos. Para determinar cómo distribuir los tiempos de procesamiento entre los distintos algoritmos, se recurrió a la confrontación de diferentes estrategias utilizando para ello la experimentación sobre un conjunto representativo de instancias. La estrategia que mejores resultados proporcionó coincide en líneas generales con la implementada en (LLE04).
La estrategia finalmente implementada aplica un tratamiento para el nodo raíz del árbol y otro diferente para el resto de los nodos del mismo.

En el nodo raíz se llama, en primer lugar, a la rutina de separación de las *desigualdades de Capacidad.* Si se encuentra al menos una desigualdad violada por más de 0,2, se sale de la separación y se reoptimiza el sistema lineal (LP). En caso de que no se haya encontrado ninguna *desigualdad de Capacidad* violada, se deshabilita temporalmente la separación de capacidades y se llama a la rutina de separación de *desigualdades Framed Capacity (FCI).* Al llamar a dicha rutina se establece como máximo 20000 nodos a explorar en el árbol de búsqueda que es generado por el algoritmo de separación. Si se encuentra alguna restricción FCI violada, se reoptimiza el LP.

En caso de no encontrar restricciones FCI violadas, se continúa con el resto de las rutinas de separación. Utilizando la idea de (LLE04), se llama a las rutinas de separación de las clases de desigualdades *MultiStar*, *Comb* e *Hypotour* en un orden cíclico, cambiado en forma dinámica. Es decir, la primera vez se llamará a las rutinas de separación en el orden 1:MultiStar, 2:Comb y 3:Hypotour. La segunda vez, el orden cicla y se llama a las rutinas en el orden 1:Comb, 2:Hypotour y 3:MultiStar. La tercera vez, será 1:Hypotour, 2:MultiStar y 3:Comb, y ya, la cuarta vez, se volverá a la configuración inicial 1:Multistar, 2:Comb y 3:Hypotour. Si alguna de estas clases de desigualdades no encuentra al menos

una desigualdad con violación superior a un cierto límite (0,05 para MultiStar, 0,1 para Comb y 0,1 para las desigualdades de Hypotour), se procede a efectuar la separación de la clase siguiente. Si ese límite sí es superado, se reoptimiza el LP inmediatamente y se vuelve a comenzar. Permanentemente se verifica que la cantidad de restricciones generadas no supere las 500 unidades. Si tal es el caso, también se reoptimiza el LP inmediatamente. Al referirse a la clase MultiStar se denota tanto las rutinas de separación de desigualdades *Homogeneous MultiStar*, *Homogeneous Partial MultiStar* y *Generalized Large MultiStar*.

Si al finalizar el llamado cíclico de estas rutinas de separación no se encuentra ninguna desigualdad violada, se sale del procedimiento de separación sin cortes. Al terminar el procedimiento sin encontrar desigualdades violadas, el algoritmo principal de *branch&cut* da por finalizada la etapa de separación en el nodo raíz del árbol y procede a generar dos nuevos nodos del árbol (*branching*). Cada uno de estos nodos es un nuevo subproblema a optimizar.

La explicación precedente se refiere a la estrategia utilizada por el algoritmo de separación en el nodo raíz. La estrategia utilizada para el resto de los nodos tiene como objetivo destinar menos tiempo al algoritmo de separación que el destinado al nodo raíz, ya que se observó que es más efectivo tratar de realizar *branching* lo antes posible.

En primer lugar, se llama a la rutina de separación de las *desigualdades de Capacidad*. Independientemente de que se haya o no encontrado restricciones violadas, se pasa a llamar a la rutina de separación de las *desigualdades de Lógica de Ruta 1*. A continuación, se llama a las *desigualdades Framed Capacity*. Esta llamada le especifica al algoritmo, un máximo de 100 nodos a analizar en el árbol de búsqueda. Por último, se llama a las rutinas de las clases *MultiStar*, *Comb*, *Hypotour*, *Camiones Ordenados* y *Rutas Dirigidas* en el orden estipulado, permitiendo reoptimizar el LP sólo en caso de que los cortes que fueran generados en cada paso superen el máximo de 500 unidades. La llamada a todas las rutinas de separación se realiza una sola vez por nodo. En caso de repetirse el proceso por haber reoptimizado el LP, se continúa llamando solamente a las rutinas de separación de las clases *Capacidad*, *Lógica de Rutas 1*, *Camiones Ordenados* y *Rutas Dirigidas*. Este proceso se repite hasta que no se encuentre ninguna desigualdad violada.

Las desigualdades válidas identificadas en cada una de las etapas del proceso de separación son agregadas al sistema lineal antes de volver a reoptimizar. Cada reoptimización del sistema lineal permite una nueva llamada al procedimiento de separación. Como se explicó anteriormente, cuando no se identifican más restricciones violadas, se sale del proceso de separación y se procede a realizar el *proceso de branching*. En el *proceso de branching* se crean (en base a la estrategia de branching) 2 nuevos subproblemas del nodo que estaba siendo analizado. Estos subproblemas heredan de su padre el sistema lineal último que se reoptimizó con el agregado de la restricción que genera el branching. Es decir, el sistema lineal con el que comienza a optimizarse un nuevo nodo, coincide con el último sistema

lineal de su padre, más la restricción agregada por el *proceso de branching.*

Es importante destacar también que cada uno de los cortes que se generan en la etapa de separación son albergados en un *pool* de cortes. Esta manera de almacenar los cortes permite ahorrar memoria en el *framework ABACUS*, ya que la misma desigualdad puede ser utilizada por diferentes nodos del árbol de enumeración.

4.2. Estrategia de Branching

El proceso de branching es otra de las partes críticas de los algoritmos de *branch&cut.* La estrategia que se utilice para generar los nuevos hijos del nodo activo, tiene especial importancia debido a que determina el tamaño y la forma que tendrá el árbol de enumeración.

La estrategia de branching más utilizada se basa en generar dos nuevos subproblemas acotando el rango que puede llegar a tomar una determinada variable x_i. En el caso de variables binarias, la estrategia de branching genera un hijo con la restricción adicional $x_i = 0$ y otro con la restricción $x_i = 1$. De la misma manera, se puede realizar branching sobre variables continuas definiendo intervalos disjuntos. Existen diferentes maneras de seleccionar las variables x_i candidatas a ser utilizadas como regla de branching. Trasladando la idea de branching sobre variables al problema CVRP, las variables preferidas son las representantes de ejes que interconectan el depósito con los clientes o las que interconectan clientes entre sí. Si bien este método es sencillo y muy utilizado en la práctica, tiene la desventaja, para el problema CVRP, de que genera un árbol de enumeración extremadamente desbalanceado. Este hecho es sencillo de explicar debido a que fijar una variable que interconecta clientes en 1, determina que ambos clientes formarán parte de una misma ruta y, además, se fija uno de los $n + K$ ejes que formarán parte de la solución final.

Por otra parte, en el otro hijo generado, se fijará la misma variable en 0. Fijar una variable que interconecta clientes en 0, es decir que uno de los $O(n^2)$ ejes que forman parte del problema, no deberá formar parte de la solución final.

Este problema de asimetría ha concentrado los esfuerzos de algunos investigadores en la búsqueda de estrategias de branching que tiendan a generar árboles más balanceados. En (TV01) se propone como alternativa realizar branching sobre desigualdades, específicamente sobre conjuntos de corte. La idea fundamental es buscar un conjunto S que cumpla $2 < X^*(\delta(S)) < 4$ y generar dos nuevos nodos con la disyunción $(X(\delta(S)) = 2) \vee (X(\delta(S)) \geq 4)$. Este proceso se reduce a branching de variables cuando $\mid S \mid = 2$.

La clave para tratar de obtener un árbol balanceado se encuentra en la buena elección del

conjunto S sobre el que se generará el branching. En (Aug95) se proponen seis estrategias distintas para encontrar los conjuntos S:

1. Seleccionar S tal que $X(\delta(S))$ esté lo más cerca posible a 3,0.
2. Seleccionar S tal que $X(\delta(S))$ esté lo más cerca posible a 2,85.
3. Seleccionar S tal que $X(\delta(S))$ esté lo más cerca posible a 3,15.
4. Seleccionar S tal que $2{,}75 \leq X(\delta(S)) \leq 3$ y $d(S)$ sea máximo ($d(S)$ representa la suma de las demandas de los clientes en S).
5. Seleccionar S tal que $2{,}75 \leq X(\delta(S)) \leq 3$ y la distancia de S al depósito sea máxima (los empates son definidos usando la demanda total). Se define distancia como la suma de las distancias al depósito de los dos nodos de S que se encuentran más cerca al mismo.
6. Seleccionar un S que cumpla $2{,}75 \leq X(\delta(S)) \leq 3$ y que contenga la mayor cantidad de supernodos. Un supernodo es un nodo que ha sido fusionado mediante el *proceso de shrinking* desarrollado en el algorimo mostrado en la sección 4.1.1 paso 2.

En este algoritmo de *branch&cut* se desarrolló una heurística constructiva basada en el proceso implementado para la detección de *desigualdades de capacidad* 4.1.1. Se arma una lista de conjuntos candidatos y se seleccionan 6 que cumplan las estrategias enumeradas anteriormente. En algunos casos puede haber menos de 6 finalistas debido a que no siempre hay conjuntos que cumplen todas las condiciones anteriores. En caso de que no se encuentre ningún conjunto S con las estrategias de (Aug95) mostradas, se procede a realizar branching sobre variables.

4.2.1. Branching sobre conjuntos

A partir de la lista generada en base a las reglas anteriores, se selecciona la finalista para generar el nuevo par de nodos, utilizando un proceso denominado branching reforzado (en inglés strong branching). La idea que subyace a *strong branching* es buscar el mejor conjunto para generar el branching utilizando, para ello, una iteración del optimizador lineal para cada uno de los conjuntos candidatos. Se optimiza el último sistema lineal (nodo activo) más el agregado de la restricción adecuada para generar el supuesto nuevo nodo del árbol. En base a los resultados obtenidos, se selecciona el conjunto que promete generar un árbol más balanceado.

4.2.2. Branching sobre variables

En caso de que el proceso de selección de candidatos no encuentre ningún conjunto S que satisfaga las reglas anteriores, se procede a realizar branching sobre variables.
La estrategia utilizada analiza las variables del último sistema resuelto con valores $0 < x_i < 1$, teniendo en cuenta la siguiente jerarquía de variables:

- Se verifica primero si hay variables que interconectan clientes (ejes). Se utiliza como regla la estrategia *CloseHalfExpensive*, que significa seleccionar entre todas las variables cuyo valor se encuentra más cerca a 0,5 la que tenga la distancia mínima asociada al eje.
- En caso de no conseguir variables del tipo anterior, se buscan variables que conecten los *nodos vehículo* con los *nodos cliente*. Se utiliza el mismo criterio *CloseHalfExpensive* que para las variables interclientes.
- Se analizan las variables que unen *nodos cliente* con el nodo depósito (d). Aquí también se utiliza el criterio de selección *CloseHalfExpensive*.
- Se verifican las variables que unen el nodo depósito *o* con los *nodos vehículo*. Luego, se verifican los multiejes dentro de los *nodos vehículo*. Por último, se verifican los multiejes de los *nodos cliente*. Todas estas variables se recorren utilizando la estrategia de seleccionar la primera que no contenga un valor 0 o 1.

4.2.3. Estrategia de Enumeración

La estrategia de enumeración que mejores resultados ha mostrado es la llamada *Dive and Best*. Esta estrategia realiza *Depth First Search - DFS* (primero buscar en profundidad) hasta encontrar una solución factible y después continúa realizando *Best First Search* (primero buscar la mejor) que consiste en elegir, de los nodos abiertos, el que tiene la menor cota inferior. *Dive and Best* combina lo mejor de las estrategias *DFS* y *Best First Search* ya que encuentra rápido una solución factible que marque una cota inferior y luego selecciona los siguientes nodos de manera tal de generar el menor árbol de *branch&cut* posible. La desventaja de Best First Search es que puede llegar a mantener abiertos a un gran número de nodos y consumir, por consiguiente, grandes cantidades de memoria.

4.2.4. Detección de problemas no factibles

Dentro de las alternativas que surgen en un proceso de enumeración, aparece la posibilidad de generar subproblemas inconsistentes mediante el proceso de *branching*. Tales inconsistencias desembocan en problemas enteros no factibles, por lo que su pronta detección resulta crucial a la hora de poder eliminar el nodo del árbol y continuar con el análisis

de otros subproblemas.

En el algorimo de *branch&cut* implementado, no se ha realizado un tratamiento especial para detectar problemas no factibles. La regla que se utiliza para *podar* un nodo (eliminarlo por infactibilidad) es la inducida por la no factibilidad de la relajación lineal asociada al subproblema. Es decir, cuando el optimizador lineal determina que la relajación lineal asociada al subproblema activo no es factible, puede podarse el nodo inmediatamente debido a que el subproblema entero que representa también debe ser no factible.

4.3. Heurísticas Primales

Otra característica que determina la eficiencia de un algoritmo de *branch&cut* es la velocidad con que se logra disminuir la cota superior global de la optimización. Como se explicó anteriormente, la cota superior de la optimización se encuentra regida por la obtención de soluciones factibles durante el transcurso de la optimización. Cuanto más rápido disminuya el valor de la cota superior, menos nodos del árbol de enumeración deberán ser generados. En el presente algoritmo se utilizan dos heurísticas orientadas a encontrar soluciones factibles. La primera se aplica antes de comenzar con la generación del árbol de enumeración. Esta heurística suele recibir el nombre de *heurística primal.*

La segunda heurística es utilizada por el algoritmo de *branch&cut*, al finalizar cada una de las optimizaciones de las relajaciones lineales. La idea de la misma es tratar de mejorar la solución no entera obtenida en el subproblema activo para generar una solución factible. A este tipo de heurísticas se las suele denominar *heurísticas de mejoramiento primal.*

4.3.1. Búsqueda de solución inicial

En la presente implementación no se ha desarrollado un algoritmo primal para generar una solución factible inicial. Si bien se ejecuta una heurística, la misma ha sido implementada para inicializar la cota superior global. La solución obtenida no se utiliza para construir el sistema lineal inicial con el que comenzará la optimización. Se considera que dicho procedimiento podría llegar a mejorar los resultados obtenidos y queda pendiente para trabajos futuros.

Para inicializar la cota superior, se implementó en primer lugar, una heurística golosa que asigna los clientes a los vehículos considerando la menor asignación posible de camiones. Con el resultado de esta heurística, se determina la cota máxima de vehículos que puedan llegar a requerirse durante todo el proceso. En base a dicha cota, se dimensiona la cantidad

de *nodos vehículo* y la cantidad de multiejes dentro de cada uno de los *nodos cliente.*

La segunda heurística utilizada es el conocido método de *ahorros* de Clarke&Wright (CW64). Esta heurística se aplica con la intención de mejorar el valor de la cota superior global encontrada, con la heurística anterior. La idea fundamental de dicho método es intentar combinar dos rutas y obtener una nueva ruta siempre y cuando dicha fusión signifique una disminución de la distancia total recorrida por la flota. Se suele inicializar el proceso estableciendo n rutas distintas (una para cada cliente) y se las combina teniendo en cuenta las que mejores ahorros proporcionen. La idea básica del algoritmo puede observarse en el siguiente pseudocódigo:

1. Calcular el vector de ahorros $s_{ij} = d_{i0} + d_{0j} - d_{ij}$ para $i, j = 1, ..., n$ $i \neq j$. Crear n rutas $(0, i, 0)$ para cada uno de los clientes $i = 1, ..., n$. Ordenar los ahorros de manera decreciente.
2. Recorrer el vector de ahorros ordenado. Para cada ahorro s_{ij} se verifica si existen dos rutas, una conteniendo el eje $(0, j)$ y la otra conteniendo el eje $(i, 0)$. Si es así se las combina borrando los ejes $(0, j)$ y $(i, 0)$, y agregando el eje (i, j) a la ruta resultante.

Este algoritmo tiene una complejidad polinomial $O(n^3)$ debido a que se ha utilizado el algoritmo *quicksort* para ordenar el vector de ahorros. Esta complejidad teórica puede disminuirse a $O(n^2 log(n))$ si se utiliza el algoritmo *heapsort* para ordenar dicho vector.

Se consideró implementar una heurística inicial con alguna metaheurística agresiva que permitiera encontrar una cota superior lo más cercana posible al óptimo similar a *TABUROUTE* (GHL94) o a *Granular Tabu Search* (TV03). Ambas heurísticas consumen un tiempo computacional significativamente mayor al dedicado por las heurísticas detalladas anteriormente. En contraposición, obtienen resultados muy cercanos al óptimo del problema.

Al analizar el desenvolvimiento global del algoritmo de *branch&cut*, pudo observarse que la *heurística de mejoramiento primal* implementada encontraba, durante el transcurso de la optimización, soluciones factibles cercanas a la óptima. Para la mayoría de las instancias analizadas, la cota superior obtenida, utilizando la heurística de mejoramiento, se acercó relativamente rápido a la solución óptima del problema. Este hecho determinó dejar esta ardua tarea para trabajos futuros. Se considera que dicho esfuerzo puede llegar a representar un incremento importante en la eficiencia del algoritmo de *branch&cut* ya que ayudaría a generar un árbol de enumeración mucho más reducido.

En la sección de resultados, se mostrará el desenvolvimiento del algoritmo implementado utilizando las heurísticas mencionadas. Posteriormente, se presentarán los resultados

de la ejecución considerando el ingreso externo de los valores correspondientes a las cotas superiores, conocidas para cada una de las instancias. Esta práctica es muy habitual en los trabajos de *branch&cut* publicados ya que asumen la posibilidad de contar con alguna buena heurística que proporcione las cotas superiores y dejan la tarea de su implementación para investigaciones en el área de búsqueda de soluciones aproximadas.
En los trabajos (ABB+95; BH02; ACH03; BHM04; LLE04; FLdAa+06) se utilizan las mejores cotas superiores conocidas hasta el momento de la publicación del trabajo como parámetros externos ingresados al algoritmo, es decir, en dichos trabajos los algoritmos implementados omiten el cálculo de cotas superiores.

4.3.2. Heurística de mejoramiento

Después de obtener la solución de cada sistema lineal, se procede a intentar extraer información de los valores asignados a cada una de las variables del sistema optimizado. La heurística utilizada consiste en ordenar las variables que interconectan *nodos cliente*.

Las variables que se encuentran fijadas o seteadas en 1 serán las que tendrán las primeras posiciones en dicho ordenamiento. Las variables que tienen valores $0 < x_i < 1$ se ordenan de mayor a menor, es decir, tienen mayor prioridad las variables que se encuentran más cerca de 1. Las variables que satisfacen la igualdad de todos los multiejes correspondientes a los clientes asociados a la misma, son premiadas y se les da mayor prioridad en el ordenamiento. Cuando hay variables con el mismo valor, se da prioridad a las que tienen menor distancia asociada (función $d(x_i)$). Una vez armada la lista con las reglas anteriores, se efectúa el ordenamiento utilizando el algoritmo *quicksort*. Con la lista ordenada se procede a intentar construir una nueva solución que satisfaga las restricciones de capacidad de cada uno de los vehículos.

A la solución obtenida anteriormente, se le aplica el algoritmo de ahorros de Clarke&Wright con la esperanza de mejorar aún más la solución obtenida. Aquí se utiliza una variación del algoritmo presentado anteriormente para que se inicialice la optimización en base a una solución establecida. En la versión original, la inicialización se efectuaba asignando un cliente a cada vehículo y se comenzaba a optimizar con n vehículos.

Como se describió en la sección 4.3.1, esta estrategia resultó muy efectiva y logró disminuir, en la mayoría de los casos, la cota superior global del algoritmo de *branch&cut* en un número reducido de optimizaciones del sistema lineal.

4.4. Fijado y Seteado de variables por implicaciones lógicas

El fijado y seteo de variables es otra de las estrategias que permite mejorar el desenvolvimiento global del algoritmo de *branch&cut*. La diferencia entre *fijar* o *setear* una variable radica en que una variable que es fijada en 0 o 1 mantiene dicho valor por el resto de la optimización. En cambio, una variable que es seteada en 0 o 1 mantiene dicho valor sólo para el nodo activo donde la misma fue seteada y para los subnodos que dependan del nodo activo en el árbol de enumeración.

El fijado y seteo de variables puede ser realizado en forma automática por el framework *ABACUS*, utilizando los costos reducidos de las variables que no son básicas del sistema lineal. El *fijado por Costo Reducido* sólo puede ser aplicado en el nodo raíz del árbol de *branch&cut*. El *seteado por Costo Reducido* se realiza con el mismo criterio recién explicado pero el valor en que se setea la variable sólo se mantiene en el nodo activo y su descendencia. En (Thi95; J̈99) puede obtenerse una explicación detallada de la manera en que *ABACUS* fija y setea variables.

Inmediatamente después de que el framework fija y setea variables, el usuario del mismo puede implementar rutinas específicas al problema que permitan fijar o setear variables mediante implicaciones lógicas. Se utiliza como fuente de inferencia el conjunto de variables que han sido fijadas y seteadas durante el transcurso de la optimización.

En el algoritmo de *branch&cut* implementado, se utilizó la lista de reglas de inferencia que se detalla a continuación:

1. **El eje es del tipo** $(q_{ib}, q_{ic}) \Rightarrow$ Se fija en 1.
2. **El eje es del tipo** (m_{ka}, m_{kb}, l) $k \neq l \Rightarrow$ Se fija en 0.
3. **El eje es del tipo** (o, m_{ka}) **y está fijado o seteado en** $0 \Rightarrow$ se setea o fija en 0 todos los ejes (q_{ja}, q_{jb}, k) $j = 1...n$.
4. **El eje es del tipo** (m_{ka}, m_{kb}, k) **y está fijado o seteado en** $0 \Rightarrow$ se setea o fija en 0 todos los ejes (q_{ja}, q_{jb}, k) $j = 1...n$.
5. **El eje es del tipo** (m_{kb}, q_{ia}) **y está fijado o seteado en** $1 \Rightarrow$ se setea o fija en 1 el eje (m_{ka}, m_{kb}, k); se setea o fija en 0 todos los ejes (m_{kb}, q_{ja}) con $j \neq i, j = 1...n$; se setea o fija en 1 el eje (q_{ia}, q_{ib}, k); se setea o fija en 0 los ejes (q_{ia}, q_{ib}, l) con $l \neq k$; se setea o fija en 0 todos los ejes (q_{jc}, q_{ia}) con $j \neq i, j = 1...n$.

6. **El eje es del tipo** (q_{ia}, q_{ib}, k) **y está fijado o seteado en** $1 \Rightarrow$ se setea o fija en 0 los ejes (q_{ia}, q_{ib}, l) con $l \neq k$, se setea o fija en 0 todos los ejes (m_{lb}, q_{ia}) con $l \neq k$.

7. **El eje es del tipo** (q_{ia}, q_{ib}, k) **y está fijado o seteado en** $0 \Rightarrow$ se setea o fija en 0 el eje (m_{kb}, q_{ia}).

8. **El eje es del tipo** (q_{ic}, q_{ja}) **y está fijado o seteado en** $1 \Rightarrow$ se setea o fija en 0 todos los ejes (q_{ic}, q_{fa}) con $f \neq i, f \neq j, f = 1...n$; se setea o fija en 0 el eje (q_{ic}, d); se setea o fija en 0 todos los ejes (q_{fc}, q_{ja}) con $f \neq i, f \neq j, f = 1...n$; se setea o fija en 0 todos los ejes (m_{kb}, q_{ja}) con $k = 1...K$; se setea o fija en 0 el eje (q_{jc}, q_{ia}); si además (q_{ia}, q_{ib}, k) está fijado o seteado en 1 se setea o fija en 1 el eje (q_{ja}, q_{jb}, k) y si (q_{ja}, q_{jb}, k) está fijado o seteado en 1 se setea o fija en 1 el eje (q_{ia}, q_{ib}, k).

9. **El eje es del tipo** (q_{ic}, d) **y está fijado o seteado en** $1 \Rightarrow$ se setea o fija en 0 todos los ejes (q_{ic}, q_{ja}) con $j \neq i, j = 1...n$.

En base a la lista mostrada, el algoritmo intenta fijar o setear variables por implicaciones lógicas. La regla para fijar una variable es que las variables que permiten la inferencia se encuentren también fijadas. Si alguna está seteada, la implicación determinará un seteo y no una fijación de la variable analizada.

4.5. Otras particularidades de la implementación

4.5.1. Relajación inicial

La relajación inicial se conformó a partir de las *desigualdades de conservación de flujo*, las *desigualdades de Lógica de Ruta 2* y las *desigualdades de Control de Capacidad Máxima*. Esta última familia podría haberse omitido debido a que se encuentra contenida por las *desigualdades de Capacidad*. Sin embargo, la experimentación ha demostrado que dicha familia junto a las *desigualdades de Lógica de Ruta* orientan al optimizador lineal a obtener soluciones válidas con mayor velocidad que si se las omite. Por otra parte, se requiere sólo una restricción de esta clase por cada vehículo, totalizando, de esta manera, un número reducido y despreciable de restricciones.

El resto de las desigualdades presentadas se utilizan como desigualdades válidas agregadas en el proceso de separación.

4.5.2. Pools de Desigualdades

Como se mencionó anteriormente, las desigualdades válidas utilizadas como cortes se almacenan en contenedores específicos denominados *pooles de cortes* o de desigualdades.

Por un lado, los *pooles* son utilizados para minimizar la utilización de la memoria ya que una restricción es generada físicamente en un sólo lugar y referenciada en múltiples subproblemas a medida que se la necesita.

En segundo lugar, los *pooles de cortes* pueden servir como primera fuente de generación de desigualdades válidas, antes de invocar las rutinas de separación. En estos casos, se recorre linealmente la lista de restricciones del pool y se verifican las desigualdades que no se encuentran presentes en el sistema lineal activo. Si se encuentran restricciones violadas no activas, se las puede volver a ingresar al sistema lineal.

Como se describe en la sección siguiente, las desigualdades que dejan de ser útiles al sistema lineal en curso son eliminadas del mismo, para hacer más eficiente la tarea del optimizador lineal. Sin embargo, siguen permaneciendo en los pooles respectivos para su consideración futura.

En el algoritmo de *branch&cut* implementado se utilizó un *pool de cortes* para cada una de las familias de desigualdades válidas, enunciadas al comienzo del capítulo. Además se agregó también un pool específico para las desigualdades generadas mediante las reglas de *branching*.

4.5.3. Eliminación de Desigualdades Inactivas

Los optimizadores lineales son muy sensibles al tamaño del sistema lineal a optimizar. A medida que el proceso de *branch&cut* evoluciona, el sistema lineal va conteniendo cada vez más desigualdades válidas. Si no se considera revisar la lista de desigualdades activas para encontrar desigualdades que han dejado de ser "importantes"para la solución que se está analizando, la velocidad de resolución comenzará a disminuir notablemente, provocando una drástica pérdida de eficiencia en la resolución global del problema. Debido a ello, el algoritmo implementado verifica y elimina las restricciones que han dejado de ser "importantes" durante las últimas 3 reoptimizaciones del sistema lineal. Se considera que una restricción deja de ser "importante" cuando las variables básicas dejan de formar parte de la base del sistema lineal. En otras palabras, alguna de las variables de *slack* pasa a formar parte de la base y la restricción pierde efectividad, dejando de ser necesaria para el sistema en curso.

4.5.4. Identificación de soluciones válidas

Otra funcionalidad que deben proporcionar los algoritmos de *branch&cut* para ser competitivos, es la verificación eficiente de soluciones válidas. Después de optimizar un sistema

lineal, el algoritmo de *branch&cut* debe chequear si la solución encontrada es o no es válida. Si es válida, puede llegar a ser una solución que supere la mejor solución encontrada hasta el momento y, en ese caso, corresponderá almacenar la solución y actualizar la cota superior global del algoritmo.

En la implementación realizada se verificó, en primer lugar, la integralidad de las variables optimizadas. De acuerdo al modelo analizado, alcanza con verificar si todas las variables contienen los valores 1 o 0. En segundo lugar, se procede a verificar que las rutas obtenidas sean válidas. Se considera que una ruta es válida cuando la suma de las demandas de todos los clientes que componen la ruta no sobrepasa la capacidad del vehículo. Además, las rutas deben comenzar y finalizar en el depósito (no pueden contener subtours).

El funcionamiento básico del algoritmo implementado puede observarse en el siguiente pseudocódigo:

1. Crear un conjunto para cada uno de los clientes $i = 1, ..., n$. Asociar a cada conjunto un valor que represente la demanda total del mismo.
2. Mientras existan ejes interclientes (i, j) sin analizar hacer:
   a) Si los nodos i y j están en un mismo conjunto devolver *no factible* y salir.
   b) en caso contrario unir los conjuntos representados por i y j en un nuevo conjunto. Actualizar la demanda total del nuevo conjunto como la suma de las demandas totales de los dos conjuntos unidos.
3. Si la demanda total de todos los conjuntos obtenidos no sobrepasa la capacidad de los vehículos asociados, devolver *factible*, en caso contrario devolver *no factible*.

Para lograr una implementación eficiente, fue necesaria la utilización de una estructura de datos adecuada para representar los conjuntos. La clase *FASTSET* incluída dentro de las herramientas de soporte proporcionadas por el *framework ABACUS* implementa dichos conjuntos utilizando árboles. Se intenta mantenerlos balanceados con el fin de poder verificar la pertenencia de cualquier elemento en un tiempo que se aproxime al logarítmico. Puede obtenerse más información sobre la manera en que dichos conjuntos se encuentran implementados *ABACUS*, en (JÖ9).

Capítulo 5

Resultados Computacionales

Los algoritmos de *branch&cut* utilizan para la resolución de los problemas, la enumeración finita de subproblemas asociados al problema original, con el fin de recorrer todas las soluciones factibles que puedan llegar a ser óptimas. Dicha enumeración es finita pero no existe garantía de que sea polinomial en el peor caso. El énfasis de estos algoritmos está puesto en la identificación de familias de desigualdades válidas, que permitan aumentar en forma significativa la cota inferior global de la optimización y reducir, por consiguiente, el árbol de enumeración asociado.

La experimentación computacional permite evaluar el desempeño del algoritmo de *branch&cut* utilizando las distintas desigualdades válidas propuestas y las diferentes decisiones adoptadas durante el desarrollo del mismo. Es también a través del análisis de los resultados que puede llegar a mejorarse el algoritmo, retroalimentando las políticas y estrategias seleccionadas para cada etapa del mismo. Asimismo, la experimentación es la herramienta fundamental para el ajuste de los parámetros que rigen el comportamiento general del algoritmo. El análisis empírico permite, además, identificar etapas que no generen los rendimientos esperados o desigualdades válidas que no se desempeñen de acuerdo a lo proyectado.

Como se mencionó al comienzo de este trabajo, las pruebas han sido realizadas sobre instancias utilizadas en la mayoría de los trabajos publicados sobre Ruteo de Vehículos con Capacidades. Las mismas han sido obtenidas de (Rei95; Ral95). *En TSPLIB* (Rei95) se define también un formato standard para los archivos que contienen las instancias de prueba de CVRP. En *www.branchandcut.org* (Ral95) pueden encontrarse todas las instancias de *TSPLIB* más un conjunto actualizado de las nuevas instancias propuestas en diferentes trabajos. Se han utilizado para la evaluación de los resultados sólo las instancias Euclidianas que poseen costos y demandas enteras (siguiendo el standard de *TSPLIB*).

El algoritmo ha sido implementado utilizando el *framework ABACUS 2.4* y el optimi-

zador lineal *CPLEX 6.0*. La ejecución de las pruebas ha sido efectuada sobre un sistema operativo *Linux* con procesador Pentium IV 3.0 GHz y 512 MB de memoria RAM. El compilador de C++ utilizado es el *GCC 2.95*, proporcionado por la distribución de linux.

5.1. Instancias de prueba

Las instancias de prueba seleccionadas de (Ral95) tienen normalizado los nombres de archivo de la siguiente manera: **[Familia]n[nroN]k[nroV].vrp** donde **[Familia]** es el nombre de la familia de instancias, **[nroN]** es la cantidad de nodos utilizadas por cada una de las instancias y **[nroV]** es el número de vehículos necesarios para obtener la mínima asignación de camiones.

Se han utilizado para los tests las familias "A" y "B" propuestas en Augerat (Aug95), la familia "E" propuesta en Christofides & Eilon (CE69), la familia de instancias "M" propuestas en (CMT79) y la familia "F" introducida en (Fis94).

5.2. Evaluación de las distintas desigualdades válidas utilizadas

Para analizar los resultados obtenidos utilizando las distintas desigualdades válidas, se ha adoptado el criterio propuesto en (LLE04). Es decir, se ha elegido un conjunto reducido de instancias para poder efectuar la puesta a punto del algoritmo y sus parámetros. En 5.1 se muestran las instancias, la capacidad de los vehículos Q, la carga (demanda total dividida por KQ, donde K es la cantidad de vehículos) y las mejores cotas superiores (UB) conocidas hasta el momento, tomadas de (BH02; TV01; LLE04; FLdAa$^+$06). En todos los casos se indica con "*" que las cotas han sido probadas como óptimas.

En primer lugar, se mostrarán los resultados obtenidos en el nodo raíz del árbol de enumeración, considerando únicamente el agregado de *desigualdades de capacidad*, *desigualdades de Lógica de Ruta 1*, *desigualdades de Rutas Dirigidas* y *desigualdades de Camiones Ordenados*. En la tabla 5.2 se muestran los resultados obtenidos en (ABB$^+$95; ABB$^+$98) (bajo el título "AEA"), los publicados en (LLE04) (bajo el título "LLE") y los obtenidos con el nuevo modelo implementado. En todos los casos se muestran las cotas inferiores obtenidas (LB) utilizando las *desigualdades de capacidad*. En "LLE" y en el modelo implementado se presentan también los tiempos de ejecución (en segundos) requeridos para lograr dichas cotas.

instancia	Q	carga	UB
E-n22-k4	6000	0,94	375*
E-n23-k3	4500	0,75	569*
E-n30-k3	4500	0,94	534*
E-n33-k4	8000	0,92	835*
E-n51-k5	160	0,97	521*
E-n76-k7	220	0,89	682*
E-n76-k8	180	0,95	735*
E-n76-k10	140	0,97	830*
E-n76-k14	100	0,97	1021*
E-n101-k8	200	0,91	815*
E-n101-k14	112	0,93	1067*
M-n101-k10	200	0,91	820*
F-n45-k4	2010	0,90	724*
F-n72-k4	30000	0,96	237*
F-n135-k7	2210	0,95	1162*

Cuadro 5.1: Instancias seleccionadas

Las cotas inferiores resaltadas corresponden a cotas que superan las publicadas en los trabajos anteriores. Es importe destacar que en las instancias con un número reducido de nodos, los tiempos de ejecución del presente algoritmo mejoran los tiempos de "LLE". Sin embargo, a medida que el tamaño de los problemas analizados crece, dicha situación se revierte. Por un lado, el hardware utilizado para presentar los nuevos resultados supera al utilizado en "LLE". Esto explica la mejora de velocidad en las instancias pequeñas, sobre todo sabiendo que la rutina de separación de *desigualdades de capacidad* utilizada es la misma que la utilizada en "LLE" (CVRPSEP). Por otra parte, es entendible la situación observada en instancias grandes debido a que el presente modelo utiliza un número de variables y restricciones superior al utilizado en los trabajos "AEA" y "LLE", dónde se utiliza la tradicional formulación de flujo de 2 índices. Además, dentro de la etapa de separación, se realiza también la separación de las *desigualdades de Lógica de Ruta 1*, de *Camiones Ordenados* y *Rutas Dirigidas*.

Hubiese sido adecuado comparar el presente algoritmo con trabajos publicados con algoritmos de *branch&cut*, implementados en base a *modelos de flujo de 3 índices* ya que las particularidades de los modelos de flujo explicadas en el capítulo 3 determinan que el modelo propuesto se asemeja mucho más a los *modelos de flujo de 3 índices*. Lamentablemente, no se han encontrado resultados publicados de algoritmos de *branch&cut* basados en tales modelos. Sin embargo, los resultados muestran que el desempeño del presente algoritmo es competitivo y resuelve las instancias en tiempos razonables, comparándolas incluso con los modelos de flujo de 2 índices. Esta misma situación se repetirá en la mayoría de los

instancia	AEA	LLE		Este modelo	
	LB	LB	Tiempo	LB	Tiempo
E-n22-k4	375	375	2	375*	0,42
E-n23-k3	569	569	2	569*	0,1
E-n30-k3	508,5	508,5	2	508,5	0,4
E-n33-k4	833,5	833,5	4	833,5	1,17
E-n51-k5	514,524	514,524	4	514,524	3,42
E-n76-k7	661,299	661,358	7	661,356	22,77
E-n76-k8	711,17	711,201	10	**711,205**	26,27
E-n76-k10	789,416	789,441	19	**789,45**	48,72
E-n76-k14	-	947,983	27	**948,083**	95,25
E-n101-k8	796,314	796,414	12	**796,422**	38,27
E-n101-k14	-	1008,15	65	**1008,16**	159,73
M-n101-k10	819,5	819,5	30	819,5	77,88
F-n45-k4	724	724	4	724*	0,98
F-n72-k4	232,5	232,5	3	232,5	1,72
F-n135-k7	1158,25	1158,25	59	1158,25	150,57

Cuadro 5.2: Cotas inferiores (LB) con Desigualdades de Capacidad

instancia	AEA	LLE		Este modelo	
	LB	LB	Tiempo	LB	Tiempo
E-n30-k3	532,5	532,5	7	532,5	0,84
E-n76-k14	-	-	-	948,093	120,01
F-n72-k4	235	236,25	16	236,25	8,7

Cuadro 5.3: Cotas inferiores (LB) con Desigualdades FCI

resultados presentados durante el resto del capítulo.

A continuación, se presentará el desempeño del algoritmo de separación de las *desigualdades Framed Capacity (FCI)*. Para generar la tabla de resultados 5.3 se realizó la separación de *desigualdades de capacidad*, *desigualdades de Lógica de Ruta 1*, *desigualdades de Rutas Dirigidas*, *desigualdades de Camiones Ordenados* y *desigualdades Framed Capacity*. Sólo tres instancias mostraron algún tipo de mejora. A diferencia de lo presentado en (LLE04) ("LLE"), donde sólo las instancias E-n30-k3 y F-n72-k4 habían mostrado incrementos, en la presente implementación se ha obtenido también mejoras en la instancia E-n76-k14. Por otra parte, los tiempos obtenidos son mejores que los publicados en "LLE".

Llegó ahora el turno de analizar los resultados obtenidos mediante la aplicación de las *desigualdades Strengthened Comb*. Para presentar los resultados, se realizó la separación de

instancia	AEA	LLE		Este modelo	
	LB	LB	Tiempo	LB	Tiempo
E-n51-k5	517,111	517,176	10	**517,267**	4,72
E-n76-k7	662,661	663,329	20	**663,361**	25,1
E-n76-k8	712,415	713,342	21	713,308	37,99
E-n76-k10	790,792	791,224	34	**791,249**	65,97
E-n76-k14	-	948,018	30	**948,142**	105,05
E-n101-k8	798,358	799,088	38	**799,453**	68,15
E-n101-k14	-	1010,04	88	**1010,19**	217,37
F-n135-k7	1158,48	1158,83	124	1158,65	209,98

Cuadro 5.4: Cotas inferiores (LB) con Desigualdades Strengthened Comb

las *desigualdades de capacidad*, *desigualdades de Lógica de Ruta 1*, *desigualdades de Rutas Dirigidas*, *desigualdades de Camiones Ordenados* y las *desigualdades Strengthened Comb*. Las instancias que mostraron incrementos con el agregado de las *desigualdades Strengthened Comb* se presentan en el cuadro 5.4.

Al igual que lo analizado para las *desigualdades de Capacidad*, se obtuvieron tiempos mejores en las instancias más pequeñas y tiempos peores en las instancias de mayor tamaño. Es importante destacar que en algunas instancias (resaltadas en negrita), las cotas inferiores obtenidas con la presente implementación superaron a las obtenidas en "LLE".

Se continuará con la presentación de los resultados obtenidos mediante la aplicación de las *desigualdades MultiStar y Partial MultiStar*. Al igual que con las pruebas realizadas anteriormente, se realizará la separación de las *desigualdades de capacidad*, *desigualdades de Lógica de Ruta 1*, *desigualdades de Rutas Dirigidas*, *desigualdades de Camiones Ordenados* y las *desigualdades MultiStar y Partial MultiStar*. A diferencia de los cuadros mostrados, en el cuadro 5.5 se mostrará sólo los resultados obtenidos en "LLE" ya que en "AEA" no se realizó separación de *desigualdades MultiStar*.

Nuevamente, se observan tiempos inferiores en algunas de las instancias pequeñas y tiempos superiores en las instancias con mayor número de nodos. Se han obtenido mejores resultados en cuatro de las instancias mostradas.

Se presentaran ahora los resultados obtenidos con las *desigualdades Hypotour*. De igual manera, se realizó el testeo utilizando las familias de *desigualdades de capacidad*, *desigualdades de Lógica de Ruta 1*, *desigualdades de Rutas Dirigidas*, *desigualdades de Camiones Ordenados* más las *desigualdades Hyportour*. Se presenta en 5.6 los resultados obtenidos y su comparación con los obtenidos en "AEA" y "LLE".

En el caso de los resultados mostrados con las *desigualdades de Hypotour* ocurre que la mayoría de los tiempos de ejecución son similares a los publicados en "LLE" y algunas cotas inferiores son mejores aunque otras peores. Estas pequeñas diferencias se atribuyen a

instancia	LLE		Este modelo	
	LB	Tiempo	LB	Tiempo
E-n33-k4	833,875	6	833,8	1,22
E-n51-k5	514,556	4	514,556	5,85
E-n76-k7	663,173	24	663,144	37
E-n76-k8	714,33	43	714,147	62,99
E-n76-k10	796,501	103	**796,63**	144,02
E-n76-k14	966,281	202	**966,455**	473,41
E-n101-k8	798,687	53	798,608	81,73
E-n101-k14	1023,79	363	**1024,39**	1320
M-n101-k10	820*	33	820*	96,38
F-n135-k7	1159,59	152	**1159,64**	274

Cuadro 5.5: Cotas inferiores (LB) con Desigualdades MultiStar y Partial MultiStar

instancia	AEA	LLE		Este modelo	
	LB	LB	Tiempo	LB	Tiempo
E-n30-k3	509,833	508,5	2	508,5	0,24
E-n33-k4	834,154	834,353	12	834,348	1,46
E-n51-k5	515,96	516,794	8	516,784	4,56
E-n76-k7	663,261	663,423	53	**663,445**	34,6
E-n76-k8	712,578	712,908	34	**712,933**	38,78
E-n76-k10	793,089	794,249	81	**794,254**	80,36
E-n76-k14	-	956,019	80	955,993	141,72
E-n101-k8	798,097	798,949	91	798.972	90,13
E-n101-k14	-	1014,803	258	1014,74	278
M-n101-k10	820*	820*	31	820*	88,63
F-n72-k4	233,6	232,5	3	232,5	2,1
F-n135-k7	1158,83	1159,328	181	1159,33	201,63

Cuadro 5.6: Cotas inferiores (LB) con Desigualdades Hypotour

las perturbaciones al sistema lineal, introducidas por la generación de las *desigualdades de Lógica de Ruta 1*, *Camiones Ordenados* y *Rutas Dirigidas*.

Por último, se presentarán los resultados obtenidos utilizando todas las desigualdades expuestas anteriormente en conjunto. El cuadro 5.7 refleja la aplicación de las *desigualdades de Capacidad*, las *desigualdades de Lógica de Ruta 1*, las *desigualdades Framed Capacity*, las *desigualdades MultiStar*, las *desigualdades Strengthened Comb*, las *desigualdades Hypotour*, las *desigualdades de Camiones Ordenados* y las *desigualdades de Rutas Dirigidas*. Para generar los resultados, las desigualdades se separaron en el siguiente orden: Capacidad, Lógica

de Ruta 1, FCI, MultiStar-Comb-Hypotour, Camiones Ordenados y Rutas Dirigidas. Como se explicó en 4.1.10, la estrategia utilizada coincide con la aplicada al nodo raíz del árbol, con el agregado de las *desigualdades de Lógica de Ruta 1*, *Camiones Ordenados* y *Rutas Dirigidas*. Las desigualdades MultiStar-Comb-Hypotour se separan de la manera explicada, utilizando un mecanismo de ciclado que rota el orden en que se las separa.

instancia	UB	LLE		Este modelo	
		LB	Tiempo	LB	Tiempo
E-n22-k4	375*	375	2	375	0,13
E-n23-k3	569*	569	2	569	0,07
E-n30-k3	534*	534	14	533,3	11,42
E-n33-k4	835*	834,707	12	834,42	7,99
E-n51-k5	521*	519	24	518,484	19,59
E-n76-k7	682*	666,408	72	666,279	118,79
E-n76-k8	735*	717,852	136	717,498	139,33
E-n76-k10	830*	799,878	158	799,859	231,61
E-n76-k14	1021*	969,609	181	969,428	429,79
E-n101-k8	815*	802,646	222	802,271	200,05
E-n101-k14	1067*	1026,94	555	1026,61	854,06
F-n45-k4	724*	724	6	724	0,74
F-n72-k4	237*	237	38	236,354	25,05
F-n135-k7	1162*	1160	155	1159,72	156,49
M-n101-k10	820*	820	33	820	33,92

Cuadro 5.7: Cotas inferiores (LB) en el nodo raíz con la aplicación de todas las desigualdades

Observando los resultados, puede verificarse que todas las cotas inferiores (LB) obtenidas con el nuevo algoritmo son inferiores o iguales a las presentados en "LLE". Uno de los principales motivos que puede llegar a marcar la diferencia en las mejores cotas obtenidas por "LLE" es la implementación de una última fase de *cortes de Gomory* (LLE04) que no fue incluída en el presente trabajo.

La idea principal de este trabajo fue presentar una nueva formulación de flujo y analizar su desempeño, comparándola con los trabajos previos del área. Sin embargo, no se pretendió obtener un algoritmo de *branch&cut* que mejore a todos los existentes, sino demostrar la competitividad de la formulación propuesta en una aplicación concreta de *branch&cut*. Es importante destacar también que los tiempos de ejecución en ambas implementaciones resultaron ser muy similares.

5.3. Estrategias de Separación

Como se comentó en la sección 4.1.10, se utilizó una estrategia de separación para el nodo raíz del árbol y otra para el resto de los nodos. Para definir la estrategia final a utilizar en el nodo raíz, se generaron distintos escenarios definidos por el orden de aplicación de los diferentes cortes.

- **Root1:**
  1. Cortes de Capacidad. Si alguno tiene una violación de más de $0,2$ se reoptimiza el sistema lineal (LP).
  2. Cortes de Lógica de Ruta 1. Los cortes encontrados se almacenan y se sigue.
  3. Cortes Framed Capacity (FCI). Si hay algún corte se reoptimiza.
  4. Cortes MultiStar, Comb, Hypotour en llamado cíclico. Si las violaciones respectivas superan $0,05$, $0,1$ o $0,1$ se reoptimiza el LP.
  5. Cortes de Camiones Ordenados y Cortes de Rutas Dirigidas.
  6. Si hay cortes se reoptimiza, si no, se termina la separación del nodo raíz.
- **Root2:**
  1. Cortes de Lógica de Ruta 1. Los cortes encontrados se almacenan y se sigue.
  2. Cortes de Capacidad. Si alguno tiene una violación de más de $0,2$ se reoptimiza el sistema lineal (LP).
  3. Cortes Framed Capacity (FCI). Si hay algún corte se reoptimiza.
  4. Cortes MultiStar, Comb, Hypotour en llamado cíclico. Si las violaciones respectivas superan $0,05$, $0,1$ o $0,1$ se reoptimiza el LP.
  5. Cortes de Camiones Ordenados y Cortes de Rutas Dirigidas.
  6. Si hay cortes se reoptimiza, si no, se termina la separación del nodo raíz.
- **Root3:**
  1. Cortes de Capacidad. Si alguno tiene una violación de más de $0,2$ se reoptimiza el sistema lineal (LP).
  2. Cortes de Lógica de Ruta 1. Los cortes encontrados se almacenan y se sigue.
  3. Cortes de Camiones Ordenados y Cortes de Rutas Dirigidas. Los cortes encontrados se almacenan y se sigue.
  4. Cortes Framed Capacity (FCI). Si hay algún corte se reoptimiza.

5. Cortes MultiStar, Comb, Hypotour en llamado cíclico. Si las violaciones respectivas superan $0,05$, $0,1$ o $0,1$ se reoptimiza el LP.
6. Si hay cortes se reoptimiza, si no, se termina la separación del nodo raíz.

- **Root4:**

1. Cortes de Capacidad. Si alguno tiene una violación de más de $0,2$ se reoptimiza el sistema lineal (LP).
2. Cortes Framed Capacity (FCI). Si hay algún corte se reoptimiza.
3. Cortes MultiStar, Comb, Hypotour en llamado cíclico. Si las violaciones respectivas superan $0,05$, $0,1$ o $0,1$ se reoptimiza el LP.
4. Si hay cortes se reoptimiza, si no, se termina la separación del nodo raíz.

Las estrategias **Root1-Root3** modifican el orden de llamada de los distintos algoritmos de separación. La estrategia **Root4** elimina la separación de las *desigualdades de Lógica de Ruta 1*, las de *Camiones Ordenados* y *Rutas Dirigidas*. Los resultados obtenidos son mostrados en los cuadros 5.8, 5.9, 5.10 y 5.11. En los resultados mostrados se informa la cota inferior obtenida (LB), utilizando dicha estrategia. Además, se presenta la cota superior obtenida, mediante la heurística primal y las heurísticas de mejoramiento (UB). Por último, se informa el tiempo en segundos utilizados y la cantidad de sistemas lineales optimizados para lograr obtener tales resultados (LPS).

De los resultados mostrados, puede observarse que la estrategia que menores tiempos insume es la **Root4**. Su promedio general de tiempos es muy similar al de la estrategia **Root1**, sin embargo demuestra mejores cotas inferiores en varias instancias que esta última. Las estrategias **Root2** y **Root3** son muy dispares y presentan tiempos de ejecución más elevados en la mayoría de las instancias. A pesar de ello, no mejoran significativamente ninguna cota inferior. Por todo lo expuesto, se ha decidido utilizar la estrategia de separación **Root4** para el nodo raíz del árbol de enumeración.

Es notable destacar la sensibilidad que demuestra el algoritmo a los distintos escenarios. Pequeñas modificaciones pueden provocar grandes variaciones en tiempos de ejecución y en los valores de las cotas inferiores y superiores obtenidos. Esto demuestra lo que se afirmaba al comienzo del capítulo sobre la importancia de la experimentación y el análisis de los resultados empíricos.

instancia	Root1			
	LB	UB	Tiempo	LPS
E-n22-k4	375	375	0,13	22
E-n23-k3	569	569	0,07	12
E-n30-k3	533,3	534	11,42	146
E-n33-k4	834,42	842	7,99	44
E-n51-k5	518,484	538	19,59	50
E-n76-k7	666,279	694	118,79	71
E-n76-k8	717,498	772	139,33	73
E-n76-k10	799,859	867	231,61	77
E-n76-k14	969,428	2456	429,79	97
E-n101-k8	802,271	860	200,05	71
E-n101-k14	1026,61	1114	854,06	145
F-n45-k4	724	724	0,74	28
F-n72-k4	236,354	237	25,05	173
F-n135-k7	1159,72	1201	156,49	63
M-n101-k10	820	820	33,92	74
Promedios			**148,60**	**76,40**

Cuadro 5.8: Resultados utilizando estrategia Root1 en nodo raíz

instancia	Root2			
	LB	UB	Tiempo	LPS
E-n22-k4	375	375	0,16	14
E-n23-k3	569	569	0,06	7
E-n30-k3	533,3	534	13,37	169
E-n33-k4	834,446	839	14,13	51
E-n51-k5	518,111	536	17,61	38
E-n76-k7	666,173	710	116,37	58
E-n76-k8	717,544	762	205,21	76
E-n76-k10	799,979	892	266,98	69
E-n76-k14	969,421	2456	480,77	101
E-n101-k8	802,449	844	306,11	87
E-n101-k14	1026,52	1125	1075	145
F-n45-k4	724	724	1,08	21
F-n72-k4	236,361	239	29,04	380
F-n135-k7	1159,72	1182	224,38	44
M-n101-k10	820	820	89	54
Promedios			**189,28**	**87,60**

Cuadro 5.9: Resultados utilizando estrategia Root2 en nodo raíz

instancia	Root3			
	LB	UB	Tiempo	LPS
E-n22-k4	375	375	0,12	22
E-n23-k3	569	569	0,06	12
E-n30-k3	533,3	534	17,46	196
E-n33-k4	834,433	842	14,04	58
E-n51-k5	518,176	551	15,33	39
E-n76-k7	666,533	706	138,66	80
E-n76-k8	717,595	774	201,3	81
E-n76-k10	799,36	882	254,24	71
E-n76-k14	969,355	2456	495,35	101
E-n101-k8	802,189	853	304,21	120
E-n101-k14	1027,15	1114	1367,23	202
F-n45-k4	724	724	0,24	29
F-n72-k4	236,36	240	59,04	420
F-n135-k7	1159,72	1183	232,24	107
M-n101-k10	820	820	47,76	63
Promedios			**209,82**	**106,73**

Cuadro 5.10: Resultados utilizando estrategia Root3 en nodo raíz

instancia	Root4			
	LB	UB	Tiempo	LPS
E-n22-k4	375	375	0,09	18
E-n23-k3	569	569	0,04	6
E-n30-k3	533,133	534	8,07	173
E-n33-k4	834,434	839	12,24	56
E-n51-k5	518,185	525	19,46	46
E-n76-k7	666,208	700	107,37	73
E-n76-k8	717,496	763	130,98	71
E-n76-k10	799,73	882	216,81	89
E-n76-k14	969,426	2456	366,4	101
E-n101-k8	802,239	858	182,35	84
E-n101-k14	1027,35	1114	960,29	186
F-n45-k4	724	724	0,48	19
F-n72-k4	236,361	240	38,03	439
F-n135-k7	1159,72	1181	142,85	91
M-n101-k10	820	820	22,57	49
Promedios			**147,20**	**100,07**

Cuadro 5.11: Resultados utilizando estrategia Root4 en nodo raíz

5.4. Fijación y seteo de variables por implicaciones lógicas

Una vez seleccionada la estrategia **Root4** como la definitiva para el nodo raíz del árbol, se procederá a analizar la utilidad de fijar y setear variables durante el proceso de separación. El mismo análisis se podría efectuar también realizando un *branch&cut* completo, sin embargo, los resultados mostrados a continuación son más que contundentes sobre la utilidad de las reglas de fijación y seteo implementadas.

En el cuadro 5.12 pueden observarse los resultados de la estrategia **Root4** y la nueva estrategia **Root5**. La definición de la estrategia **Root5** es la misma que la de **Root4** pero sin fijado y seteo de variables mediante implicaciones lógicas. En estos resultados se advierte el incremento significativo de los tiempos de ejecución en varias de las instancias de prueba. Los promedios finales de los tiempos utilizados por ambas estrategias demuestran dicho incremento.

Es importante aclarar que la cantidad de variables fijadas incluye las fijadas por Costos Reducidos. Como se explicó en la sección 4.4, el framework *ABACUS* permite el fijado y el seteado de variables por Costo Reducido. Debido a ello y a la sensibilidad recientemente comentada, hay instancias del escenario **Root5** que han finalizado con un número final de fijaciones mayor al de la estrategia **Root4**. Sin embargo, en todos los casos, el tiempo de ejecución demuestra la utilidad de las implicaciones lógicas utilizadas para la fijación y el seteo de variables.

instancia	Root4		Root5	
	Tiempo	Fix	Tiempo	Fix
E-n22-k4	0,09	372	0,09	388
E-n23-k3	0,04	440	0,04	462
E-n30-k3	8,07	729	12,35	746
E-n33-k4	12,24	643	9,46	651
E-n51-k5	19,46	1832	27,4	1471
E-n76-k7	107,37	1066	116,11	678
E-n76-k8	130,98	250	207,37	0
E-n76-k10	216,81	0	230,1	0
E-n76-k14	366,4	0	420,86	0
E-n101-k8	182,35	42	246,87	6
E-n101-k14	960,29	0	1124,45	0
F-n45-k4	0,48	1809	0,83	1650
F-n72-k4	38,03	3906	36,78	4904
F-n135-k7	142,85	5266	217,85	1517
M-n101-k10	22,57	10187	27,36	10309
Promedios	**147,20**	**1769,47**	**178,53**	**1518,80**

Cuadro 5.12: Resultados utilizando estrategia Root5 en nodo raíz

5.5. Branching de variables vs Branching de conjuntos

Debido a la diferencia abismal que existe entre ambos enfoques, se ha decidido mostrar sólo tres instancias que requieren un branching reducido para alcanzar el óptimo. En las mismas, se necesita aplicar *branching* para obtener el resultado óptimo, pero la diferencia entre la cota inferior obtenida en el nodo raíz y la superior global es reducida. De esta manera, el *branching* está acotado ya que el árbol a generar no puede ser demasiado grande. Los resultados obtenidos pueden observarse en el cuadro 5.13. En la instancia **A-n33-k5** puede observarse que con una cota inferior en el nodo raíz de 659 y una superior de 661 el árbol generado con *branching de variables* es infinitamente superior al generado con *branching de conjuntos*.

instancia	Branching			
	Conjuntos		Variables	
	Tiempo	Nodos	Tiempo	Nodos
E-n51-k5	25,32	3	360,82	157
A-n32-k5	17,75	3	30,97	33
B-n41-k6	16,16	9	459,26	537
Promedios	**19,74**	**5**	**283,68**	**242,33**

Cuadro 5.13: Branching de conjuntos vs. Branching de variables

5.6. Comparación con trabajos previos

En la presente sección se mostrará la ejecución completa del algoritmo de *branch&cut* implementado sobre las instancias de las familias "A", "B", "E", "M" y "F", detalladas al comienzo de este capítulo. Los resultados serán comparados con los obtenidos en el trabajo (LLE04) ("LLE") indicando en todos los casos la cota inferior obtenida en el nodo raíz (LB), el tiempo (en segundos) requerido para obtenerla, la cota inferior obtenida y el tiempo utilizado por el proceso de *branching* (en caso de que fuera necesario), y la cantidad de nodos del árbol de enumeración analizados para abordar a la solución mostrada.

En primer lugar, se mostrarán los resultados obtenidos sobre las instancias presentadas en el cuadro 5.1. En estos resultados, se utiliza para calcular la cota superior (UB), el algoritmo primal incluido en el algoritmo y la heurística de mejoramiento primal aplicada al finalizar cada una de las optimizaciones. La cota superior va disminuyendo a medida que avanza la optimización y puede generar, de esta manera, un árbol de enumeración más grande que el generado si se pasara dicha cota como parámetro del algoritmo. Se ha fijado un tiempo máximo de 8 horas para efectuar la optimización de cada una de las instancias. Los resultados obtenidos pueden observarse en el cuadro 5.14.

	Mejor	Nodo Raíz		Branch&cut		
Instancia	**UB**	**LB**	**Seg.**	**LB**	**Seg.**	**Nodos**
E-n22-k4	375*	**375**	1	-	-	-
E-n23-k3	569*	**569**	1	-	-	-
E-n30-k3	534*	**534**	5	-	-	-
E-n33-k4	835*	835	5	**835**	7	5
E-n51-k5	521*	519	12	**521**	28	7
E-n76-k7	682*	667	115	675	28800	1727
E-n76-k8	735*	718	172	726	28800	1545
E-n76-k10	830	801	263	810	28800	1112
E-n76-k14	1021	970	347	979	28800	1201
E-n101-k8	815*	803	251	807	28800	801
E-n101-k14	1067	1028	873	1037	28800	657
F-n45-k4	724*	**724**	1	-	-	-
F-n72-k4	237*	237	45	**237**	51	3
F-n135-k7	1162*	1160	176	**1162**	2678	105
M-n101-k10	820*	820	18	**820**	35	3

Cuadro 5.14: Resultados obtenidos con cálculo propio de la cota superior (UB)

A continuación, se presentarán los resultados obtenidos al aplicar el algoritmo sobre todas las instancias presentadas. Para calcular dichos resultados se ha pasado la mejor cota

superior conocida como parámetro. El algoritmo generará de esta manera un árbol mucho más reducido. Como se mencionó anteriormente en la sección 4.3.1, la mayoría de los trabajos presentados se basan en las mejores soluciones conocidas para utilizarlas como cotas superiores globales del algoritmo. Por ello, se ha decidido aplicar la misma política para presentar los resultados finales. En los cuadros 5.15, 5.16, 5.17, 5.18 y 5.19 ser observan los resultados obtenidos frente a los obtenidos en "LLE". Se ha fijado un tiempo máximo de 8 horas para efectuar la optimización de cada una de las instancias. Cabe destacar que las cotas inferiores (LB) obtenidas en los nodos raíz de los árboles de enumeración son siempre enteras debido a que las instancias TSPLIB contienen sólo distancias enteras y, por consiguiente, el valor de la función objetivo de una solución válida deberá ser siempre entero. Debido a esto, puede aprovecharse esta situación redondeando siempre las cotas obtenidas a su entero superior.

Instancia	Mejor UB	LLE			Este trabajo				
					Nodo Raíz		Branch&cut		
		LB	Seg.	Nodos	LB	Seg.	LB	Seg.	Nodos
A-n32-k5	784*	**784**	15	3	781	6	**784**	7	3
A-n33-k5	661*	**661**	23	7	660	10	**661**	11	3
A-n33-k6	742*	**742**	38	15	733	2	**742**	8	9
A-n34-k5	778*	**778**	27	8	766	4	**778**	9	7
A-n36-k5	799*	**799**	51	24	791	10	**799**	24	15
A-n37-k5	669*	**669**	25	8	665	7	**669**	10	5
A-n37-k6	949*	**949**	531	304	925	8	**949**	964	347
A-n38-k5	730*	**730**	116	60	718	12	**730**	60	37
A-n39-k5	822*	**822**	138	53	811	56	**822**	343	61
A-n39-k6	831*	**831**	109	57	816	10	**831**	147	51
A-n44-k6	937*	**937**	620	211	921	21	**937**	844	231
A-n45-k6	944*	**944**	157	62	929	12	**944**	182	65
A-n46-k7	914*	**914**	50	3	912	19	**914**	22	3
A-n48-k7	1073*	**1073**	372	113	1055	59	**1073**	516	123
A-n53-k7	1010*	**1010**	363	107	999	44	**1010**	534	103
A-n54-k7	1167*	**1167**	7246	1643	1135	35	1163	28800	2003
A-n55-k9	1073*	**1073**	468	152	1059	30	**1073**	648	117
A-n63-k9	1616	1608	28800	3411	1580	111	1603	28800	2075
A-n63-k10	1314	1304	28800	3911	1267	114	1298	28800	1995
A-n64-k9	1401	1381	28800	3511	1351	122	1373	28800	1943
A-n65-k9	1174*	**1174**	1324	253	1156	63	**1174**	1291	121
A-n69-k9	1159	1147	28800	3189	1116	103	1144	28800	1635
A-n80-k10	1763	1734	28800	1773	1710	244	1729	28800	1017

Cuadro 5.15: Comparación con instancias A

Instancia	Mejor UB	LLE			Este trabajo				
					Nodo Raíz		Branch&cut		
		LB	Seg.	Nodos	LB	Seg.	LB	Seg.	Nodos
B-n31-k5	672*	**672**	8	1	**672**	1	-	-	-
B-n34-k5	788*	**788**	36	19	785	3	**788**	14	17
B-n35-k5	955*	**955**	5	1	**955**	1	-	-	-
B-n38-k6	805*	**805**	37	21	801	3	**805**	8	5
B-n39-k5	549*	**549**	9	1	**549**	1	-	-	-
B-n41-k6	829*	**829**	42	24	827	8	**829**	16	7
B-n43-k6	742*	**742**	125	63	734	8	**742**	102	57
B-n44-k7	909*	**909**	8	1	**909**	2	-	-	-
B-n45-k5	751*	**751**	46	21	748	8	**751**	37	13
B-n45-k6	678*	**678**	299	159	674	19	**678**	155	31
B-n50-k7	741*	**741**	11	1	**741**	2	-	-	-
B-n50-k8	1312*	**1312**	31026	5694	1281	29	1308	28800	3189
B-n51-k7	1032*	**1032**	209	122	1026	18	**1032**	461	127
B-n52-k7	747*	**747**	25	15	746	1	**747**	26	13
B-n56-k7	707*	**707**	46	14	705	18	**707**	52	11
B-n57-k7	1153*	**1153**	441	168	1151	26	**1153**	177	35
B-n57-k9	1598*	**1598**	1366	264	1590	59	**1598**	2761	427
B-n64-k9	861*	**861**	42	13	860	36	**861**	43	3
B-n67-k10	1032*	**1032**	3309	935	1027	46	**1032**	2906	217
B-n78-k10	1221*	**1221**	87408	8641	1205	142	1216	28800	1345

Cuadro 5.16: Comparación con instancias B

Instancia	Mejor UB	LLE			Este trabajo				
					Nodo Raíz		Branch&cut		
		LB	Seg.	Nodos	LB	Seg.	LB	Seg.	Nodos
E-n13-k4	247*	-	-	-	**247**	1	-	-	-
E-n22-k4	375*	**375**	2	1	**375**	1	-	-	-
E-n23-k3	569*	**569**	2	1	**569**	1	-	-	-
E-n30-k3	534*	**534**	14	1	**534**	8	-	-	-
E-n31-k7	379*	-	-	-	378	3	**379**	4	3
E-n33-k4	835*	**835**	16	3	**835**	1	-	-	-
E-n51-k5	521*	519	59	17	519	14	**521**	23	3
E-n76-k7	682*	**682**	118683	8631	667	115	676	28800	1595
E-n76-k8	735*	729	28800	2321	718	171	728	28800	1499
E-n76-k10	830	816	28800	2209	801	262	812	28800	1035
E-n76-k14	1021	986	28800	2081	970	349	983	28800	1049
E-n101-k8	815*	811	28800	1621	803	246	809	28800	681
E-n101-k14	1067	1040	28800	917	1028	867	1040	28800	511

Cuadro 5.17: Comparación con instancias E

Instancia	Mejor UB	LLE			Este trabajo				
					Nodo Raíz		Branch&cut		
		LB	Seg.	Nodos	LB	Seg.	LB	Seg.	Nodos
F-n45-k4	724*	**724**	6	1	**724**	1	-	-	-
F-n72-k4	237*	**237**	38	3	**237**	37	-	-	-
F-n135-k7	1162*	**1162**	3092	265	1160	176	**1162**	2060	41

Cuadro 5.18: Comparación con instancias F

Instancia	Mejor UB	LLE			Este trabajo				
					Nodo Raíz		Branch&cut		
		LB	Seg.	Nodos	LB	Seg.	LB	Seg.	Nodos
M-n101-k10	820*	**820**	33	1	**820**	18	-	-	-
M-n121-k7	1034	-	-	-	1017	381	1021	28800	343

Cuadro 5.19: Comparación con instancias M

En todos los resultados mostrados puede observarse lo destacado anteriormente. Cuando la cantidad de nodos del árbol de enumeración a explorar es grande, los tiempos computacionales obtenidos son muy inferiores a los obtenidos con los modelos de flujo de 2 índices. Cuando la cantidad de nodos a explorar no es tan importante, los resultados se emparejan bastante.

Esta diferencia es esperable ya que a la formulación presentada le cuesta mucho más procesar cada sistema lineal (LP) debido a su gran cantidad de variables.

Si además, se considera el uso de la memoria, las diferencias serán aún mayores debido a que con 512 MB disponibles de memoria RAM, en instancias grandes, se hace necesario el uso del disco rígido con archivos de *swap*. Esto provoca una pérdida de eficiencia aún más notoria si se lo compara con los resultados reportados en "LLE" (LLE04).

Aunque las diferencias existen y son razonables, el algoritmo implementado demuestra ser competitivo y aceptable para que sea utilizado como punto de partida en futuras investigaciones.

Capítulo 6

Conclusiones

En este trabajo se analizaron modelos de flujo conocidos para el CVRP y se presentó un nuevo modelo basado en flujo con particularidades que lo hacen más flexible a la implementación de variantes, que los tradicionales *modelos de flujo de 2 índices* comúnmente utilizados. Se demostró también que el problema CVRP pertenece a la clase de problemas *NP-Hard* y se mencionaron los distintos enfoques utilizados para resolverlo en forma exacta.

Al presentar la nueva formulación, se analizó sus ventajas y desventajas, comparándolo con los *modelos de flujo de 2 y 3 índices*. Se mencionaron las posibles modificaciones y extensiones orientadas a disminuir el número de variables utilizadas por el mismo y se mostró su adaptación a una de las variantes más conocidas de CVRP, denominada CVRP con ventanas de tiempo. Durante la presentación del modelo pudo demostrarse también su correctitud en relación con los problemas tratables por los *modelos de flujo de 2 índices*.

El modelo de flujo presentado utiliza solamente una cantidad de variables del orden $O(n^2)$, frente a la cantidad de $O(n^2K)$ variables requerida por los modelos de flujo de 3 índices tradicionales. Además, se logró demostrar que el modelo propuesto no pierde las ventajas y la flexibilidad de las formulaciones de flujo de 3 índices. Como desventaja, se obtuvo un modelo con un número mayor de restricciones.

Posteriormente, en el capítulo 4 se presentó la implementación de un algoritmo de *branch&cut*, con el objetivo de verificar empíricamente el desenvolvimiento del nuevo modelo y su contrastación con los resultados obtenidos en las publicaciones actuales, basadas en los modelos tradicionales. En la descripción del algoritmo implementado se recorre en detalle todas las etapas relevantes de un algoritmo de *branch&cut* competitivo, como así también se presentan las familias de desigualdades válidas utilizadas como planos de corte. La mayor parte de las desigualdades utilizadas surgen de trabajos publicados y la separación de las mismas se realiza utilizando el paquete CVRPSEP (Lys05) que gentilmente Jens Lysgaard ha programado y cedido a la comunidad científica.

En el capítulo 5, se muestran los resultados obtenidos con el algoritmo de *branch&cut* implementado. Se analizan, mediante la comparación de resultados, las distintas estrategias adoptadas en cada una de las etapas del algoritmo. Se presentan también los incrementos de la cota inferior global, obtenidos con cada una de las familias de desigualdades válidas que se han utilizado. Por último, se comparan los resultados obtenidos con los presentados en el último trabajo publicado de *branch&cut* para el problema CVRP.

Como se mencionó en el capítulo de resultados, el algoritmo implementado mejora los tiempos de las instancias más pequeñas, mientras que dicha situación se revierte en las instancias con mayor número de nodos. En realidad, la diferencia fundamental se encuentra con las instancias donde se hace necesario aplicar *branching* para armar un árbol de enumeración de tamaño importante. Esto suele ocurrir cuando la cota inferior obtenida en el nodo raíz del árbol se encuentra a una distancia importante de la cota superior global. El incremento de la cota inferior se basa en enumerar una cantidad importante de subproblemas, buscando llegar a situaciones donde la misma pueda ser incrementada en forma segura. Debido a que el modelo presentado contiene una cantidad de variables y de restricciones muy superior a las utilizadas en los *modelos de flujo de 2 índices*, en el que se basan la mayoría de las publicaciones existentes sobre CVRP, la ejecución reiterada de optimizaciones tiende a provocar una brecha importante entre los tiempos de ejecución obtenidos con el presente modelo y los obtenidos en los trabajos referidos.

Es sabido que los optimizadores lineales disminuyen fuertemente su rendimiento a medida que el número de variables y restricciones aumenta. Sin embargo, considerando que el modelo presentado ofrece la información adicional de conocer el número de vehículo que atraviesa cada nodo, los resultados obtenidos pueden considerarse satisfactorios.

Para comparar de manera ecuánime los resultados obtenidos por la presente implementación, hubiese sido ideal contar con resultados de *branch&cut* aplicados a *modelos de flujo de 3 índices*. Sin embargo, no se encontraron trabajos publicados que se enfoquen hacia dicha dirección.

6.1. Trabajos futuros

El estudio realizado constituye una primera aproximación al modelo de flujo presentado y abre un amplio camino de enfoques viables para intentar mejorar los resultados computacionales obtenidos. Entre las tareas más importantes se encuentra el estudio poliedral del modelo para la identificación de nuevas familias de desigualdades que saquen especial

provecho de la estructura que lo caracteriza.

Es importante realizar un análisis detallado de la interrelación existente entre los multiejes incluidos en los *nodos cliente* y los ejes que interconectan *nodos cliente*. El mismo análisis cabe aplicarlo a la interconexión que existe entre los *nodos vehículo* y los *nodos cliente*, y a la variación de los valores asociados a los multiejes involucrados.

Al analizar el avance de las optimizaciones, se pudo vislumbrar una cierta dificultad para obtener soluciones con multiejes enteros. En general, se obtienen valores fraccionales que muestran un intento de repartir la demanda del cliente entra varios vehículos. A medida que las desigualdades de capacidad y el resto de las ofrecidas en CVRPSEP se agregan al sistema, los valores de los multiejes van tendiendo a hacerse enteros (binarios 0-1). Se considera una tarea pendiente poder vislumbrar desigualdades válidas que mejoren la integralidad de los multiejes, reforzando la relación entre los clientes y las rutas. Lograr resultados en dicha dirección permitiría la utilización de las *desigualdades de eliminación de subtours* presentadas inicialmente en el modelo y el descarte de las *desigualdades de capacidad* finalmente utilizadas. Este cambio podría ofrecer importantes ventajas gracias al conocimiento obtenido proporcionado por los estudios realizados para el problema del viajante de comercio (TSP).

Otra de las tareas pendientes, es la implementación de una buena heurística primal que permita encontrar una cota inicial ajustada y completar, de esa manera, la implementación de un *branch&cut* mucho más robusto, general e independiente de los resultados conocidos por trabajos publicados.

Otro de los problemas observados, es la utilización de una cantidad elevada de variables que limita la aplicación del presente modelo sólo a instancias de tamaño pequeño y tamaño mediano. Además, como pudo observarse en los resultados computaciones presentados, el aumento en el tamaño de las instancias provocan una abrupta caída de la velocidad de resolución, limitando también la resolución de instancias difíciles que requieran, para su resolución, una exploración intensiva del árbol de soluciones. Es por todo esto que surge la premisa natural de intentar reducir la cantidad de variables utilizadas.

Dentro de las ideas analizadas para poder mitigar los problemas anteriores, surge la posibilidad de implementar, en trabajos futuros, un algoritmo de *branch&cut&price* que permita trabajar con un número reducido de variables activas. Este algoritmo, incorpora las variables necesarias utilizando una etapa de identificación que selecciona las candidatas a ingresar al sistema lineal. Este proceso, se denomina comúnmente en la literatura como etapa de *pricing*. Se cree que en este sentido puede llegarse a obtener importantes mejoras de rendimiento ya que la mayoría de las variables multiejes permanecen inactivas durante la mayor parte de la optimización.

La segunda idea consiste en aplicar un algoritmo de *branch&cut* tradicional sobre un modelo de flujo de 2 índices, obtener la solución óptima con dicho modelo y luego armar una solución arbitraria inicial para el modelo propuesto, basándose en la optimización efectuada en el primer paso. Esta alternativa también puede llegar a dar buenos frutos debido a que optimiza, en primer lugar, la asignación óptima de rutas y, luego, incorpora la información relativa al vehículo que recorre cada cliente. Esta alternativa tendría sentido en situaciones donde se necesite la información del número de vehículo asociado a cada nodo, para poder agregar restricciones adicionales. Un caso posible es el que fue mostrado para el problema de ruteo de vehículos con ventanas de tiempo.

Es importante destacar que, hasta el momento, no se le ha dedicado al estudio del problema CVRP un esfuerzo equiparable al aplicado para el del viajante de comercio (TSP). Si bien se considera a CVRP como un problema más difícil que TSP, es importante dedicar mayores esfuerzos tanto a estudios teóricos orientados al análisis de los poliedros asociados, la identificación de desigualdades válidas y facetas, como también a generar nuevos modelos que permitan obtener diferentes enfoques del mismo problema.

Los algoritmos de *branch&cut* han demostrado ser muy efectivos para la resolución exacta de problemas combinatorios difíciles. Si bien el tamaño de las instancias analizadas para el problema CVRP sigue siendo reducido, si se lo compara con el tamaño de los problemas aplicados que surgen en el ámbito de la industria y el comercio, la investigación en dicha dirección demuestra ser uno de los mejores enfoques para tratar su resolución con métodos exactos. El presente trabajo intenta aportar, mediante el modelo expuesto, una herramienta basada en programación lineal para resolver el problema en forma exacta.

Apéndice A

Detalles de Implementación

En este anexo se describirán brevemente los detalles asociados a la implementación del presente algoritmo de *branch&cut*. En primer lugar, se expondrá el entorno de desarrollo utilizado para la implementación y, posteriormente, se pasará a relevar los distintos detalles relativos al algoritmo.

A.1. El framework ABACUS

El *framework* ABACUS (A Branch-And-CUt System), creado por Stefan Thienel como parte de su Tesis Doctoral (Thi95), presenta un entorno de desarrollo especializado para la implementación de algoritmos de *branch&cut*, *branch&bound* y *branch&cut&price*. El mismo ha sido diseñado bajo el paradigma de la programación orientada a objetos y fue concebido para ser utilizado bajo el lenguaje de programación C++. Este entorno posee un refinado diseño de clases especialmente ideadas para satisfacer todas las necesidades que un algoritmo como los enumerados pueda llegar a requerir. Utilizando los conceptos de herencia y de redefinición, propios del paradigma de objetos, se puede especializar el *framework* para lograr utilizarlo en cualquier clase de problemas combinatorios tratables con métodos de programación lineal y programación lineal entera.

Entre los objetivos principales que han sido tenidos en cuenta para el diseño del entorno ABACUS se pueden mencionar la *Eficiencia*, *Flexibilidad*, *Facilidad de Uso*, *Extensibilidad* y *Portabilidad*.

La *Eficiencia* se encuentra especialmente cuidada logrando un *overhead* computacional máximo que no supera el 10 % del obtenido mediante algoritmos específicos implementados

fuera de entornos como ABACUS (Thi95).

La *flexibilidad* ha sido lograda utilizando, por un lado, un excelente diseño de clases y de funciones abstractas que permiten a los usuarios del *framework* introducir comportamientos específicos para cada una de las etapas y momentos de la optimización. Por otro parte, la utilización de un conjunto variado de parámetros permite especificar una amplia gama de comportamientos standard que pueden requerirse en los problemas a tratar.

La *facilidad de uso* ha sido reforzada mediante una excelente documentación. Además, se ha prestado especial atención en la elección de los nombres de las clases y las funciones de manera de ser auto-explicativas y permitir un código claro y conciso.

Existen diversos proyectos para extender el framework en entornos de multiprocesamiento, como así también un permanente mantenimiento que permite adaptar el entorno a las nuevas versiones de compiladores y optimizadores lineales que aparecen continuamente en el mercado.

Hace ya unos años que la licencia de ABACUS se ha convertido en *GNU Lesser General Public License*, convirtiéndose, de esta manera, en un software de código abierto (Open Source). Esta particularidad es otra característica que asegura la *extensibilidad* y la *portabilidad* del mismo.

Por último, cabe mencionar que la *portabilidad* ha sido lograda permitiendo que el *framework* interactúe con los optimizadores lineales *CPLEX*, *XPRESS* y *SOPLEX*. Además, existen versiones para utilizarlo en Sistemas Operativos Windows, como así también en entornos basados en Sistemas Operativos Unix y Linux.

A.2. La estructura de clases de ABACUS

Dentro de la amplia variedad de clases que provee el *framework*, existe un conjunto de clases destacadas que deben ser incluidas en toda implementación que lo utilice. Dichas clases proveen la funcionalidad básica que permite concretar un algorítmo de optimización.

Se explicarán a continuación las clases fundamentales provistas por ABACUS:

Clase *ABA_MASTER*

Esta clase controla el proceso de la optimización. Dentro de esta clase se inicializa el

árbol de enumeración, se generan las variables y las restricciones de la formulación inicial, se crean los *pooles* de cortes y se instancia el primer nodo del árbol de enumeración (nodo raíz). El usuario tiene además, la posiblidad de implementar en esta clase, heurísticas primales, realizar la lectura de los archivos con las instancias a optimizar, leer archivos con parámetros adicionales, etc. El entorno ABACUS lee además los parámetros generales especificados en el archivo **.abacus**. El usuario puede modificar el archivo de parámetros o puede setearlos mediante código implementado dentro de la clase *ABA_MASTER*.

Clase *ABA_SUB*

El *framework* proporciona esta clase para representar a los distintos subproblemas que conformarán el árbol de enumeración. Se instancia un objeto de esta clase por cada nodo del árbol creado. Hay un constructor particular para el nodo raíz del árbol y otro para el resto de los nodos. En el caso de este último, el constructor pasa como parámetro un puntero a su nodo padre.

Dentro de esta clase, se implementan las funcionalidades principales de los algoritmos de *branch&cut*. Existen funciones específicas para poder implementar la separación de desigualdades válidas, el fijado y el seteo de variables, la ejecución de heurísticas de mejoramiento de las soluciones encontradas, la generación de reglas de branching, la verificación de la validez de una solución, criterios especiales para podar un nodo por infactibilidad, pricing de variables (en el caso en que se necesita realizar dicho proceso), etc.

Dependiendo de las funciones que se redefinan en esta clase, se determinará el funcionamiento general que adoptará el algoritmo de optimización implementado.

Clase *ABA_VARIABLE*

Cada una de las variables que se utilizan en la optimización deben ser instancias de la clase *ABA_VARIABLE*. En realidad, dicha clase esta pensada para ser heredada y programada de acuerdo a la funcionalidad propia de cada aplicación. La manera en que se ha diseñado esta clase permite generar variables de cualquier tipo y forma. Además de seleccionar el tipo de valor que la variable contendrá (enteras, binarias, reales, etc), puede determinarse la forma con que la misma será referida dentro del algoritmo (con 1 índice, con 2 índices, con 3 índices, etc).

Clase *ABA_CONSTRAINT*

Esta clase fue diseñada con el objetivo de representar las restricciones del problema y los cortes que se agregarán durante el proceso de separación. Cada una de las restricciones utilizadas debe ser una instancia derivada de esta clase. El usuario debe implementar cla-

ses propias para personalizar el tipo de restricciones que el problema a tratar requiera. El *framework* ha logrado diseñar, mediante esta clase, un mecanismo sumamente flexible que permite la adaptación del entorno a cualquier tipo de problema de optimización.

El mecanismo principal que ofrece esta clase es la posibilidad de redefinir la función **coeff(ABA_VARIABLE *v)**. Esta función toma como parámetro una variable y debe devolver el coeficiente que la misma tendrá asociado en la representación de la restricción instanciada. De esta manera, el entorno construye, con total flexibilidad para el usuario, las distintas restricciones que conformarán las formulaciones de los problemas a optimizar.

El entorno provee además un conjunto muy amplio de clases para implementar un sinfín de comportamientos y procesos. Puede obtenerse una visión completa de las distintas características del *framework* y su utilización en (Thi95; Thi96; JT97a; JT97b; J99).

A.3. Descripción de la implementación

La implementación del algoritmo sigue los lineamientos sugeridos en los manuales del *framework* ABACUS. Muchas de las ideas y de las decisiones adoptadas se encuentran basadas en la implemetación del problema TSP, realizada como ejemplo de uso del *framework* en la Tesis Doctoral de Stefan Thienel. El análisis del código implementado para dicho problema sirvió como guía principal a la hora de decidir la implementación de muchas de las clases utilizadas.

A.3.1. Lectura de las instancias de prueba

La primera de las clases implementada fue la clase *TSPLIB*. Esta clase es básicamente un *parser* que, bajo los lineamientos de la especificación TSPLIB95 (Rei95), realiza la lectura de las distintas instancias que pueden ser optimizadas para CVRP. Esta clase permite el procesamiento de todos los formatos posibles que figuran en TSPLIB95.

A.3.2. Variables del modelo

El modelo propuesto se encuentra basado en los modelos de flujo en redes. Tales modelos se representan, en general, utilizando redes de flujo implementadas sobre grafos. Utilizando

las ideas observadas de la implementación de TSP comentada anteriormente, se decidió heredar la clase *ABA_VARIABLE* en una nueva clase denominada *GVARIABLE*. Esta nueva clase representa variables asociadas a ejes de un grafo. La principal diferencia con la implementación original de *ABA_VARIABLE* es el agregado de la información concerniente a los nodos extremos del eje en el grafo asociado. Por último, esta clase fue nuevamente heredada para construir dos nuevas clases: *GVARIABLE2* y *GVARIABLE3*. La clase *GVARIABLE2* representa las variables de 2 índices (ejes con un nodo que representa la del eje cabeza y otro la cola del mismo). La clase *GVARIABLE3* implementa, por su parte, las variables de 3 índices (multiejes representados por 2 nodos y un número de multieje).

A su vez, cada una de estas variables implementa una función que permite preguntar qué tipo de variable es (2 o 3 índices). El esquema de herencias mencionado puede visualizarse en la figura A.1.

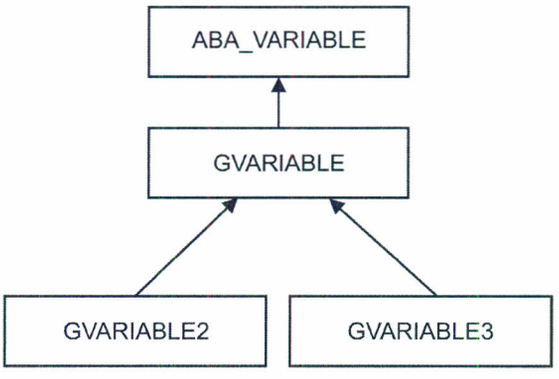

Figura A.1: Jerarquía de clases de variables

A.3.3. Restricciones del modelo

De la misma manera que se decidió la creación de una clase específica para representar variables asociadas a ejes de un grafo, fue necesario implementar clases particulares para el tratamiento de restricciones con variables del tipo *GVARIABLE*. La clase implementada recibió el nombre de *GCONSTRAINT* y tiene como principal diferencia frente a la clase *ABA_CONSTRAINT*, la manera en que se implementa la función **coeff(ABA_VARIABLE *v)**. Esta nueva clase, que hereda de *ABA_CONSTRAINT*, incluye dos nuevas funciones: **coeff(in t, int h)** y **coeff(in t, int h, int m)**. La primera

recibe como parámetro el número de los dos nodos extremos del eje representado. La segunda, recibe además el número de multieje que se asocia al eje en cuestión. La función de la primera es recibir variables de 2 índices y la de la segunda es recibir variables de 3 índices. Dependiendo de la forma que se quiera dar a la restricción instanciada, se implementará de manera adecuada cada una de las funciones. A partir de la clase GCONSTRAINT, se crearon nuevas clases heredadas específicas para cada uno de los tipos de restricciones utilizados en el modelo. La lista de clases creadas para cada una de las desigualdades se detalla a continuación:

- **CVFLUJO**: para las *Restricciones de conservación de flujo.*
- **CVCAPACIDAD**: para las *Restricciones de control de capacidad máxima.*
- **CVLOGICARUTA1**: para las *Restricciones de Lógica de Ruta 1.*
- **CVLOGICARUTA2**: para las *Restricciones de Lógica de Ruta 2.*
- **CVCAMIONESORDENADOS**: para las *Restricciones de Camiones Ordenados.*
- **CVCAMIONESRUTASDIRIGIDAS**: para las *Restricciones de Rutas Dirigidas.*
- **CVSUBTOUR1**: para las *Restricciones de Capacidad* que tiene la forma 1 mostrada en la sección 4.1.1.
- **CVSUBTOUR2**: para las *Restricciones de Capacidad* que tiene la forma 2 mostrada en la sección 4.1.1.
- **CVSUBTOUR3**: para las *Restricciones de Capacidad* que tiene la forma 3 mostrada en la sección 4.1.1.
- **CVMULTISTAR1**: para las *Restricciones Multistar y Partial MultiStar* con la forma 1 mostrada en la sección 4.1.7.
- **CVMULTISTAR2**: para las *Restricciones Multistar y Partial MultiStar* con la forma 2 mostrada en la sección 4.1.7.
- **CVGLMULTISTAR**: para las *Restricciones Generalized Large Multistar.*
- **CVFRAMEDCAPACITY**: para las *Restricciones Framed Capacity.*
- **CVCOMB**: para las *Restricciones Strengthened Comb.*
- **CVHYPOTOUR**: para las *Restricciones 2-edges extended Hypotour.*
- **BRREGLAIZQ**: para establecer la regla de *branching* $X(\delta(S)) = 2r(S)$.
- **BRREGLADER**: para establecer la regla de *branching* $X(\delta(S)) \geq 2r(S) + 2$.

A.3.4. Clase Master

En primer lugar, se implementó una clase heredada de *ABA_MASTER* específica para el manejo de problemas de optimización con grafos. Esta nueva clase recibió el nombre de *GRAPHMASTER*. Utilizando la clase especializada se creó una clase específica para tratar el problema de ruteo de vehículos con capacidades, que se denominó *CVRP*. Aquí se especializó el comportamiento global del algoritmo para adecuarlo al problema a tratar, utilizando el *branch&cut* explicado.

A.3.5. Clase Subproblema

Mediante la clase *ABA_SUB* se implementó la clase heredada *SUBCVRP*. Esta clase contiene el manejo de la fase de separación, de las heurísticas de mejoramiento, fijado y seteo de variables, branching, etc. Todos estas etapas fueron programadas especialmente para el problema CVRP abordado.

A.3.6. Parámetros específicos

Además de los parámetros que ofrece ABACUS mediante el archivo **.abacus**, se implementó la posibilidad de setear determinadas configuraciones propias del problema CVRP. Dichas configuraciones deben especificarse en el archivo **.cvrp** incluido dentro del directorio donde se encuentra el programa ejecutable.

A continuación, se describen los distintos parámetros y la utilidad asociada a los mismos:

ObjInteger: Permite establecer si puede suponerse que la optimización será efectuada con distancias enteras. En TSPLIB95 se define que todos los problemas deben cumplir esta condición.

ShowBestTour: Determina si al finalizar la optimización se debe o no mostrar la mejor solución encontrada.

LogicalFixOrSet: Mediante este parámetro se informa al algoritmo si puede o no utilizar el seteo y la fijación de variables mediante implicaciones lógicas.

CortesRutasDirigidas, CortesCamionesOrdenados, CortesLogicadeRutas, CortesMultiStar, CortesGlMultiStar, CortesFCI, CortesComb y **CortesHypotour :** Mediante estos parámetros se puede habilitar o deshabilitar la utilización de cada uno de

los cortes enunciados.

SetBranching: Utilizando este parámetro se puede elegir entre hacer branching por conjuntos o branching por variables.

Todos los parámetros mostrados permiten como valores válidos *true* y *false*.

A.3.7. Clase Branching

En esta clase se implementó la heurísitica de identificación de conjuntos, candidatos para realizar *branching* sobre conjuntos.

Bibliografía

[ABB+95] P. Augerat, J. M. Belenguer, E. Benavent, A. Corberan, D. Naddef, and G. Rinaldi. *Computational results with a Branch and Cut Code for the Capacitated Vehicle Routing Problem.* Research Report 949-M, Universite Joseph Fourier, Grenoble, France, 1995. 4.1.1, 4.1.1, 4.1.6, 4.1.9, 4.3.1, 5.2

[ABB+98] P. Augerat, J. M. Belenguer, E. Benavent, A. Corberan, and D. Naddef. *Separating capacity constraints in the CVRP using tabu search.* European Journal of Operations Research 106, 546-557, 1998. 4.1.1, 4.1.1, 5.2

[ACH98] N. R. Achuthan, L. Caccetta, and S. P. Hill. *Capacitated vehicle routing problem: some new cutting planes.* Operations Research 17, Asia-Pac., 109-123, 1998. 4, 4.1.8

[ACH03] N. R. Achuthan, L. Caccetta, and S. P. Hill. *An Improved Branch-and-Cut Algorithm for the Capacitated Vehicle Routing Problem.* Transportation Science 37, 153-169, 2003. 4.1.1, 4.3.1

[AHM90] J. R. Araque, L. A. Hall, and T. L. Magnanti. *Capacitated trees, capacitated routing and associated polyhedra.* Discussion paper, Center of Operations Research and Econometrics, Catholic University of Louvain, Belgium, 1990. 4, 4.1.7

[AMO93] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, United States, 1993. 2.2.1

[Ara90] J. R. Araque. *Lots of combs of different sizes for vehicle routing.* Discussion paper, Center of Operations Research and Econometrics, Catholic University of Louvain, Belgium, 1990. 4, 4.1.6

[Aug95] P. Augerat. *Approche poly'edrale du probléme de tournées de véhicles.* PhD thesis, Polytechnique de Grenoble, 1995. 4, 4.1.5, 4.1.9, 4.2, 4.2, 5.1

[BH02] U. Blasum and W. Hochstättler. *Application of the branch and Cut Method to the Vehicle Routing Problem.* Technical Report ZPR2000-386, Universität zu Köln, 2002. 4.1.1, 4.1.6, 4.3.1, 5.2

[BHM04] R. Baldacci, E. Hadjiconstantinou, and A. Mingozzi. *An Exact Algorithm for the Capacitated Vehicle Rounting Problem Based on a Two-Commodity Network Flow Formulation.* Operations Research 52, 723-738, 2004. 4.3.1

[BMMN95] M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser. *Network Routing.* Handbooks on Operations Research and Management Science 8, Amsterdam: Elsevier, 1995. 4.1.9

[BQ64] M. Balinski and R. Quandt. *On an integer program for a delivery problem.* Operations Research, 12:300-304, 1964. 2.2.3

[CCM91] V. Campos, A. Corberán, and E. Mota. *Polyhedral results for a vehicle routing problem.* European journal of operational research 52, 7585, 1991. 4

[CE69] N. Christofides and S. Eilon. *An algoritm for the vehicle dispatching problem.* Operations Research Quarterly Vol. 20, 309-318, 1969. 5.1

[CH93] G. Cornuejols and F. Harche. *Polyhedral study of the capacitated vehicle routing problem.* Mathematical Programming 60, North Holland, 21-52, 1993. 4, 4.1.1

[CMT79] N. Christofides, A. Mingozzi, and P. Toth. *The Vehicle Routing Problem. In Combinatorial Optimization.* John Wiley & Sons, New York, 1979. 5.1

[CW64] G. Clarke and J. V. Wright. *Scheduling of vehicles from a central depot to a number of delivery points.* Operations Research 12, 568-581, 1964. 2.1, 4.3.1

[DR59] G. B. Dantzig and R. H. Ramser. *The Truck Dispatching Problem.* Management Science 6, 80-91, 1959. 2.1

[FGT95] M. Fischetti, J. J. Salazar Gonzalez, and P. Toth. *Experiments with a multi-commodity formulation for the symmetric capacitated vehicle routing problem.* Proceedings of the 3rd Meeting of the EURO Working Group on Transportation, 169-173, Barcelona, 1995. 2.2.1.2

[Fis94] M. Fisher. *Optimal solution of vehicle routing problem using minimum K-Trees.* Operations Research 42, 626-642, 1994. 5.1

[Fis95] M. Fisher. *Vehicle routing.* Handbooks of Operations Research and Management Science, chapter 1, 8:1-31, 1995. 2.2

[FJ81] M.L. Fisher and R. Jaikumar. *A generalized assignment heuristic for the vehicle routing problem.* Networks 11, 109-124, 1981. 2.2.1.2

[FLdAa+06] R. Fukasawa, J. Lysgaard, M. Poggi de Aragão, M. Reis, E. Uchoa, and R. F. Werneck. *Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem.* Mathematical Programming: Series A and B Vol.106-3, 491-511, 2006. 4.3.1, 5.2

[GCJS57] W. M. Garvin, H. W. Crandall, J. B. John, and R. A. Spellman. *Applications of linear programming in the oil industry.* Management Science, 3:407-430, 1957. 2.2.2

[GG79] B. Gavish and S. Graves. *The travelling salesman problem and related problems.* Working paper 7905, Graduate School of Management, University of Rochester, Rochester, NY, 1979. 2.2.2

[GG82] B. Gavish and S. Graves. *Scheduling and routing in transportation and distributions systems: Formulations and new relaxations.* Working paper, Graduate School of Management, University of Rochester, Rochester, NY, 1982. 2.2.2

[GHL94] M. Gendreau, A. Hertz, and G. Laporte. *A tabu search for the vehicle routing problem.* Management Science 40, 1276-1290, 1994. 4.3.1

[GJR84] M. Grötschel, M. Jünger, and G. Reinelt. *A cutting plane algorithm for the linear ordering problem.* Operations Research 32, 1194-1220, 1984. 1.4

[Gou94] L. Gouveia. *A result on projection for the vehicle routing problem.* European journal of operational research 85, 610-624, 1994. 4.1.8

[GP79a] M. Grötschel and M. W. Padberg. *On the symmetric travelling salesman problem I: inequalities.* Mathematical Programming 16, 265-280, 1979. 4.1.6

[GP79b] M. Grötschel and M. W. Padberg. *On the symmetric travelling salesman problem II: lifting theorems and facets.* Mathematical Programming 16, 281-302, 1979. 4.1.6

[Inc94] CPLEX Optimization Inc. *Using the CPLEX Callable Library, including Using the CPLEX Linea Optimizer with CPLEX Barrier and Mixed Integer Solvers.* Version 3.0, 1994. 1

[JÖ99] M. Jünger. *ABACUS 2.3 User's Guide and Reference Manual.* OREAS, 1999. 1, 4.4, 4.5.4, A.2

[JNS98] E. Johnson, G. Nemhauser, and M. Savelsbergh. *Progress in Linear Programming Based Branch-and-Bound Algorithms: An Exposition.* Technical Report, Georgia Institute of Technology, 1998. 1.4

[JRT94] M. Jünger, G. Reinelt, and S. Thienel. *Practical Problem Solving with Cutting Plane Algorithms in Combinatorial Optimization.* Interdisziplinares Zentrum für Wissenschaftliches Rechnen, Universität Heidelberg, Preprint 94-24, 1994. 1, 1.4

[JT97a] M. Jünger and S. Thienel. *The Design of the Branch and Cut System ABACUS.* Universität zu Köln, Technical Report 97.260, 1997. 1, A.2

[JT97b] M. Jünger and S. Thienel. *Introduction to ABACUS - A Branch and Cut System.* Universität zu Köln, Technical Report 97.263, 1997. 1, A.2

[LEL02] A.Ñ. Letchford, R. W. Eglese, and Jens Lysgaard. *Multistars, partial multistars and the capacitated vehicle routing problem.* Mathematical Programming 94, Springer, 21-40, 2002. 4, 4.1, 4.1.7, 4.1.8

[LLE04] J. Lysgaard, A.Ñ. Letchford, and R. W. Eglese. *A New Branch-and-Cut Algorithm for Capacitated Vehicle Routing Problems.* Mathematical Programming Vol.100, 423-445, 2004. 4.1, 4.1.1, 4.1.1, 4.1.1, 4.1.5, 4.1.6, 4.1.7, 4.1.9, 4.1.10, 4.3.1, 5.2, 5.2, 5.2, 5.2, 5.6, 5.6

[LN83] G. Laporte and Y. Nobert. *A branch and bound algorithm for the capacitated vehicle routing problem.* OR Spektrum 5, 77-85, 1983. 2.2.1.1

[LN84] G. Laporte and Y. Nobert. *Comb inequalities for the vehicle routing problem.* Methods of Operations Research 51, 271-276, 1984. 4, 4.1.6

[LN87] G. Laporte and Y. Nobert. *Exact algorithms for the vehicle routing problem.* Ann. Discr. Math. 31, 147-184, 1987. 2.2.1.1

[Lys05] J. Lysgaard. *CVRPSEP: A package of separation routines for the Capacitated Vehicle Routing Problem.* Department of Management Science and Logistics, Aarhus School of Bussiness, Denmark, http://www.hha.dk/ lys/CVRPSEP.htm, 2005. 4.1, 4.1.1, 4.1.1, 4.1.5, 4.1.7, 6

[Mil76] P. Miliotis. *Integer programming approaches for the traveling salesman problem.* Mathematical Programing 10, 367-378, 1976. 1.4

[MT90] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations.* Chichester, Wiley, 1990. 4.1.5

[NW88] G. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Wiley, Chichester, UK, 1988. 4

[Poc] Y. Pochet. *New valid inequalities for the vehicle routing problem.* In preparation. 4.1.5

[PR87] M.W. Padberd and G. Rinaldi. *Optimization of a 532 city symmetric traveling salesman problem by branch-and-cut.* Operations Research Letters 6, 1-7, 1987. 1.4, 1.4

[PR91] M.W. Padberd and G. Rinaldi. *A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems.* SIAM Review 33, 60-100, 1991. 1.4, 1.4

[Ral95] Ted Ralphs. *www.branchandcut.org.* Department of Industrial and Systems Engineering, Lehigh University, http://branchandcut.org/VRP/data/, 1995. 1, 5, 5.1

[Rei95] Gerhard Reinelt. *TSPLIB95.* Universität Heidelberg, http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsplib.html, 1995. 1, 5, A.3.1

[Thi95] Stefan Thienel. *ABACUS, A Branch-And-CUt System.* Tesis Doctoral, Universität zu Köln, 1995. 1, 4.4, A.1, A.2

[Thi96] Stefan Thienel. *A Simple TSP-Solver: An ABACUS Tutorial.* Universität zu Köln, Technical Report 96.245, 1996. 1, A.2

[TV01] P. Toth and D. Vigo. *The Vehicle Routing Problem.* Society for Industrial and Applied Mathematics (SIAM), 2001. 2.1, 2.2, 2.2.1.2, 2.2.2, 2.2.2, 2.2.3, 3.1.6, 4, 4.1.5, 4.2, 5.2

[TV03] P. Toth and D. Vigo. *The Granular Tabu Search and its Application to the Vehicle-Routing Problem.* INFORMS Journal on Computing V.15 4, 333-346, 2003. 4.3.1