Biblioteca Digital
F C E N - U B A

BIBLIOTECA CENTRAL LUIS F LELOIR
BIBLIOTECA CENTRAL LELOIR
FACULTAD DE CIENCIAS EXACTAS Y NATURALES UBA

Tesis Doctoral

Algoritmos para el problema de localización y ruteo de vehículos con capacidades y premios

Negrotto, Daniel

2015-09-24

Este documento forma parte de la colección de tesis doctorales y de maestría de la Biblioteca Central Dr. Luis Federico Leloir, disponible en digital.bl.fcen.uba.ar. Su utilización debe ser acompañada por la cita bibliográfica con reconocimiento de la fuente.

This document is part of the doctoral theses collection of the Central Library Dr. Luis Federico Leloir, available in digital.bl.fcen.uba.ar. It should be used accompanied by the corresponding citation acknowledging the source.

Cita tipo APA:
Negrotto, Daniel. (2015-09-24). Algoritmos para el problema de localización y ruteo de vehículos con capacidades y premios. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires.

Cita tipo Chicago:
Negrotto, Daniel. "Algoritmos para el problema de localización y ruteo de vehículos con capacidades y premios". Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. 2015-09-24.

EXACTAS UBA
Facultad de Ciencias Exactas y Naturales

UBA
Universidad de Buenos Aires

Dirección: Biblioteca Central Dr. Luis F. Leloir, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires. Intendente Güiraldes 2160 - C1428EGA - Tel. (++54 +11) 4789-9293

Contacto: digital@bl.fcen.uba.ar

UNIVERSIDAD DE BUENOS AIRES
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Algoritmos para el problema de localización y ruteo de vehículos con capacidades y premios

Tesis presentada para optar al título de
Doctor de la Universidad de Buenos Aires
en el área Ciencias de la Computación

Daniel Negrotto

Directora de tesis: Dra. Irene Loiseau
Consejera de estudios: Dra. Irene Loiseau

Septiembre de 2015

Algoritmos para el problema de localización y ruteo de vehículos con capacidades y premios

El problema de *Localización y Ruteo de Vehículos con Capacidades* (CLRP) es la combinación de dos problemas muy estudiados del área de la Investigación Operativa: el problema de localización de depósitos con capacidades (CFLP) y el problema de ruteo de vehículos con múltiples depósitos (MDVRP). Dado un conjunto de posibles localizaciones se busca determinar cuáles utilizar para satisfacer las demandas de un conjunto de clientes y programar las rutas que los visitan. Se busca minimizar los costos de apertura de depósitos, de utilización de vehículos y de ruteo satisfaciendo restricciones de capacidad tanto en los vehículos como en los depósitos.

En este trabajo se presenta una nueva versión del problema denominada *Localización y Ruteo de Vehículos con Capacidades y Premios* (PC-CLRP) que busca generalizar el problema CLRP permitiendo la posibilidad de que los clientes sean o no visitados. Los clientes atendidos otorgan un beneficio y la maximización de la suma de los beneficios forma parte del objetivo del nuevo problema.

Se proponen en este trabajo algoritmos para el problema PC-CLRP. En primer lugar se introduce un algoritmo metaheurístico para resolverlo basado en el método de optimización por Colonia de Hormigas. Se implementa una metaheurística de 3 colonias de hormigas que colaboran construyendo las distintas etapas de una solución PC-CLRP: localización, clusterizado y ruteo. Posteriormente, se presentan modelos de programación lineal entera basadas en modelos de flujo de 2 índices y 3 índices. Se analizan distintas familias de desigualdades válidas utilizadas para CLRP y se proponen nuevas versiones de las mismas para el problema PC-CLRP. Además, se definen nuevas desigualdades válidas y cortes de optimalidad junto a sus correspondientes algoritmos de separación. Por último, se implementa un algoritmo *Branch&Cut* utilizando uno de los modelos de programación lineal entera propuestos. Se reportan los resultados obtenidos por ambos algoritmos para el problema PC-CLRP sobre un conjunto de instancias especialmente diseñadas para el nuevo problema. Se compara además los resultados frente a los reportados en otros trabajos sobre el problema CLRP obteniendo resultados competitivos.

Palabras claves: problema de localización y ruteo de vehículos, programación lineal entera, branch and cut, colonia de hormigas, optimización combinatoria, recolección de premios.

Algorithms for the prize-collecting capacitated location routing problem

The Capacitated Location Routing Problem (CLRP) is the combination of two well studied problems in Operations Research: Capacitated Facility Location Problem (CFLP) and Multiple Depots Vehicles Routing Problem (MDVRP). Given a set of available locations, the aim is to find a subset of depots to satisfy customer demands and to determine the routes to visit the customers. The objective is to minimize the cost of the opened depots, the fixed cost of the vehicles in use and the cost of the routes satisfying vehicle and depot capacity constraints.

In this work we present a new version of the problem named Prize-Collecting Capacitated Location Routing Problem (PC-CLRP). This problem is a generalization of CLRP where customers have not the requirement to be visited. A customer gives a benefit if it is considered as part of the proposed solution. The maximization of this benefits is a new objective for this problem. We proposed an algorithmic approach to solving PC-CLRP.

First, we introduce a new meta-heuristic algorithm based on the Ant Colony Optimization algorithm. Our ant algorithm optimizes the 3 levels of decision: location, clustering and routing.

Next, we present new models based on two-index and three-index flow integer linear programming formulations. Known valid inequalities for CLRP are analyzed and adapted for PC-CLRP. This is complemented with the development of new valid inequalities and optimality cuts and their corresponding separation algorithms.

Finally, we implement a *Branch&Cut* algorithm based on one of the proposed models. Computational results are reported for both implemented algorithms on a new set of instances specially designed for the new problem. Additionally, we compare it with recent work for CLRP on instances from the literature obtaining competitive results.

Keywords: location routing problem, integer linear programming, branch and cut, ant colony, combinatorial optimization, prize-collecting

A Inés y nuestras amadas hijas, Julia y Josefina.

Lo desconocido es una abstracción; lo conocido, un desierto; pero lo conocido a medias, lo vislumbrado, es el lugar perfecto para hacer ondular deseo y alucinación.

Juan José Saer, *El entenado*

Agradecimientos

Quiero agradecer en primer lugar a mi Directora, Irene Loiseau, por todo el apoyo, el estímulo, el aguante y todo el tiempo tan valioso que me dedicó durante este largo y fructífero proceso. Un agradecimiento especial es para Julián Aráoz Durand que nos ayudó enormemente con la definición del problema a estudiar y que estuvo siempre dispuesto a organizar reuniones de trabajo cada vez que visitaba Buenos Aires. Quiero dedicar también esta tesis a la memoria de mi gran amigo Néstor Sameghini, a quien debo mi vocación por la computación y el haber elegido esta hermosa carrera.

Además, quiero agradecer especialmente a los jurados de esta tesis por todo el tiempo dedicado a evaluar el trabajo.

Otro agradecimiento especial es para Isabel Méndez Díaz, Paula Zabala, Juan Miranda Bront, Pablo Factorovich y a todos los integrantes del Grupo de Investigación Operativa, Optimización Combinatoria y Grafos. Gracias por toda la ayuda y por los lindos momentos compartidos en congresos y workshops durante estos años. Gracias Javier Marenco por las fructíferas conversaciones acerca de los distintos problemas del área. Todos tuvieron algo que ver con que este trabajo sea posible.

Quisiera también expresar mi reconocimiento a las instituciones desde las cuales he realizado este trabajo y he completado mi formación tanto académica como profesional. Muchas gracias al Departamento de Computación de la Facultad de Ciencias Exactas y Naturales de la Universidad de Buenos Aires.

Agradezco y dedico también este trabajo a mis padres. Gracias por haberme apoyado desde siempre con el estudio de la computación.

Esta tesis está especialmente dedicada a Inés, mi esposa y a mis dos hijas: Julia y Josefina. Sin todo el amor y el apoyo que me transmiten día a día esta tesis no hubiese sido posible. Gracias por bancarme en todo momento, por confiar siempre en mí, por dedicar fines de semana, postergar vacaciones y dejar muchas cosas de lado para que este trabajo llegara a su fin. Gracias Inés por todo el amor, por estar siempre, por ayudarme, por inspirarme y por haber sido la primera y última lectora de cada una de las páginas de este trabajo. Este logro es totalmente nuestro.

ÍNDICE GENERAL

1. Introducción **2**

1.1. El problema de Localización y Ruteo con Capacidades y Premios 4

1.2. Revisión de la literatura 7

1.2.1. Problemas de Localización y Ruteo 7

1.2.2. Problemas de Ruteo con Premios 9

1.2.3. Problemas de Localización y Ruteo con Premios 10

2. Algoritmo de Colonia de Hormigas para el problema PC-CLRP **11**

2.1. Optimización por Colonia de Hormigas 11

2.2. Metaheurística de Colonia de Hormigas para el problema PC-CLRP 14

2.3. Localización ACS 17

2.4. Activar Clientes 18

2.5. Clusterizado ACS 18

2.6. Ruteo ACS 20

2.7. Vecindades 22

2.8. Búsqueda Local 24

2.9. Procedimientos de actualización de feromona 27

2.9.1. Inicialización 27

2.9.2. Actualizaciones locales 28

2.9.3. Actualizaciones globales 29

2.10. Resultados Computacionales 30

2.11. Conclusiones 38

3. Algoritmo Branch&Cut para el problema PC-CLRP **40**

3.1. Programación Lineal 40

3.2. Programación Lineal Entera Mixta 41

3.3. Algoritmos de Plano de Corte . . . 42
3.4. Algoritmos *Branch&Bound* . . . 45
3.5. Algoritmos *Branch&Cut* . . . 47
3.6. Modelos de programación lineal entera para PC-CLRP . . . 49
3.6.1. Modelo 2 índices inicial . . . 49
3.6.2. Modelo 2 índices extendido . . . 50
3.7. Desigualdades válidas . . . 54
3.7.1. Desigualdades válidas w-z . . . 54
3.7.2. Desigualdades válidas PC-Capacidad y PC-Subtour . . . 54
3.7.3. Desigualdades y-Capacidad . . . 56
3.7.4. Desigualdades válidas 3-Path . . . 57
3.7.5. Desigualdades válidas Co-Circuit y 1 Co-Circuit . . . 57
3.7.6. Desigualdades válidas PC-path-elimination . . . 58
3.7.7. Desigualdades válidas derivadas de CVRP . . . 59
3.8. Cortes de Optimalidad . . . 63
3.8.1. Cortes Facility-degree . . . 63
3.8.2. Cortes de depósitos en equilibrio . . . 64
3.8.3. Cortes de clientes en equilibrio . . . 64
3.8.4. Cortes de activaciones implicadas 1 . . . 65
3.8.5. Cortes de activaciones implicadas 2 . . . 65
3.9. Algoritmo Branch&Cut . . . 66
3.9.1. Relajación inicial . . . 66
3.9.2. Preprocesamiento . . . 67
3.9.3. Algoritmos de separación . . . 68
3.9.4. Estrategia de separación . . . 77
3.9.5. Estrategia de branching . . . 80
3.9.6. Selección de nodos . . . 82
3.9.7. Heurística inicial . . . 82
3.9.8. Heurística primal . . . 82
3.10. Resultados Computacionales . . . 86
3.11. Conclusiones . . . 99

4. Conclusiones Generales **102**

4.1. Trabajo futuro 104

A. Anexo I **106**

A.1. Conjuntos de instancias utilizados 106

A.2. Generación de nuevas instancias 107

A.3. Modelo 3 índices 111

B. Glosario de acrónimos **114**

C. Glosario de notación **117**

Bibliografía **128**

ÍNDICE DE FIGURAS

2.1. Operadores de Búsqueda Local Intra-ruta 25
2.2. Operadores de Búsqueda Local Intra-cluster 26
2.3. Operadores de Búsqueda Local Prize-Collecting 26

3.1. Algoritmos de Plano de Corte. (a) Relajación Lineal. (b) Cápsula convexa. 43
3.2. Restricción PC-Subtour violada 55
3.3. Restricción clientes en equilibrio 65

ÍNDICE DE CUADROS

2.1. Resultados experimentación en estrategia de apertura de depósitos 32
2.2. Resultados experimentación Clonación 33
2.3. Resultados experimentación procedimiento *activarClientes* 34
2.4. Resultados CLRP sobre las instancias Prodhon 36
2.5. Comparación con metaheurísticas recientes (gaps a BKS) 36
2.6. Resultados 1 Prize-Collecting 37
2.7. Resultados 2 Prize-Collecting 38

3.1. Resultados nodo raíz 88
3.2. Resultados con participación de distintos costos en la cota inferior 88
3.3. Resultados con participación de ganancias en la cota inferior 89
3.4. Resultados Branch&Cut parcial y completo 89
3.5. Resultados nodo raíz y Branch&Cut completo sobre CLRP 90
3.6. Cantidad de desigualdades encontradas al aplicar el algoritmo Branch&Cut completo 90
3.7. Eficiencia en versiones I, II, III y IV 92
3.8. Eficiencia en versiones V, VI. VII y VIII 92
3.9. Resultados comparación estrategias de separación 94
3.10. Resultados comparación estrategias de branching 95
3.11. Resultados CLRP sobre las instancias Prodhon 96
3.12. Comparación resultados CLRP sobre las instancias Prodhon 97
3.13. Resultados Branch&Cut 20-50 clientes 98
3.14. Resultados Branch&Cut 100 clientes 98
3.15. Resultados Branch&Cut 200 clientes 99

A.1. Instancias de Prodhon 107
A.2. Instancias generadas 1 - PC-CLRP 109

A.3. Instancias generadas 2 - PC-CLRP 110

ÍNDICE DE ALGORITMOS

2.1. Optimización por Colonia de Hormigas 13
2.2. Algoritmo Principal 16
2.3. activarDepositos 18
2.4. activarClientes 19
2.5. clusterizar 20
2.6. rutear 22
3.7. Algoritmo de Planos de Corte 43
3.8. Algoritmo Branch&Bound para un problema de minimización 46
3.9. Algoritmo Branch&Cut para un problema de minimización 48
3.10. Estrategia de Branching 81
3.11. Heurística Primal 83
3.12. construccionC&W 84
3.13. mejoramientoC&W 85

Capítulo 1

INTRODUCCIÓN

El Problema de Localización y Ruteo de Vehículos (*Location Routing Problem - LRP*) es la combinación de dos problemas muy estudiados del área de la Investigación Operativa: el Problema de Ruteo de Vehículos (*Vehicle Routing Problem - VRP*) y el Problema de Localización de Plantas o Depósitos (*Facility Location Problem - FLP*). En el sector logístico, al momento de diseñar cadenas de suministro, estos dos problemas son usualmente tratados de manera independiente. Esto se debe a que la localización es una decisión estratégica que suele considerarse durante una etapa de planeamiento y el ruteo es una actividad de tipo operacional que puede ser programada y modificada dinámicamente en un período de tiempo acotado. Sin embargo, como se demuestra en el trabajo de Salhi y Rand [89], este enfoque puede generar soluciones subóptimas y es por esto que el *LRP* se ha convertido en un campo muy interesante de investigación.

El Problema de Localización y Ruteo con Capacidades (*Capacitated Location Routing Problem - CLRP*) es la variante más estudiada del *LRP*. Este problema se suele modelar utilizando un grafo no dirigido donde los nodos representan a los clientes a atender y al conjunto de posibles localizaciones de depósitos. Los ejes representan los tramos que conectan los clientes entre sí y los clientes con los depósitos. Cada tramo tiene asociado un costo de ruteo. Se cuenta con una flota homogénea de vehículos. Cada vehículo dispone de una cierta capacidad máxima de carga e incurre en un determinado costo en caso de ser utilizado para realizar una ruta. Además, se debe determinar qué depósitos abrir y cada depósito abierto tiene también un costo asociado y una capacidad máxima de almacenamiento. Cada cliente tiene asociada una cierta demanda que se necesita satisfacer en su totalidad. Sólo un vehículo debe atender la demanda de cada cliente. La demanda total de los clientes pertenecientes a una ruta no puede exceder la capacidad del vehículo que la realiza. Cada ruta tiene que comenzar y terminar en un mismo depósito. Los vehículos recorren una sola ruta. La suma de las demandas de los clientes atendidos desde un determinado depósito no puede sobrepasar la capacidad del mismo. El objetivo de este problema es la minimización simultánea de los costos de los ejes atravesados, de los costos de apertura de depósitos y de los costos de vehículos utilizados.

Este trabajo se centra en el estudio de una nueva versión del *LRP* que se ha denominado Localización y Ruteo de Vehículos con Capacidades y Premios (*Prize-Collecting Capacitated Location Routing Problem - PC-CLRP*). La diferencia principal de esta variante es la posibilidad de decidir si se dejan o no clientes sin visitar. Si un cliente es visitado se obtendrá una ganancia o premio. Por consiguiente, se deberá elegir cuáles clientes atender con el fin de maximizar los beneficios.

El estudio de la versión con premios de este problema está motivado en el funcionamiento de las cadenas de suministro. Las empresas pueden evaluar la posibilidad de cumplir con las tareas de distribución de algunos clientes mediante outsourcing. En ciertos casos convendrá atender al cliente directamente, mientras que en otros, el costo de outsourcing será preferible al costo de realizar el ruteo por cuenta propia.

En este trabajo se presenta un estudio algorítmico del problema *PC-CLRP*. A continuación, en la sección 1.1, se define formalmente el problema a partir de un primer modelo de programación lineal entera. En la sección 1.2 se presenta la revisión bibliográfica efectuada sobre las distintas variantes de problemas de localización y ruteo y de los problemas de ruteo con premios.

En el capítulo 2 se describe un algoritmo metaheurístico propuesto para el problema *PC-CLRP*. Se presenta un algoritmo de Colonia de Hormigas que utiliza 3 colonias diferentes trabajando secuencialmente para construir soluciones *PC-CLRP*. La metaheurística cuenta con una colonia especializada para la etapa de localización, otra para la de clusterizado y una tercera diseñada para construir rutas entre los depósitos abiertos y los clientes. Al final del capítulo se informan los resultados obtenidos, tanto para el problema *PC-CLRP* aquí estudiado como para el problema *CLRP*. Se comparan los resultados con otros trabajos publicados para el problema *CLRP*. Los resultados obtenidos muestran un muy buen desempeño de la metaheurística aún para las instancias del problema *CLRP*.

En el capítulo 3 se presenta un segundo modelo de programación lineal entera y se implementa un algoritmo *Branch&Cut* basado en dicho modelo. Se adaptan las familias de desigualdades válidas conocidas para el problema *CLRP*. Se proponen nuevas familias de desigualdades válidas y de cortes de optimalidad que explotan las particularidades del problema. Se demuestra la validez de las desigualdades y se proponen nuevos algoritmos de separación. Se utiliza el algoritmo presentado en el capítulo 2 como heurística inicial y se propone una nueva heurística de mejoramiento. Se muestran además todos los detalles involucrados en la implementación del algoritmo *Branch&Cut*. Entre otros, la etapa de fijación y seteo de variables, la estrategia de branching o las estrategias de separación. Luego se presentan los resultados computacionales

obtenidos sobre los problemas *PC-CLRP* y *CLRP*. Para este último caso se comparan los resultados obtenidos con otros trabajos publicados que implementan algoritmos *Branch&Cut*. Se utilizan los resultados obtenidos en el capítulo 2 para comparar las cotas inferiores obtenidas con el algoritmo exacto. Se obtienen las soluciones óptimas de las instancias pequeñas y de algunas medias (20 y 50 clientes). Las cotas inferiores obtenidas en el resto de las instancias muestran un buen desempeño del algoritmo *Branch&Cut*.

En el apéndice A.1 se detallan las instancias de prueba utilizadas. En el apéndice A.2 se describe el algoritmo generador de instancias *PC-CLRP*, diseñado para construir el nuevo conjunto de instancias específicas para el problema *PC-CLRP*. En el apéndice A.3, por último, se propone un nuevo modelo de programación lineal entera de 3 índices.

1.1. El problema de Localización y Ruteo con Capacidades y Premios

El Problema de Localización y Ruteo con Capacidades y Premios (*PC-CLRP*) puede ser modelado en un grafo completo, dirigido y con pesos en los ejes. Se dispone de un flota de vehículos con capacidades homogéneas. Cada vehículo genera un costo cuando es utilizado. Además, se cuenta con un conjunto de posibles localizaciones de depósitos. Se debe seleccionar cuáles abrir. Cada depósito tiene un costo de apertura asociado. Asimismo, cada cliente tiene una ganancia asociada.

Formalmente, la red puede ser representada como un grafo dirigido $G = (V, A)$ con un conjunto de nodos V, donde $V = I \cup J$ representa la unión de las posibles localizaciones de depósitos I y el conjunto de clientes J. El conjunto de ejes dirigidos A interconecta los clientes entre sí, y también a las localizaciones de depósitos con los clientes. No se establece conexión alguna entre las diferentes localizaciones. Cada cliente $j \in J$ tiene una demanda d_j y una ganancia b_j. Cada localización $i \in I$ tiene un costo de apertura O_i asociado y una capacidad máxima W_i. Cada eje $(i, j) \in A$ tiene un costo c_{ij} y cada vehículo utilizado genera un costo F. Además, cada vehículo tiene una capacidad máxima Q.

El objetivo de este problema puede pensarse como la determinación de los depósitos a abrir, los clientes a atender y las rutas a efectuar para satisfacer las demandas de los clientes atendidos. Se deben cumplir además las siguientes restricciones:

- Cada cliente puede o no ser visitado. En caso de ser visitado, el mismo debe ser atendido por un único vehículo.
- La demanda total de los clientes visitados en una misma ruta no debe exceder la capacidad máxima del vehículo que la recorre.

- Las demandas de los clientes atendidos desde un depósito no debe exceder la capacidad máxima del depósito.
- Las rutas comienzan y terminan en un mismo depósito.
- Solamente un viaje es permitido por cada vehículo.

La función objetivo de este problema puede ser expresada como la maximización de los beneficios y, al mismo tiempo, la minimización de los 3 costos fundamentales: costos de apertura de depósitos, costos de vehículos utilizados y costos de ruteo (ejes utilizados por todos los viajes).

Se define el conjunto de ejes del grafo como $A = \{(i,j) : i \neq j \wedge i,j \in J\} \cup \{(i,j) : i \in I, j \in J\} \cup \{(j,i) : i \in I, j \in J\}$. Las variables toman los valores $x_{ij} = 1$ si el eje $(i,j) \in A$ es utilizado en alguna ruta, 0 en caso contrario y $z_i = 1$ si la localización i es abierta, 0 en caso contrario.

Antes de presentar una primera formulación matemática del problema se define notación auxiliar que permite expresar el modelo de manera más clara y concisa. Para cada subconjunto $U \subseteq V$ se define $E(U) = \{(u,w) \in A : u,w \in U\}$, $\delta^+(U) = \{(u,w) \in A : u \in U, w \notin U\}$ y $\delta^-(U) = \{(u,w) \in A : u \notin U, w \in U\}$. Para cada par de subconjuntos disjuntos U y W se define $U : W = \{(u,w) \in A : u \in U, w \in W\}$. Para un conjunto dado $M \subseteq A$ se define $x(M) = \sum_{e \in M} x_e$. Para un conjunto de clientes $S \subseteq J$, se define $d(S) = \sum_{j \in S} d_j$ y $b(S) = \sum_{j \in S} b_j$. En todos los casos, haciendo abuso de notación se representan los conjuntos de un solo elemento sin utilizar llaves.

La formulación (PC2) basada en un modelo de 2 índices es la siguiente:

$$min \sum_{e \in A} c_e x_e + \sum_{i \in I} O_i z_i + \sum_{i \in I} \sum_{e \in \delta^+(i)} F x_e - \sum_{i \in J} b_i \sum_{e \in \delta^+(i)} x_e \tag{1.1}$$

sujeto a

$$x(\delta^+(i)) = x(\delta^-(i)) \qquad \forall_{i \in V} \tag{1.2}$$

$$x(\delta^+(i)) \leq 1 \qquad \forall_{i \in J} \tag{1.3}$$

$$x(\delta^+(S)) \geq \frac{\sum_{i \in S} d_i x(\delta^+(i))}{Q} \qquad \forall_{S \subseteq J} \tag{1.4}$$

$$x(\delta^+(i)) \leq N z_i \qquad \forall_{i \in I} \tag{1.5}$$

$$x(i:S) + x(S: I \setminus i) + x(E(S)) \leq \sum_{j \in S} x(\delta^+(j)) \qquad \forall_{S \subseteq J} \quad i \in I \tag{1.6}$$

$$x(i:S) + x(E(S)) \leq \sum_{j \in S} x(\delta^+(j)) - 1 \qquad \forall_{\substack{S \subseteq J \\ d(S) > W_i}}^{i \in I} \tag{1.7}$$

$$x_e \in \{0,1\} \qquad \forall_{e \in A} \tag{1.8}$$

$$z_i \in \{0,1\} \qquad \forall_{i \in I} \tag{1.9}$$

La función objetivo (1.1) efectúa la minimización de los costos de ruta, de apertura de localizaciones y de uso de vehículos. Además, maximiza los beneficios de los clientes visitados. Un cliente es visitado si la suma de las variables que representan ejes de salida contabiliza el valor 1.

Las ecuaciones (1.2) determinan la conservación de flujo en todos los nodos del grafo.

Las desigualdades (1.3) piden que cada cliente sea visitado a lo sumo una vez.

Las desigualdades (1.4) son las conocidas desigualdades de capacidad adaptadas a este nuevo problema. Establecen al mismo tiempo que no puede haber subtours entre clientes y que no se debe superar la capacidad máxima de los vehículos. En esta formulación se ha presentado una nueva versión de las desigualdades teniendo en cuenta la posibilidad de que un cliente no sea visitado.

Las desigualdades (1.5) establecen la relación entre las variables x y z. En las mismas, se establece que cada localización abierta (con ejes de salida activados) debe tener la variable respectiva z con valor 1. Para ello, se utiliza una constante N, de valor lo suficientemente grande como para superar a la máxima cantidad de rutas salientes de un depósito.

Las desigualdades (1.6) exigen que las rutas comiencen y finalicen en un mismo depósito. En la literatura se las conoce como "path elimination constraints" (véase Belenguer y otros [16]). Sin embargo, la presente versión considera la posibilidad de clientes no atendidos incluyendo, en la parte derecha de la misma, la suma del flujo de salida de cada cliente de S para considerar los clientes activos.

Las desigualdades (1.7) controlan la capacidad máxima de los depósitos.

La fomulación presentada está inspirada en el modelo de 2 índices introducido en Belenguer y otros [16]. A diferencia de la propuesta para el problema *CLRP*, la formulación (PC2) es una versión dirigida adaptada para el problema *PC-CLRP*.

El problema *CLRP* es $\mathcal{NP} - Hard$ porque es la combinación y generalización de dos problemas $\mathcal{NP} - Hard$: el Problema de Localización Capacitado (*Capacitated Facility Location Problem - CFLP*) y el Problema de Ruteo de Vehículos Capacitado (*Capacitated Vehicle Routing Problem - CVRP*). En efecto, el problema *CFLP* puede pensarse como el problema *CLRP* con vehículos de capacidad ilimitada ($Q = \infty$), costo de utilización de vehículos igual a 0 ($F = 0$) y costo infinito para los ejes que conectan clientes entre sí ($c_{ij} = \infty \quad \forall i, j \in J$). Por su parte, el problema *CVRP* puede pensarse como el problema *CLRP* con un único depósito ($|I| = 1$).

El problema *PC-CLRP* es también $\mathcal{NP} - Hard$ porque es, por su parte, una generalización del problema *CLRP*. Para ello basta considerar ganancias $b_i = \infty \quad \forall i \in J$.

1.2. Revisión de la literatura

De acuerdo a lo relevado, se han encontrado en la literatura algunos trabajos para los problemas de Localización y Ruteo, otros aplicados a varios tipos de problemas de Ruteo con Premios y trabajos recientes que describen aplicaciones muy específicas de problemas de Localización y Ruteo con Premios.

1.2.1. Problemas de Localización y Ruteo

La idea de combinar localización de depósitos y ruteo de vehículos aparece inicialmente en los trabajos de Maranzana [69], Von Boventer [101] y Webb [103]. Sin embargo, es Watson-Gandy y Dohrn [102] probablemente el primer trabajo donde se trata en simultáneo claramente el ruteo de vehículos y la localización de depósitos.

Min y otros [73] propusieron una clasificación de los trabajos *LRP* basada en los métodos de resolución y en las perspectivas de los problemas. La clasificación más reciente de Nagy y Salhi [75] presenta una taxonomía jerárquica que agrupa la literatura basándose en diferentes características y metodologías utilizadas. Se describen los trabajos que tratan problemas determinísticos, estocásticos y dinámicos. Dentro de las metodologías exactas se presentan formulaciones de 2-índices para el *CLRP* propuestas por Laporte [62], Baldacci y otros [13], Belenguer y otros [16] y Contardo y otros [21]. Formulaciones de 3-índices introducidas para el *LRP* por Perl y Daskin [80] y Hansen y otros [52] y para el *CLRP* por Prins y otros [85].

Los algoritmos exactos para el problema *CLRP* son muy recientes. Siguiendo la revisión de Prodhon y Prins [87], algunos de ellos son el algoritmo *Branch&Cut* de Belenguer y otros [16], el algoritmo

Branch&Price de Akca y otros [4], la modelización como un problema de partición de conjuntos junto a la descomposición del problema *LRP* en varios *Multiple Depot Vehicle Routing Problem - MDVRP* efectuada por Baldacci y otros [13], y el reciente algoritmo *Branch&Cut&Price* de Contardo y otros [22].

Dada la complejidad del problema *CLRP*, con los algoritmos exactos se logra abordar y resolver a optimalidad sólo instancias pequeñas y medianas. Debido a ello, en la literatura se han propuesto un gran número de algoritmos heurísticos para tratar instancias del *CLRP* más grandes. Nagy y Salhi [75] clasificaron a estos algoritmos heurísticos como basados en métodos *secuenciales*, *iterativos*, *jerárquicos* y *clusterizados*.

Los métodos secuenciales normalmente resuelven el problema de localización y luego diversos problemas de ruteo utilizando los depósitos abiertos en el primer paso. Según Salhi y Rand [89], este tipo de enfoques anula la colaboración entre ambos subproblemas. Dentro de este tipo de heurísticas podemos citar los trabajos de Tuzun y Burke [98] que proponen una metaheurística de tabú search de 2 fases que itera entre las fases de localización y ruteo para encontrar mejores soluciones. En ese trabajo se logra abordar instancias de 200 clientes. El trabajo de Prins y otros [85] propone también un algoritmo de 2 fases con intercambio de información entre ambas fases. En la primera se resuelve un problema *CFLP* integrando los clientes en super-clientes y aplicando un método de relajación lagrangiana. En la segunda fase se aplica una búsqueda tabú granular (*Granular Tabu Search - GTS* - introducida por Toth y Vigo [94]) para resolver un problema de ruteo con múltiples depósitos (*MDVRP*). Escobar [34] presenta también un algoritmo de 2 fases híbrido aplicado al problema *CLRP* con excelentes resultados.

Se han propuesto métodos basados en una estructura jerárquica para resolver el problema *CLRP*. Por lo general, primero se resuelve un problema *CFLP* y luego se resuelven diversos subproblemas *CVRP*. Algunos ejemplos de estos métodos son los trabajos de Melechovsky y otros [71] y Albareda-Sambola y otros [5].

Barreto y otros [15] introdujeron los métodos clusterizados para el problema *CLRP*. En la primera fase, el conjunto de clientes se particiona en clusters de acuerdo a la capacidad de los vehículos. En la segunda fase, se resuelven para cada cluster Problemas de Viajante de Comercio (*Traveling Salesman Problem - TSP*). Por último, en una última fase se agrupan los circuitos encontrados en supernodos para resolver el problema *CFLP* correspondiente.

En la reciente revisión del *LRP* efectuada por Prodhon y Prins [87] los algoritmos heurísticos se agrupan de acuerdo al tipo de técnica utilizada. Entre ellas se mencionan trabajos basados en heurísticas constructivas, metaheurísticas basadas en vecindades (*Greedy randomized adaptive search procedure - GRASP* [84], *GRASP* x ELS [18], *Simulated Annealing - SA* [104] y [57], *Variable Neighborhood Search - VNS* [81], *Variable Neighborhood Descent - VND* [55] y *Adaptive Large-Neighborhood Search - ALNS* [53]), metaheurísticas

genéticas [83], metaheurísticas multi agente - MACO [92] y metaheurísticas híbridas combinadas con métodos exactos - matheuristics (LRGTS [85], *GRASP* +ILP [23], *Variable Large Neighborhood Search - VLNS* + ILP [81]).

La revisión de Prodhon y Prins [87] también analiza varios problemas asociados como los problemas multi-escalonados de localización y ruteo *Multi-Echelon Location-Routing Problems - MELRP*, problemas con depósitos móviles, problemas con objetivos no-lineales y un gran número de otras variantes cercanas a *LRP*. Para mayor detalle se puede consultar dicha revisión y la previa de Nagy y Salhi [75].

1.2.2. Problemas de Ruteo con Premios

Diversos tipos de problemas de ruteo han sido estudiados en la literatura bajo el nombre de problemas con beneficios (with Profits) o de recolección de premios (Prize-Collecting). De acuerdo a la reciente revisión de Archetti y otros [8] la mayor parte de los mismos son presentados como variantes del problema del Viajante de Comercio (*TSP*).

El *Orienteering Problem - OP* (versión del *TSP* con premios y longitud total restringida), estudiado inicialmente en Tsiligirides [90] y Golden y otros [44], ha sido tratado también por Laporte y Martello [61], Gendreau y otros [42] y Thomadsen y Stidsen [91] bajo el nombre de *Selective Traveling Salesperson Problem - STSP*. El *Maximum Collection Problem - MCP* (Butt y Cavalier [17], Kataoka y Morito [88]) y el *Bank Robber Problem - BRP* (Awerbuch y otros [10]) son otros de los nombres con que se ha tratado al *OP*.

Otros problemas con un sólo vehículo son el *Profitable Tour Problem - PTP* y el *Prize-Collecting Traveling Salesman Problem - PCTSP* estudiados en Balas [11, 12], Dell'Amico y otros [26, 27] y Fischetti y Toth [37]. En Archetti y otros [7] se presenta la versión capacitada del *PTP*.

Se han estudiado también versiones generalizadas de estos problemas para más de un vehículo. Se puede destacar el *Team Orienteering Problem - TOP*, (estudiado también bajo el nombre *Multiple Tour Maximum Collection Problem - MTMCP* en Butt y Cavalier [17]) y la versión capacitada del mismo problema *Capacitated Team Orienteering Problem - CTOP* estudiada en Archetti y otros [7]. La versión con ventanas de tiempo *Team Orienteering Problem with Time Windows - TOPTW* ha sido estudiada por Vansteenwegen y otros [99], Montemanni y Gambardella [74], Gambardella [39], Tricoire y otros [97], Labadie y otros [60] y Lin y Yu [64]. Todos los trabajos sobre *TOPTW* proponen algoritmos metaheurísticos.

Las definiciones y descripciones detalladas de cada uno de estos problemas puede encontrarse en Archetti y otros [8] y en las revisiones previas de Vansteenwegen y otros [100] y Feillet y otros [36]. Entre los métodos de resolución utilizados se encuentran algoritmos exactos de generación de columnas, relajaciones lagrangianas, *Branch&Price* y *Branch&Cut*. Además, se han presentado también diversos trabajos con

heurísticas tradicionales y metaheurísticas basadas en algoritmos *VNS*, colonia de hormigas (*Ant Colony Optimization - ACO*), *GRASP*, *GTS*, *SA* y Algoritmos Evolutivos entre otros.

1.2.3. Problemas de Localización y Ruteo con Premios

Una variante no capacitada del Problema de Localización y Ruteo de Vehículos con Premios puede encontrarse en el trabajo de Polat [82]. En este trabajo se estudia el problema de recolección de envases de vidrio para su posterior reciclado. Para ello, es necesario localizar centros de recolección de botellas, para que los vecinos se acerquen y dejen sus envases vacíos. Al mismo tiempo es necesario también retirar envases de vidrio desde determinados puntos fijos (restaurantes, hospitales, etc). Esta es una versión particular del *Single-Echelon Facility Location Problem - SEFLP*. El problema es tratado como la combinación de dos problemas: *Partial Maximal Coverage Location Problem - MCLP-P* y *Vehicle Routing Problem with Profits - VRP-P*. Se utiliza la heurística clásica para localización y ruteo de 2 fases, clusterizado primero y ruteo después. Luego, se utiliza una variante de la heurística propuesta por Chao y otros [19] para el *OP*. Se finaliza con una etapa de mejora basada en 2 operadores de búsqueda local.

Otra variante aplicada del Problema de Localización y Ruteo con Premios ha sido estudiada en el trabajo de Ahn y otros [2, 3]. La misma ha sido denominada como *Generalized Location Routing Problem with Profits - GLRPP*, y es el estudio de ciertos problemas originados en la industria aeroespacial relacionados con la exploración de las superficies planetarias en misiones espaciales. En ese problema, se estudia un Problema de Localización y Ruteo con Premios, con características especiales que lo hacen diferente de los problemas de Localización y Ruteo generales que aparecen en la literatura. Se descompone el problema en diferentes subproblemas, clusterizando primero el conjunto de posibles ubicaciones. Dentro de cada cluster se consideran todas las combinaciones de bases y misiones. Luego, se resuelve un problema *MDVRP* with Profits utilizando un algoritmo heurístico basado en un método *Branch&Price*. Por último, se resuelve un problema de partición de conjuntos (*Set-Partitioning Problem - SPP*) para seleccionar una estrategia por cluster. La función objetivo de dicho problema busca minimizar las ganancias de los clientes no atendidos, pero no considera ninguno de los costos habituales en los problemas logísticos. En su tesis de doctorado, Tresoldi [96] estudia también este problema junto a otros dos problemas de ruteo y describe a este tipo de problemas de Localización y Ruteo como problemas con decisiones de localización "pasivas", ya que los costos no son tenidos en cuenta en la función objetivo a optimizar. De acuerdo a lo relevado, no se han estudiado aún problemas de Localización y Ruteo con Premios que consideren los costos de distribución como parte de la función a optimizar.

Capítulo 2

ALGORITMO DE COLONIA DE HORMIGAS PARA EL PROBLEMA PC-CLRP

Debido a la complejidad que presentan los problemas de optimización NP-Hard, el estudio de técnicas alternativas a los métodos exactos ha adquirido una importante atención por parte de la comunidad científica. Las heurísticas son técnicas que permiten encontrar soluciones de buenas calidad con un esfuerzo computacional moderado. Es importante señalar que los algoritmos heurísticos no garantizan optimalidad.

Las metaheurísticas son algoritmos que utilizan y mejoran las ideas de los algoritmos heurísticos. El término metaheurística fue introducido por F. Glover [43] a partir del estudio de los vínculos entre la Investigación Operativa y la Inteligencia Artificial. Las metaheurísticas son metodologías de alto nivel para la exploración de espacios de soluciones, que utilizan y combinan diferentes tipos de heurísticas especialmente diseñadas para problemas de optimización particulares.

2.1. Optimización por Colonia de Hormigas

La optimización por Colonia de Hormigas es una metaheurística en la cual una colonia artificial de hormigas coopera en la búsqueda de buenas soluciones para un problema de optimización complejo. La optimización por Colonia de Hormigas (presentada en detalle en Dorigo y Stützle [31]) fue inspirada al observar la conducta de recolección de alimentos de las hormigas reales. Las hormigas exploran aleatoriamente los alrededores del hormiguero. Cuando encuentran comida, vuelven al nido dejando rastros de feromona (sustancia química segregada que puede ser olida o rastreada por otras hormigas). Las hormigas pueden seguir varios caminos alternativos a las fuentes de comida y volver pero se ha observado que, gracias al refuerzo de los rastros de feromona dejado por el paso de sucesivas hormigas, sólo los caminos más cortos permanecen en uso, ya que las hormigas prefieren seguir los caminos con concentraciones más fuertes de feromona.

El algoritmo de optimización por Colonia de Hormigas (ACO) replica esta conducta, agregando algunas características que lo hace más eficiente para la implementación en una computadora. Las hormigas son

implementadas como un conjunto de agentes. Cada una construye una solución completa del problema de manera incremental. En el pseudocódigo 2.1 puede observarse la idea general de los algoritmos *ACO*.

En cada paso constructivo (línea 4 del algoritmo 2.1), las hormigas seleccionan un nuevo elemento que formará parte de la solución. Para ello, se tienen en cuenta 2 parámetros: el rastro de feromona y el valor de atracción. Las metaheurísticas de colonias de hormigas utilizan un mecanismo constructivo basado en funciones aleatorias que permiten componer, paso a paso, una nueva solución. Este mecanismo se vale de información histórica (feromona) y de información heurística del problema (valor de atracción) generada sobre cada elemento evaluado para seleccionar el nuevo candidato a incorporar. Esta construcción es sumamente elemental y requiere, para que el método tenga éxito, que el valor de atracción represente de la mejor manera a todas las variables involucradas en el problema. La combinación de ambos parámetros se realiza mediante la llamada función aleatoria proporcional. En el caso de problemas de ruteo simples, donde la única variable a optimizar es la suma de las distancias recorridas entre nodos, generalmente el valor heurístico es calculado como la función inversa a la distancia sobre cada tramo. Es decir, será menos atractivo un nodo que se encuentre a mayor distancia y más atractivo uno más cercano. Llevándolo a un ejemplo, para el problema del Viajante de Comercio clásico (*TSP*), el valor heurístico utilizado cuando se está en un nodo y se quiere seleccionar el siguiente, es la función inversa de la distancia desde el nodo a los posibles nodos siguientes.

El valor de atracción de un movimiento entre el nodo i y el nodo j se calcula de acuerdo a una heurística que expresa la conveniencia a priori de dicho movimiento. El nivel de feromona de un movimiento representa una indicación dinámica a posteriori de su efectividad. En otras palabras, si la hormiga artificial huele un rastro de feromona fuerte que la conduce a un nodo, sabe que es una dirección prometedora para explorar. La *función aleatoria proporcional* empleada en el paso constructivo de un problema *TSP* sería

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum_{l \in N_i^k} \tau_{il}^\alpha \eta_{il}^\beta} & si\ j \in N_i^k \\ 0 & en\ caso\ contrario \end{cases} \tag{2.1}$$

Es decir, la hormiga k, estando en el nodo i elige el nodo j con probabilidad p_{ij}^k considerando el valor de atracción η_{ij} y el nivel de feromona τ_{ij}. Los parámetros α y β determinan la influencia de τ_{ij} y η_{ij} en la función aleatoria y N_i^k es la vecindad considerada sobre el nodo i. Se define la vecindad N_i^k de un nodo i como el conjunto de los k nodos más cercanos.

El paradigma *Ant Colony System - ACS* introducido por Dorigo y Gambardella [29] modifica la regla constructiva reemplazando la función anterior por la llamada *función pseudo-aleatoria proporcional*.

$$j = \begin{cases} arg\,\text{máx}_{l \in N_i^k} \{\tau_{il}^{\alpha} \eta_{il}^{\beta}\} & si\ q \leq q_0 \\ J & en\ caso\ contrario \end{cases} \tag{2.2}$$

En este caso se utiliza una variable aleatoria q uniformemente distribuida en $[0, 1]$, q_0 $(0 \leq q_0 \leq 1)$ es un parámetro y J es una variable aleatoria seleccionada de acuerdo a la distribución probabilística dada por la ecuación 2.1. En otras palabras, con probabilidad q_0 la hormiga k elige el mejor movimiento considerando los niveles de feromona y los valores de atracción. Mientras que con probabilidad $1 - q_0$ elige una exploración aleatoria que es proporcional a los niveles de feromona y a los valores de atracción sobre los ejes involucrados.

Inmediatamente después de cada paso constructivo, la feromona asociada al eje incorporado es actualizada. La actualización incluye refuerzo y evaporación de feromona. La feromona se evapora para evitar que el algoritmo quede atrapado en mínimos locales reduciendo el nivel de la misma sobre los ejes participantes.

En la ecuación siguiente puede observarse la función de actualización local de feromona comúnmente utilizada en los algoritmos ACS

$$\tau_{ij} = (1 - \xi)\tau_{ij} + \xi\tau_0 \tag{2.3}$$

donde τ_{ij} es el valor de la matriz de feromona para el eje (i, j), $0 \leq \xi \leq 1$ es el parámetro que determina la tasa de evaporación de feromona y τ_0 es el mismo parámetro constante utilizado en la inicialización de la matriz de feromona (línea 1 del pseudocódigo 2.1).

En el pseudocódigo 2.1 se ha utilizado la notación Z^{bs} para representar a la mejor solución del algoritmo, Z^{it} la mejor solución de la iteración, L^{bs} el valor de la función objetivo de la mejor solución Z^{bs} y L^{it} el valor de la función objetivo de la mejor solución de la iteración.

Algoritmo 2.1 Optimización por Colonia de Hormigas

```
1:  Inicialización (Parámetros, Matriz de Feromona, datos, Solución Inicial Z^bs)
2:  while not(finalizado) do
3:     for each ant do
4:        Construir una nueva solución
5:        Actualización Local de Feromona
6:     end for
7:     if L^it < L^bs then
8:        Z^bs ← Z^it
9:     end if
10:    Búsqueda Local
11:    Actualización Global de Feromona
12: end while
13: return Z^bs
```

El procedimiento de Búsqueda Local (línea 10 del algoritmo 2.1) permite mejorar cada una de las soluciones construidas con el fin de intentar alcanzar óptimos locales. La utilización de dichos procedimientos es opcional en la definición de *ACO* presentada en Dorigo y Stützle [31], pero su utilización mejora la calidad de las soluciones construidas y disminuye notablemente el tiempo de convergencia del algoritmo.

Al finalizar cada iteración, se efectúa el proceso de actualización global de feromona (línea 11 del pseudocódigo 2.1). En el paradigma *ACS* se actualizan sólo los ejes de la mejor solución encontrada hasta el momento (Z^{bs}). En este caso, se aplica un proceso de evaporación y de refuerzo regido por la ecuación

$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho\Delta\tau_{ij}^{bs} \quad \forall(i,j) \in Z^{bs} \tag{2.4}$$

donde $0 \leq \rho \leq 1$ es el parámetro que determina la tasa de evaporación de feromona, $\Delta\tau_{ij}^{bs} = 1/C^{bs}$ y C^{bs} es la longitud del camino hamiltoniano de la mejor solución.

Diferentes variantes de algoritmos de Colonia de Hormigas han sido propuestos en la literatura. La diferencia fundamental radica en la manera en que la feromona es actualizada como así también en la implementación del paso constructivo (línea 4 de 2.1). Algunas de las variantes analizadas en [31] son *Ant System - AS*, Elitist *AS*, Ant-Q, *ACS*, MAX-MIN *AS*, Rank-based *AS*, ANTS y Hyper-cube *AS*.

2.2. Metaheurística de Colonia de Hormigas para el problema PC-CLRP

El algoritmo presentado en este trabajo se basa en el paradigma *ACS* introducido por Dorigo y Gambardella [29]. Se usan 3 colonias diferentes *ACS* que trabajan secuencialmente para construir soluciones para el *PC-CLRP*. Cada colonia se especializa en la construcción de una parte de la solución. Se cuenta con una colonia que determina los depósitos a abrir (localización), otra que asigna los clientes a los depósitos abiertos (clusterizado) y una tercera que arma las rutas entre los depósitos y los clientes (ruteo).

Hay muchas implementaciones de la metaheurística *ACO* que difieren en la manera en que la feromona es usada y actualizada. En el paradigma *ACS*, al finalizar cada iteración, se realiza una actualización global de feromona utilizando sólo la mejor solución calculada. El algoritmo implementado, sin embargo, está inspirado en el framework *Multiple Ant Colony System - MACS* presentado por Gambardella y otros [40] y luego en Dorigo y Stützle [31] que utiliza la mejor solución calculada global y la mejor solución de la iteración para efectuar la actualización de feromona. Un enfoque similar fue aplicado para el problema *CLRP* por Ting y Chen [92].

Se utilizará la misma notación presentada en la sección 1.1 para *PC-CLRP*. En el algoritmo 2.2 puede verse el pseudocódigo del algoritmo principal de la metaheurística.

El algoritmo de Colonia de Hormigas comienza construyendo una solución inicial mediante un procedimiento adaptado de la heurística clásica de Vecinos Cercanos de Flood [38]. La localización, el clusterizado y el ruteo son efectuados con una heurística constructiva golosa que genera la solución utilizando para ello un conjunto acotado de alternativas basado en una vecindad reducida. Esta vecindad es calculada utilizando las distancias más cercanas entre los clientes y los depósitos. La solución generada mediante este proceso es usualmente de baja calidad, sin embargo, el propósito de la misma es ser utilizada como inicialización de la matriz de feromona.

Durante un número de iteraciones especificadas por el parámetro *maxiter*, se efectúa un proceso constructivo y de mejora donde las 3 colonias ACS trabajan para resolver los 3 problemas de optimización asociados: localización, clusterizado y ruteo. Los 3 ACS tienen su propia matriz de feromona y generan una parte de la solución completa. Cada hormiga en el ACS de localización tiene su contrapartida en el ACS de clusterizado y en el ACS de ruteo. Cada hormiga de la primera colonia selecciona un conjunto completo de localizaciones a abrir. Luego, la hormiga pasa esta información a su socia correspondiente de la colonia de clusterizado. La asignación de clientes a depósitos tiene lugar para cada uno de los depósitos pasados. Los resultados son enviados luego a la colonia de ruteo. Esta última colonia recibe la información de localización-clusterizado de las etapas precedentes y genera un ruteo completo para cada solución parcial recibida.

El primer paso de cada colonia (localización, clusterizado, ruteo; líneas 5-8) es una fase constructiva. Cada colonia utiliza en cada paso constructivo una función aleatoria que interrelaciona el nivel de feromona y el nivel de atracción heurístico de los tramos adyacentes. Las funciones utilizadas por la metaheurística presentada son las funciones pseudo-aleatorias proporcionales introducidas en el paradigma ACS.

Para cada hormiga se seleccionan los depósitos a abrir. La colonia de clusterizado se divide en 2 etapas. La primera aplica un proceso denominado *activarClientes* que selecciona los clientes que estarán activos y los que permanecerán inactivos durante el resto de la construcción. La segunda etapa aplica un procedimiento de clusterizado para agrupar los clientes y asignarlos a depósitos. Por último, se efectúa un proceso de ruteo que construye rutas para atender a los clientes activos, iniciando y finalizando las mismas en los depósitos abiertos.

Luego de la etapa constructiva, una búsqueda local rápida es aplicada para intentar mejorar las soluciones (línea 9). La hormiga que obtuvo el mejor resultado tiene un trato diferenciado. Si el resultado es lo suficientemente bueno (de acuerdo a un parámetro general θ), se creará una copia de la hormiga ("clonado") y el procedimiento de ruteo se repetirá un número fijo de veces para evaluar otras alternativas de ruteo (líneas 15-25). Este proceso de clonado copia la información de localización y clusterizado de la hormiga,

regenerando sólo la parte de la solución correspondiente al ruteo. Al final de cada iteración, se realiza una búsqueda local profunda para mejorar la hormiga que tuvo la mejor solución de la iteración.

Finalmente, se realiza el procedimiento de actualización de feromona global utilizando para ello la mejor solución global hasta el momento y la mejor solución de la iteración.

Algoritmo 2.2 Algoritmo Principal

```
1:  Z^bs ← SolucionInicialVecinosCercanos()
2:  InicializarFeromona(ρ0L, ρ0C, ρ0R)
3:  for iter = 1...maxIter do
4:    for each ant a do
5:      activarDepositos(a, 1)
6:      activarClientes(a)
7:      clusterizar(a)
8:      rutear(a)
9:      fastLocalSearch(a)
10:     if L^a < L^bs then
11:       bestAnt ← Z^a
12:     end if
13:   end for
14:   dif ← (L^bestAnt − L^bs)/L^bs * 100
15:   if dif < θ then
16:     for iterR = 1...maxIterClonado do
17:       b ← clonar(bestAnt)
18:       borrarRuteo(b)
19:       rutear(b)
20:       fastLocalSearch(b)
21:       if L^b < L^bestAnt then
22:         bestAnt ← b
23:       end if
24:     end for
25:   end if
26:   localSearch(bestAnt)
27:   if L^bestAnt < L^bs then
28:     Z^bs ← Z^bestAnt
29:   end if
30:   actualizarGlobalFeromona(Z^bs, Z^bestAnt)
31: end for
32: return Z^bs
```

El mecanismo que se ha diseñado para construir soluciones *PC-CLRP* se basa en la generación de soluciones *CLRP* con la posibilidad de dejar de lado a algunos clientes (clientes desactivados). La posibilidad de activar/desactivar clientes está plasmada en 3 lugares diferentes del algoritmo.

Por un lado, se ha decidido modificar la función de distancia incorporando a la misma la información

de los beneficios. Como se mencionó en la sección anterior, es sumamente importante utilizar un valor de atracción heurístico que represente de la mejor manera a las distintas particularidades del problema. En el problema estudiado, además de considerarse la distancia entre nodos, es importante tener en cuenta los beneficios que los clientes otorgan. La función de distancia es utilizada en el cálculo de los valores de atracción de las fases constructivas de cada una de las colonias *ACS*. Por lo tanto, se ha propuesto modificarla para que cada movimiento constructivo se oriente hacia soluciones eficientes del problema. Los detalles de esta modificación serán presentados luego en la sección *Vecindades* 2.7.

Las otras estrategias consideradas se ubican en la colonia de clusterizado. Esta colonia ha sido dividida en 2 etapas. Primero, se seleccionan los clientes a activar/desactivar y luego se efectúa el clusterizado de los clientes activos. Al efectuar el clusterizado, cuando un depósito abierto no puede incorporar más clientes activos, se verifica si alguno de los clientes desactivados en la primera etapa pueden acomodarse en los depósitos abiertos. Si es así, se activa el cliente correspondiente y se lo asigna al cluster del depósito abierto con espacio disponible.

Por último, se han incorporado nuevos operadores de búsqueda local que permiten activar o desactivar clientes en el proceso de mejoramiento por búsqueda local. En las siguientes secciones se describen los detalles de la metaheurística implementada.

2.3. Localización ACS

El *ACS* de localización selecciona cuáles depósitos abrir. El número de depósitos a abrir inicialmente es un parámetro del procedimiento y es fijado en el algoritmo principal. Se testearon varias alternativas, pero finalmente la estrategia elegida fue la combinación de varias. Cada 10 iteraciones del algoritmo principal se abre inicialmente un solo depósito y se van agregando después más depósitos a medida que los mismos son necesarios en la etapa de clusterizado. En el resto de las iteraciones se utiliza como número de depósitos a abrir la cantidad de depósitos abiertos en la mejor solución encontrada hasta el momento.

En todos los casos, en el procedimiento de clusterizado, si se alcanza la capacidad máxima de los depósitos activos y quedan clientes sin servir, se agrega un nuevo depósito ejecutando nuevamente este procedimiento.

La función que selecciona la próxima localización a abrir se basó en la regla aleatoria proporcional

$$p_j^\epsilon = \begin{cases} \frac{\pi_j^{\alpha_L} \eta_j^{\beta_L}}{\sum_{i \in D^\epsilon} \pi_i^{\alpha_L} \eta_i^{\beta_L}} & si\ j \in D^\epsilon \\ 0 & en\ caso\ contrario \end{cases} \tag{2.5}$$

donde π_j es el nivel de feromona para la localización j, D^ϵ es el conjunto de depósitos sin abrir en la iteración ϵ, α_L y β_L son los parámetros que determinan la influencia de π_j y η_j.

η_j es el valor de atracción o valor heurístico para el depósito j. Este número es la inversa de una función que contiene la multiplicación de 3 números: el costo relativo de apertura del depósito, la proporción entre la suma de las demandas de los clientes en el vecindario del depósito y la capacidad total del depósito, y el promedio de las distancias mínimas y máximas desde los clientes al depósito en la vecindad (N_j^k). Se define el vecindario como el conjunto de los k clientes más cercanos, siendo k un parámetro. En la sección *Vecindades* 2.7 puede encontrarse más información sobre el cálculo de los vecindarios.

$$\frac{1}{\eta_j} = \frac{O_j}{\sum_{i \in I} O_i} \times \frac{\sum_{i \in N_j^k} d_i}{W_j} \times \frac{min_{i \in N_j^k} c_{ij} + max_{i \in N_j^k} c_{ij}}{2} \tag{2.6}$$

Como se muestra en el algoritmo 2.3, una vez que el próximo depósito es elegido, se efectúa el procedimiento de actualización local de feromona.

Algoritmo 2.3 activarDepositos

Input: $ant, totalDepositosActivos$

1: **for** each $i = 1 \dots totalDepositosActivos$ **do**
2: $\quad openDepot_i \leftarrow decisionRuleLocation(ant)$
3: $\quad actualizarLocalFeromona(ant, openDepot_i)$
4: **end for**

2.4. Activar Clientes

Antes de asignar los clientes a los depósitos (clusterizado), se efectúa el procedimiento de activación de clientes (pricing) para seleccionar qué clientes quedarán activos y cuáles inactivos en la solución actual. Los clientes seleccionados como inactivos quedarán fuera de las próximas 2 fases (clusterizado y ruteo). Para implementar el algoritmo de selección de clientes (algoritmo 2.4), se utilizó la matriz de feromona del *ACS* de clusterización. Por esta razón, se ha considerado el procedimiento de activación de clientes como parte del *ACS* de clusterización. El registro de los clientes que quedaron inactivos en iteraciones previas se modeló mediante un depósito virtual adicional (depósito $totalDepositos + 1$).

El procedimiento *decisionRulePricing* se aplica para cada cliente con el fin de seleccionar el mejor depósito que se adecúa a sus necesidades utilizando información histórica y aleatoria. Si el depósito seleccionado es el depósito virtual, un valor aleatorio con distribución uniforme es calculado. Si este valor aleatorio es mayor que el parámetro general $parPricing$ (línea 5), el cliente queda inactivo en la solución actual y se ejecuta el procedimiento de actualización de feromona local correspondiente.

2.5. Clusterizado ACS

En este procedimiento se efectúa la asignación de clientes a depósitos abiertos. El algoritmo 2.5 muestra el pseudocódigo del procedimiento de clusterizado. Para cada depósito activo que no alcanzó la capacidad

Algoritmo 2.4 activarClientes

Input: *ant*

```
1:  for each i = 1...totalClientes do
2:    j ← decisionRulePricing(ant)
3:    if j = totalDepositos + 1 then
4:      c ← randomUniform(0...1)
5:      if c > parPricing then
6:        desactivarCliente(ant, i, totalDepositos + 1)
7:        actualizarLocalFeromona(ant, i, totalDepositos + 1)
8:      end if
9:    end if
10: end for
```

máxima, se selecciona el próximo cliente utilizando la regla de decisión pseudo-aleatoria proporcional (línea 8). Se definió la regla de decisión pseudo-aleatoria proporcional basándose en las ideas de Dorigo y Stützle [31]. Cuando se está en el depósito i, se selecciona el cliente j de acuerdo a la regla

$$j = \begin{cases} arg\,\text{máx}_{l \in A^{\epsilon}} \{\tau_{il}^{\alpha_C} \eta_{il}^{\beta_C}\} & si\ q \leq q_{0C} \\ J & en\ caso\ contrario \end{cases} \tag{2.7}$$

$$J = \frac{\tau_{ij}^{\alpha_C} \eta_{ij}^{\beta_C}}{\sum_{l \in A_i^{\epsilon,k}} \tau_{il}^{\alpha_C} \eta_{il}^{\beta_C}} \qquad si\ j \in A_i^{\epsilon,k} \tag{2.8}$$

donde q es una variable aleatoria uniformemente distribuida en $[0, 1]$, q_{0C} $(0 \leq q_{0C} \leq 1)$ es un parámetro y J es una variable aleatoria calculada de acuerdo a la distribución dada anteriormente. En otras palabras, con probabilidad q_{0C} se selecciona el mejor movimiento indicado por el nivel de feromona aprendido y la información heurística de los pares depósito i/cliente j en cuestión (2.7). Con probabilidad $1 - q_{0C}$ la selección se realiza utilizando la función aleatoria proporcional 2.8 utilizando los mismos niveles de feromona y heurísticos para los clientes participantes.

τ_{ij} es el nivel de feromona para el cliente j asignado al depósito i, η_{ij} es el valor heurístico para el par ij. α_C y β_C son los parámetros que determinan la influencia de τ_{ij} y η_{ij}.

A^{ϵ} es el conjunto de clientes que se encuentra sin asignar en la iteración ϵ.

$A_i^{\epsilon,k}$ es la vecindad de tamaño k de clientes sin asignar sobre el depósito i en la iteración ϵ. En la sección *Vecindades* 2.7 se presentará el método de cálculo de dicha vecindad.

El valor heurístico η_{ij} se calcula como el promedio de las distancias mínimas y máximas desde el cliente j al cluster que se está construyendo alrededor del depósito i.

Una vez que el cliente es elegido, lo agregamos al cluster correspondiente y efectuamos la actualización local de feromona (líneas 9 y 10). En los casos donde no se puede elegir un nuevo cliente, chequeamos si el depósito actual alcanzó su capacidad máxima (línea 12). Este proceso se repite hasta que todos los clientes estén asignados a algún depósito.

Cuando algún cliente activo queda sin asignar al finalizar este proceso, podría ser necesario abrir un nuevo depósito (línea 19). Sin embargo, antes de pasar a un nuevo depósito se verifica si alguno de los clientes inactivos tiene demandas que puedan acomodarse en el espacio libre del depósito considerado (línea 17). Esto es realizado mediante el procedimiento *chequearPricing*. Es decir, solamente en este paso se considera la posibilidad de activar clientes dejados inactivos en el procedimiento *activarClientes*. Sólo puede darse esta situación si no pudo acomodarse ninguno de los clientes activos remanentes en los depósitos abiertos y queda espacio ocioso en los mismos.

Algoritmo 2.5 clusterizar

Input: *ant*

```
1:  while !clusteringCompleto(ant) do
2:    accion ← false
3:    procesado[1..totalDepositosActivos(ant)] ← false
4:    for each i = 1 ... totalDepositosActivos(ant) do
5:      if not(procesado[i]) then
6:        accion ← true
7:        dep ← depositoActivo(ant, i)
8:        if c ← pseudoDecisionRuleClustering(ant, dep) then
9:          agregarCluster(ant, dep, c)
10:         actualizarLocalFeromona(ant, dep, c)
11:       else
12:         procesado[i] ← capacidadCompleta(ant, dep)
13:       end if
14:     end if
15:   end for
16:   if not(accion) then
17:     chequearPricing(ant)
18:     if not(clusteringCompleto(ant)) then
19:       activarDepositoAdicional(ant)
20:     end if
21:   end if
22: end while
```

2.6. Ruteo ACS

La fase de ruteo es responsable de la construcción de las rutas entre los depósitos abiertos y sus clientes. Un vehículo se encargará de hacer un viaje, comenzando desde un depósito abierto, visitando una lista de clientes y volviendo, para culminar, al mismo depósito.

El procedimiento es bosquejado en el algoritmo 2.6. El algoritmo construye un ruteo completo para una hormiga y trabaja sobre un depósito activo por vez (paso 2). Se construyen a la vez distintas rutas ($1..totalRutasActivas(ant, dep)$). Para cada una de las rutas activas que se está construyendo se selecciona

un nodo candidato. Este candidato se selecciona por medio de, nuevamente, una regla de decisión pseudo-aleatoria proporcional similar a la regla de decisión usada en la fase de clusterizado. En este caso, el valor heurístico se calcula como la inversa del costo de ruteo sobre cada eje. Cada hormiga seleccionará, para cada ruta activa, el próximo nodo j (cliente o depósito) que puede ser agregado a la ruta después del último nodo i. Esto se realiza de acuerdo a la regla de decisión pseudo-aleatoria proporcional

$$j = \begin{cases} arg\,\text{máx}_{l \in R_i^{\epsilon,k}} \{\psi_{il}^{\alpha_R} \phi_{il}^{\beta_R}\} & si\ q' \leq q_{0R} \\ J & en\ caso\ contrario \end{cases} \tag{2.9}$$

$$J = \frac{\psi_{ij}^{\alpha_R} \phi_{ij}^{\beta_R}}{\sum_{l \in R_i^{\epsilon,k}} \psi_{il}^{\alpha_R} \phi_{il}^{\beta_R}} \qquad si\ j \in R_i^{\epsilon,k} \tag{2.10}$$

donde q' es una variable aleatoria uniformemente distribuida en $[0, 1]$, q_{0R} ($0 \leq q_{0R} \leq 1$) es un parámetro y J es una variable aleatoria calculada de acuerdo a la distribución dada anteriormente (2.10). q_{0R} determina la influencia relativa entre seleccionar el mejor conocimiento aprendido (2.9) y explorar nuevas alternativas (2.10).

ψ_{ij} es el nivel de feromona en el eje (i, j) (nodo j después del nodo i en la ruta), ϕ_{ij} es el valor heurístico para el mismo eje (i, j), y α_R y β_R son los parámetros que determinan la influencia relativa de ψ_{ij} y ϕ_{ij}. $R_i^{\epsilon,k}$ es el conjunto de nodos todavía no seleccionados por la hormiga actual en la vecindad de tamaño k del nodo i en la iteración ϵ. El valor heurístico ϕ_{ij} se calcula como el valor inverso del costo del eje (i, j).

Al finalizar el armado de la lista de candidatos por ruta activa se verifica, en primer lugar, si hubo tales candidatos (puede no haber candidatos si las rutas han agotado su capacidad). Si no se encontró ningún candidato, y quedan clientes por rutear, es necesario aumentar la cantidad de rutas activas (líneas 14 y 15).

En caso de haber candidatos, se usa una variable aleatoria uniformemente distribuida para seleccionar alguno. No todas las rutas pueden tener candidatos, por lo tanto, el valor de la variable aleatoria toma en cuenta sólo las rutas con candidatos. El candidato elegido se agrega a la ruta activa respectiva (líneas 17-25) y se efectúa la actualización local de feromona.

Algoritmo 2.6 rutear

Input: *ant*

```
1:  while not(ruteoCompleto(ant)) do
2:    dep ← seleccionarDeposito(ant)
3:    posSel ← 0
4:    while clientesSinRutear(ant, dep) > 0 do
5:      candidatos ← ∅
6:      for each i = 1 ... totalRutasActivas(ant, dep) do
7:        u ← ultimoRutaActiva(ant, i)
8:        if c ← pseudoDecisionRuleRouting(ant, u, i) then
9:          candidatos_i ← c
10:         posSel ← posSel + 1
11:       end if
12:     end for
13:     if posSel = 0 and not(ruteoCompleto(ant)) then
14:       totalRutasActivasInc(ant, dep)
15:       ruta ← inicializarRuta(totalRutasActivas(ant, dep))
16:     else
17:       r ← randomUniform(0 ... posSel)
18:       for each i = 1 ... totalRutasActivas(ant, dep) do
19:         r ← r − 1
20:         if r < 0 then
21:           u ← ultimoRutaActiva(ant, ruta)
22:           agregarNodoRutaActiva(ant, i, candidatos_i)
23:           actualizarLocalFeromona(edge(candidatos_i, u))
24:         end if
25:       end for
26:     end if
27:   end while
28: end while
```

Este proceso se repite hasta que todos clientes clusterizados sean asignados a rutas desde los depósitos abiertos.

2.7. Vecindades

La fases constructivas de cada una de las colonias *ACS* utilizan funciones aleatorias para elegir los nuevos elementos a incorporar. Estas funciones utilizan la información de la feromona y cierta información heurística (valor de atracción) sobre cada uno de los elementos candidatos. La definición de estos elementos se realiza utilizando un conjunto acotado de vecinos cercanos basados en una función de distancia modificada. Las funciones pseudo-aleatorias analizan una vecindad de tamaño k, siendo k un parámetro del algoritmo.

La función de distancia c_{ij} definida sobre los distintos ejes (i, j) del grafo G utilizado (definida en 1.1)

se modificó para modelar el problema. Esta modificación tiene como objetivo incorporar las ganancias a los costos de los ejes. De esta forma, cuando se construye una nueva solución, la información de las ganancias es considerada indirectamente. Como resultado, un cliente ubicado a menor distancia y que produzca mayores ganancias se considerará más atractivo que otros. Esto es calculado de la siguiente manera:

$$c'_{ij} = c_{ij} - \frac{b_i}{2} - \frac{b_j}{2} \qquad i, j \in J \tag{2.11}$$

siendo c_{ij} el costo del eje (i, j), b_i y b_j las ganancias de visitar a los clientes i y j. Esta modificación sólo es aplicada a los ejes inter-clientes. En el caso de los ejes depósito-cliente restaremos solamente el costo del beneficio del cliente respectivo:

$$c'_{ij} = c_{ij} - \frac{b_j}{2} \qquad i \in I \quad j \in J \tag{2.12}$$

Para comprender la idea subyacente a la modificación, basta observar que todo eje que forme parte de una ruta implicará que sus nodos adyacentes formen parte de la solución. Como cada nodo cliente seleccionado tendrá un eje de entrada y otro de salida activados, se puede restar la mitad de su ganancia al costo de los ejes, obteniendo la nueva función de distancia buscada.

Proposición 2.1. *La minimización de la suma de los costos modificados c' es equivalente a la minimización de los costos originales c menos los costos de los beneficios b.*

Demostración. La función objetivo definida en la sección 1.1, establece que se quiere minimizar

$$min \sum_{e \in A} c_e x_e + \sum_{i \in I} O_i z_i + \sum_{i \in I} \sum_{e \in \delta^+(\{i\})} F x_e - \sum_{i \in J} b_i \sum_{e \in \delta^+(\{i\})} x_e \tag{2.13}$$

Por definición $A = \{(i, j) : i \neq j \wedge i, j \in J\} \cup \{(i, j) : i \in I, j \in J\} \cup \{(j, i) : i \in I, j \in J\}$. Es decir, A contiene 2 conjuntos, los ejes inter-clientes y los ejes depósito-clientes. Por lo tanto,

$$\sum_{e \in A} c_e x_e = \sum_{\substack{i,j \in J \\ i<j}} c_{ij} x_{ij} + \sum_{i \in I, j \in J} (c_{ij} x_{ij} + c_{ji} x_{ji}) \tag{2.14}$$

Lo mismo puede hacerse con el término de las ganancias. Como cada cliente activo tiene un eje de entrada y otro de salida (exigido por las ecuaciones de grado (1.3)), puede reescribirse las ganancias como:

$$\sum_{i \in J} b_i \sum_{e \in \delta^+(\{i\})} x_e = \sum_{i \in J} \frac{b_i}{2} \sum_{e \in \delta(\{i\})} x_e = \sum_{i \in J} \frac{b_i}{2} (\sum_{j \in J} x_{ij} + \sum_{j \in J} x_{ji} + \sum_{j \in I} x_{ij} + \sum_{j \in I} x_{ji}) \tag{2.15}$$

De 2.15 se deduce que cada eje (i, j) con $i, j \in J$ se contabiliza 2 veces. Una vez para el nodo i y otra para el nodo j. Por lo tanto, se puede reescribir el término del lado derecho de 2.15 como:

$$\sum_{\substack{i,j \in J \\ i<j}} (\frac{b_i}{2} + \frac{b_j}{2}) x_{ij} + \sum_{i \in J} \frac{b_i}{2} (\sum_{j \in I} x_{ij} + \sum_{j \in I} x_{ji}) \tag{2.16}$$

Combinando los costos 2.14 y los beneficios 2.16 se obtiene:

$$\begin{aligned} &\sum_{\substack{i,j \in J \\ i<j}} c_{ij} x_{ij} + \sum_{i \in I, j \in J} (c_{ij} x_{ij} + c_{ji} x_{ji}) - \sum_{\substack{i,j \in J \\ i<j}} (\frac{b_i}{2} + \frac{b_j}{2}) x_{ij} - \sum_{i \in J} \frac{b_i}{2} (\sum_{j \in I} x_{ij} + \sum_{j \in I} x_{ji}) = \\ &\sum_{\substack{i,j \in J \\ i<j}} c_{ij} x_{ij} - \sum_{\substack{i,j \in J \\ i<j}} (\frac{b_i}{2} + \frac{b_j}{2}) x_{ij} + \sum_{i \in I, j \in J} (c_{ij} x_{ij} + c_{ji} x_{ji}) - \sum_{i \in J, j \in I} \frac{b_i}{2} x_{ij} - \sum_{i \in J, j \in I} \frac{b_i}{2} x_{ji} = \\ &\sum_{\substack{i,j \in J \\ i<j}} (c_{ij} - \frac{b_i}{2} - \frac{b_j}{2}) x_{ij} + \sum_{i \in I, j \in J} (c_{ij} - \frac{b_j}{2}) x_{ij} + \sum_{i \in I, j \in J} (c_{ji} - \frac{b_j}{2}) x_{ji} \end{aligned} \tag{2.17}$$

Ahora se puede reemplazar 2.17 por las definiciones anteriores de c' y se obtiene lo buscado:

$$\sum_{\substack{i,j \in J \\ i<j}} c'_{ij} x_{ij} + \sum_{i \in I, j \in J} c'_{ij} x_{ij} + \sum_{i \in I, j \in J} c'_{ji} x_{ji} = \sum_{e \in A} c'_e x_e$$

□

2.8. Búsqueda Local

Dos procedimientos diferentes de búsqueda local fueron diseñados para intentar mejorar las soluciones obtenidas en la iteración. El primer procedimiento, llamado $fastLocalSearch$, efectúa un proceso de búsqueda local rápido usando sólo un subconjunto pequeño de operadores de búsqueda local (línea 9 del algoritmo 2.2). El segundo implementa un procedimiento mucho más complejo y costoso para tratar de conseguir la mejor solución posible de la iteración. Se aplica únicamente a la mejor solución obtenida en la iteración (línea 26 del algoritmo 2.2).

Los operadores usados pueden categorizarse en 3 grupos fundamentales: 1) intra-ruta, 2) intra-cluster, 3) intra-depósito. El primer grupo realiza sólo movimientos dentro de la misma ruta. El segundo grupo de operadores mueve nodos dentro del mismo cluster. Finalmente, el grupo intra-depósitos sólo hace movimientos de clientes de depósito a depósito.

Se han utilizado operadores de búsqueda local tradicionales y se han desarrollado algunos específicos para el problema *PC-CLRP*. Los operadores implementados por grupo son:

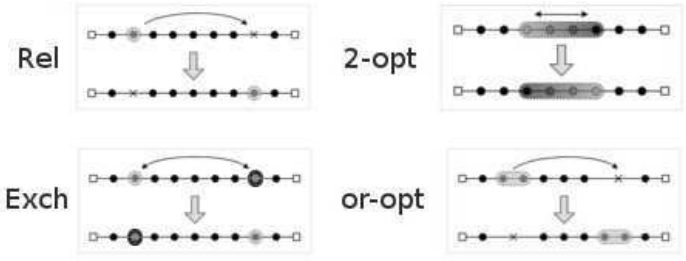

Figura 2.1: Operadores de Búsqueda Local Intra-ruta

- Intra-ruta: *Relocate*, *Exchange*, *OR-opt*, *2-opt*
- Intra-cluster: *Relocate*, *Exchange*, *2-opt*, *Cross-Exchange*, *Maximize-Served*, *Minimize-Served*, *Exchange-Served*
- Intra-depósito: *Relocate*, *Exchange*, *Route-Change*, *Depot-Erase*

En la figura 2.1 puede verse un ejemplo de los movimientos dentro de una ruta. En la figura 2.2 se observan ejemplos de los movimientos Intra-cluster tradicionales. Los operadores *Relocate* y *Exchange* Intra-depósito funcionan de manera similar pero considerando exclusivamente movimientos entre depósitos diferentes.

Los nuevos operadores diseñados para considerar la variante Prize-Collecting del problema son (figura 2.3):

Maximize-Served: intenta incorporar clientes que se encuentran inactivos a la solución.

Minimize-Served: intenta desactivar clientes (pasándolos al conjunto de clientes no clusterizados).

Exchange-Served: intercambia un cliente activo por otro no activo (clusterizado por no clusterizado).

Route-Change: intenta mover una ruta completa de un depósito a otro (modificando a su vez los clusters respectivos).

Depot-Erase: verifica si algún depósito tiene costos que superan las ganancias. Se consideran todos los costos, incluyen costos de apertura del depósito, costo por uso de vehículos y costos de ruteo. Si es así, se elimina el depósito con todas sus rutas.

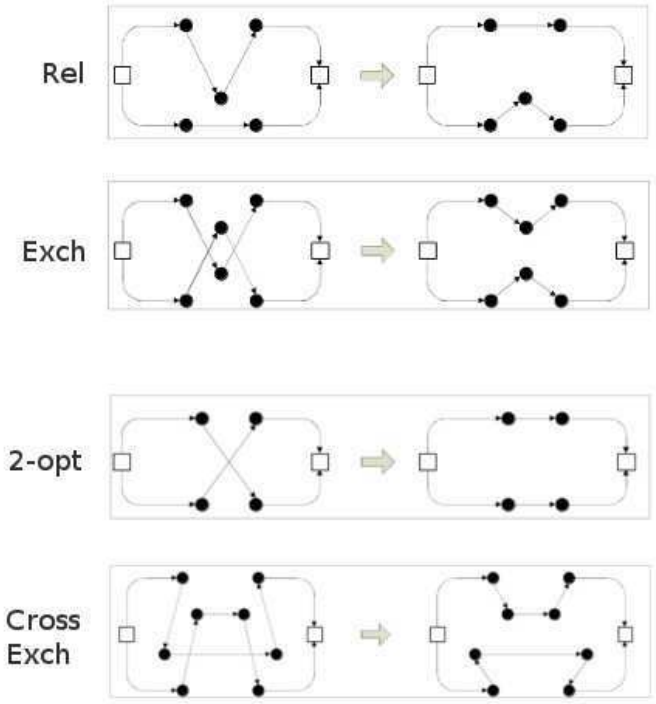

Figura 2.2: Operadores de Búsqueda Local Intra-cluster

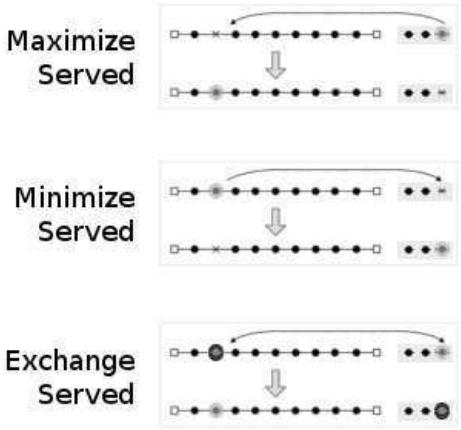

Figura 2.3: Operadores de Búsqueda Local Prize-Collecting

Todos los operadores utilizados generan un listado de los posibles movimientos y seleccionan el que mayor disminución provoque en la función objetivo. Los movimientos que son incompatibles son descartados.

El orden de ejecución de los diferentes operadores es el siguiente:

- $fastLocalSearch$: *intra-depósito Relocate*, *intra-depósito Exchange*, *intra-depósito Route-Change*, *Maximize-Served*, *Minimize-Served*, *Exchange-Served*, *intra-ruta Cross-Exchange* (limitado a cadenas de 2 nodos), *intra-ruta Relocate*, *intra-ruta Exchange*, *intra-ruta 2-opt*, *intra-ruta OR-opt*
- *localSearch* completo: *Maximize-Served*, *Minimize-Served*, *Exchange-Served*, *intra-depósito Relocate*, *intra-cluster Relocate*, *intra-depósito Exchange*, *intra-cluster Exchange*, *intra-cluster 2-opt*, *intra-ruta Relocate*, *intra-ruta Exchange*, *intra-ruta 2-opt*, *intra-ruta OR-opt*, *intra-ruta Cross-Exchange* (limitado a cadenas de 4 nodos), *intra-depósito Route-Change*, *Depot-Erase*

2.9. Procedimientos de actualización de feromona

Los 3 *ACS* descriptos anteriormente ejecutan procedimientos de actualización local de feromona una vez que terminaron con el proceso constructivo.

La idea detrás del procedimiento de actualización local de feromona es la de hacer menos atractivos a los elementos seleccionados con el fin de favorecer la diversificación de futuras soluciones. Luego, al finalizar la construcción de una solución completa se ejecuta un procedimiento de actualización global de feromona para registrar la información más relevante de la iteración (línea 30 Algoritmo 2.2). En todos los casos, se testearon diferentes estrategias y parámetros, siendo los reportados los que mejores resultados han proporcionado.

2.9.1. Inicialización

La inicialización de las matrices de feromona se realiza con valores calculados en base a la solución inicial construida con la heurística de Vecinos Más Cercanos (sección 2.2).

En el caso de la matriz de feromona del *ACS* de localización, la inicialización es $\pi_j^0 = 10/O_1 \quad \forall j \in I$ siendo O_1 el costo de apertura del depósito 1.

La matriz del *ACS* de clusterizado se inicializa como $\tau_{ij}^0 = \tau_0 \quad \forall i \in I, j \in J$ siendo $\tau_0 = 1/(D_i * C_i * 1000000)$ donde D_i y C_i son la distancia total y la cantidad de clusters de la solución inicial (calculada con la heurística de Vecinos Más Cercanos).

Por último, la matriz del *ACS* de ruteo es inicializada como $\psi_{ij}^0 = 1/(D_i * C_i) \quad \forall (i,j) \in A$ siendo D_i y C_i la distancia total y la cantidad de clusters (o depósitos abiertos) de la solución inicial.

2.9.2. Actualizaciones locales

El procedimiento de actualización local de feromona para la etapa ACS de localización es implementado mediante la siguiente ecuación:

$$\pi_j^{\epsilon+1} = (1-\rho_L)\pi_j^{\epsilon} + \rho_L \pi_0 \qquad \text{si la localización } j \in Z_h^{\epsilon} \tag{2.18}$$

Donde π_j^{ϵ} es el valor de la matriz de feromona para la localización j en la iteración ϵ, Z_h^{ϵ} refiere a la solución construida por la hormiga h en la iteración ϵ, $0 \leq \rho_L \leq 1$ es la tasa de evaporación de feromona. El parámetro ρ_L se usa para evitar la acumulación ilimitada de feromona. El valor τ_0 ha sido obtenido mediante experimentación y se calcula como $\tau_0 = 1/(O_j * 10000)$ (valor muy pequeño) usando el costo de apertura del depósito O_j.

Los procedimientos correspondientes a las actualizaciones locales de feromona para las etapas de clusterizado y ruteo utilizan un enfoque similar.

La etapa de clusterizado usa la siguiente ecuación cuando el cliente j pertenece a la localización i:

$$\tau_{ij}^{\epsilon+1} = (1-\rho_C)\tau_{ij}^{\epsilon} + \rho_C \tau_0 \qquad si\ (i,j) \in Z_h^{\epsilon} \tag{2.19}$$

Donde τ_{ij}^{ϵ} es el valor de la matriz de feromona para la localización i y el cliente j en la iteración ϵ, $0 \leq \rho_C \leq 1$ es la tasa de evaporación de feromona y τ_0 el mismo valor utilizado para la inicialización. En la etapa de Pricing ($activarClientes$) se aplica también la ecuación anterior a la localización virtual adicional descripta en la sección de $activarClientes$ 2.4.

En el caso de la colonia ACS de ruteo se usa la siguiente ecuación:

$$\psi_{ij}^{\epsilon+1} = (1-\rho_R)\psi_{ij}^{\epsilon} + \rho_R \psi_0 \qquad \text{si el eje } (i,j) \in Z_h^{\epsilon} \tag{2.20}$$

Donde ψ_{ij}^{ϵ} es el valor de la matriz de feromona para el eje (i,j) en la iteración ϵ, $0 \leq \rho_R \leq 1$ es la tasa de evaporación de feromona y ψ_0 es calculado como $1/nC$, siendo nC la cantidad de clientes asignados al cluster que se está ruteando. La actualización local es efectuada sobre el eje (i,j) y también sobre el eje de dirección inversa (j,i). La dirección no es importante cuando se actualiza la feromona, ya que se podría invertir la dirección de toda la ruta y obtener una solución equivalente. Por eso se actualizan ambas direcciones.

2.9.3. Actualizaciones globales

Finalmente, para el procedimiento de actualización global se utiliza una variante de la estrategia elitista propuesta en los trabajos de Dorigo, D. Maniezzo, V. Maniezzo y Colorni [30, 32, 33, 68]. La estrategia elitista proporciona refuerzos adicionales a los rastros que pertenecen a la mejor solución encontrada desde el inicio del algoritmo (Z^{bs}). Dicha estrategia trata de orientar la búsqueda de futuras hormigas hacia soluciones que incluyan algunos de los rastros de la mejor solución conocida hasta el momento. En el algoritmo presentado, las soluciones elitistas utilizadas son la mejor solución global conocida (Z^{bs}) y la mejor solución de la iteración (Z^{it}). Los rastros pertenecientes a esas soluciones incrementarán sus valores de feromona, mientras que los que no estén en las soluciones elitistas decrecerán su nivel de feromona. El procedimiento global se aplica secuencialmente para actualizar la feromona de las 3 colonias ACS.

La ecuación utilizada para la etapa ACS de localización es

$$\pi_j^{\epsilon+1} = (1-\rho_L)\pi_j^{\epsilon} + \rho_L \Delta\pi_j \tag{2.21}$$

donde

$$\Delta\pi_j = \begin{cases} (L_w - L_b + L_w - L_s)/L_w * nC & si\ j \in Z^{bs} o\ Z^{it} \\ 0 & \text{en caso contrario} \end{cases} \tag{2.22}$$

y L_b, L_w y L_s son los costos totales de la mejor solución global Z^{bs}, de la peor solución y de la mejor solución de la iteración Z^{it} respectivamente y nC es el tamaño del cluster j.

El procedimiento de actualización de feromona global para las etapas de clusterizado y ruteo utilizan un enfoque similar. La colonia de clusterizado utiliza las ecuaciones

$$\tau_{ij}^{\epsilon+1} = (1-\rho_C)\tau_{ij}^{\epsilon} + \rho_C \Delta\tau_{ij} \qquad si\ (i,j) \in Z_h^{\epsilon} \tag{2.23}$$

con

$$\Delta\tau_{ij} = \begin{cases} (L_w - L_b + L_w - L_s) * nC * 10/L_w & si\ (i,j) \in Z^{bs} o\ Z^{it} \\ 0 & \text{en caso contrario} \end{cases} \tag{2.24}$$

La colonia de ruteo usa las ecuaciones

$$\psi_{ij}^{\epsilon+1} = (1-\rho_R)\psi_{ij}^{\epsilon} + \rho_R \Delta\psi_{ij} \qquad \text{si el eje } (i,j) \in Z_h^{\epsilon} \tag{2.25}$$

con

$$\Delta\psi_{ij} = \begin{cases} (L_w - L_b + L_w - L_s)/L_w & \text{si el eje } (i,j) \in Z^{bs} o\ Z^{it} \\ 0 & \text{en caso contrario} \end{cases} \tag{2.26}$$

2.10. Resultados Computacionales

En esta sección se describirá la implementación del algoritmo así como los resultados obtenidos utilizando un conjunto de instancias de la literatura para el *CLRP* y un nuevo conjunto de instancias generadas a partir de las anteriores.

El algoritmo de optimización de colonia de hormigas fue implementado en C++ usando el compilador GNU C++ versión 4.6.3. Los tests fueron corridos en una máquina con procesador Intel Core i7-3770, 3.40 Ghz con 16GB de memoria bajo el sistema operativo Linux kernel 3.8.

Un conjunto de experimentos computacionales se corrieron para determinar los mejores seteos de parámetros. Las opciones analizadas para cada parámetro incluyeron los valores $nAnts \in \{4, 7, 10, 20\}$ $maxIter \in \{15, 40, 80, 100\}$, $maxiterClonado \in \{N, N/2, N/4, N/8, N/10\}$, $\theta \in \{0{,}1, 0{,}3, 0{,}5, 0{,}7, 0{,}9\}$, $\beta_L, \beta_C, \beta_R \in \{1, 2, 3, 4, 5\}$, $q_{0C}, q_{0R} \in \{0{,}1, 0{,}3, 0{,}5, 0{,}7, 0{,}9\}$, $\rho_L, \rho_C, \rho_R \in \{0{,}1, 0{,}3, 0{,}5, 0{,}7, 0{,}9\}$, $k \in \{5, 10, 20, N\}$ y $parPricing \in \{0{,}3, 0{,}6, 0{,}8\}$. N refiere a la cantidad total de clientes. Los parámetros α_L, α_C y α_R han sido fijados al valor 1. Los parámetros α y β determinan la influencia relativa de los niveles de feromona y de los valores heurísticos. Se ha decidido testear la influencia de ambos parámetros fijando los parámetros α y variando sólo los parámetros β.

Para elegir el mejor conjunto de parámetros se ha utilizado el software *Iterated Racing for Automatic Algorithm Configuration Software - IRACE* versión 1.06 [65]. Este software aplica el procedimiento denominado *iterated racing* para intentar encontrar la mejor configuración a un problema de optimización. Se configuran todos los parámetros del algoritmo de optimización y el rango de valores permitidos para cada uno. El algoritmo *IRACE* evalúa de manera iterativa distintas combinaciones posibles de parámetros y va seleccionando los mejores candidatos. Finalmente devuelve una lista con las mejores combinaciones encontradas.

Utilizando las mejores configuraciones encontradas por *IRACE*, se efectuaron diversos experimentos para seleccionar el mejor conjunto de parámetros. La elección final es una combinación de varias de las opciones propuestas por el paquete *IRACE*, ya que la misma no fue identificada directamente como candidata. Para testear las distintas configuraciones se han fijado todos los parámetros en alguno de los valores propuestos por *IRACE* menos uno (el que se quería analizar). El parámetro analizado fue variado con un subconjunto de valores prometedores de las opciones anteriores. Este procedimiento se ha repetido para cada uno de los parámetros.

Los parámetros que obtuvieron los mejores resultados en promedio son: $nAnts = 4$, $maxIter = 100$, $maxIterClonado = \text{máx}(N/4, 10)$, $\theta = 0{,}5$, $\alpha_L = 1$, $\alpha_C = 1$, $\alpha_R = 1$ (L:Localización, C:Clusterizado,

R:Ruteo), $\beta_L = 1$, $\beta_C = 5$, $\beta_R = 2$, $q_{0C} = 0{,}9$, $q_{0R} = 0{,}9$, $\rho_L = 0{,}7$, $\rho_C = 0{,}9$, $\rho_R = 0{,}3$, $k = 10$ y $parPricing = 0{,}8$. Estos parámetros han sido los utilizados para testear todas las instancias de prueba.

De acuerdo a la revisión bibliográfica efectuada, *PC-CLRP* no fue estudiado hasta el momento. Debido a ello, no se han podido conseguir instancias específicamente diseñadas para este problema. Para poder chequear el algoritmo propuesto, se utiliza un conjunto de 30 instancias para el problema general *CLRP* (instancias de Prodhon [86]) y se generan nuevas instancias con la información adicional requerida. Las instancias para el problema *CLRP* se encuentran detalladas en el apéndice A.1. El algoritmo utilizado para la generación de las nuevas instancias se detalla en el apéndice A.2. Las nuevas instancias pueden descargarse del sitio web `http://www.dc.uba.ar/People/negrotto`.

Antes de pasar a mostrar los resultados generales del algoritmo, se reportarán parte de los experimentos efectuados para analizar el funcionamiento de la metaheurística. Para llevar a cabo estos experimentos se ha utilizado un subconjunto de 15 instancias de pequeño y mediano tamaño, tomadas del conjunto de 60 nuevas instancias basadas en las instancias de Prodhon mencionadas anteriormente (subconjunto de instancias con 20 y 50 clientes).

Para poder analizar el impacto real de las diferentes estrategias propuestas, en cada uno de los experimentos, se ha desactivado la búsqueda local profunda (procedimiento *localSearch*) y los operadores *intra-depósito* del procedimiento $fastLocalSearch$. La idea es aplicar solamente los operadores de búsqueda local más simples para arreglar mínimamente las soluciones construidas. De esta forma, las soluciones resultantes no están tan influenciadas por la etapa de búsqueda local y pueden observarse mejor las variaciones de los parámetros o los cambios en las estrategias propuestas como experimento.

Estrategia de apertura de depósitos

El primer experimento reportado busca mostrar la incidencia de las distintas decisiones analizadas para el ACS de localización. Como se mencionó anteriormente en la sección 2.3, se utilizó una estrategia combinada para seleccionar la cantidad de depósitos a abrir inicialmente. Normalmente, se usa como parámetro del procedimiento 2.3 la cantidad de depósitos abiertos de la mejor solución encontrada al momento. Cada 10 iteraciones, en cambio, se llama al procedimiento seleccionando la apertura de un sólo depósito. Se testearon numerosas alternativas antes de definir esta estrategia. Se reportan las principales variantes analizadas:

(I) Seleccionar siempre un solo depósito

(II) Seleccionar siempre la cantidad de depósitos de la mejor solución

(III) Seleccionar siempre la cantidad de depósitos de la mejor solución -1

(IV) Estrategia que combina opciones anteriores (elegida)

En todos los casos se reporta la información de la mejor solución obtenida después de efectuar 10 corridas independientes por instancia. Se reporta el costo, el número de la menor corrida donde se obtuvo el resultado (run), y el *gap* a la mejor solución conocida para la instancia (*Best Known Solution* - *BKS*) calculando $gap = 100 * (costo - BKS)/BKS$. La solución *BKS* es la solución obtenida por el algoritmo de colonia de hormigas completo utilizando los procedimientos originales de búsqueda local con todos los operadores activos. Dichos resultados serán mostrados luego en el cuadro 2.6. Por su parte, el cuadro 2.1 muestra los resultados obtenidos para cada una de las estrategias anteriores.

Instancia	(I)			(II)			(III)			(IV)		
	Costo	Run	Gap	Costo	Run	Gap	Costo	Run	Gap	Costo	Run	Gap
pc20-5-1-l	-32371	7	-3.41	-31839	3	-5.00	-32371	7	-3.41	-31839	3	-5.00
pc20-5-2-s	-2971	0	0.00	-2393	2	-19.45	-2971	0	0.00	-2971	0	0.00
pc20-5-2-l	-44667	8	-0.12	-44667	3	-0.12	-44667	8	-0.12	-44725	4	0.00
pc50-5-1-l	-38661	7	0.00	-36097	5	-6.63	-38661	7	0.00	-37989	0	-1.73
pc50-5-1b-s	-4139	4	-10.77	-4365	0	-5.90	-4139	4	-10.77	-4365	0	-5.90
pc50-5-1b-l	-61596	9	-0.92	-61324	5	-1.36	-61596	9	-0.92	-61850	8	-0.51
pc50-5-2-s	-4835	3	0.00	-1476	7	-69.47	-4835	2	0.00	-4878	4	0.88
pc50-5-2-l	-42406	1	-3.20	-38115	7	-12.99	-42557	2	-2.85	-43321	0	-1.11
pc50-5-2b-s	-4521	9	0.00	-3890	7	-13.95	-4521	4	0.00	-4521	9	0.00
pc50-5-2BIS-l	-14221	2	0.06	-14181	2	-0.21	-14110	7	-0.71	-14157	0	-0.38
pc50-5-2bBIS-l	-34461	1	-0.41	-34449	9	-0.45	-34532	6	-0.21	-34606	6	0.00
pc50-5-3-s	-5992	3	-0.95	-739	7	-87.78	-5992	3	-0.95	-6050	6	0.00
pc50-5-3-l	-37581	7	-2.39	-37852	8	-1.69	-37581	7	-2.39	-37113	7	-3.61
pc50-5-3b-s	-21223	4	0.54	-20300	6	-3.82	-21223	4	0.54	-20876	6	-1.09
pc50-5-3b-l	-57992	0	-3.65	-59429	3	-1.26	-57992	0	-3.65	-59645	4	-0.90
prom.		4.33	-1.68		4.93	-15.33		4.66	-1.69		3.80	-1.29

Cuadro 2.1: Resultados experimentación en estrategia de apertura de depósitos

Puede observarse en el cuadro que la estrategia (IV) es la que mejor desempeño ofrece en promedio. Además, las mejores soluciones son, en la mayor parte de los casos, encontradas dentro de las primeras corridas. Esta última observación es también importante ya que muestra el esfuerzo potencial que el algoritmo tuvo que realizar para encontrar las mejores soluciones.

Clonado de hormigas

El segundo experimento tiene como objetivo verificar la eficacia del proceso de "clonado" efectuado en el algoritmo principal 2.2. Como se mencionó anteriormente, dicho proceso se efectúa inmediatamente después de terminar la etapa constructiva. El proceso de "clonado" copia la información de localización y clusterizado de la mejor hormiga de la iteración para intentar explorar diferentes alternativas de ruteo. En este experimento se analizan 5 alternativas:

(I) Desactivar el proceso de clonado

(II) Aplicar el proceso de clonado $maxIterClonado = 1/2 * \text{máx}(N/4, 10)$ veces

(III) Aplicar el proceso de clonado $maxIterClonado = max(N/4, 10)$ veces (elegida)

(IV) Aplicar el proceso de clonado $maxIterClonado = 2 * \text{máx}(N/4, 10)$ veces

(V) Aplicar el proceso de clonado $maxIterClonado = 4 * \text{máx}(N/4, 10)$ veces

En todos los casos se reporta la información de la mejor solución obtenida después de efectuar 10 corridas independientes por instancia. Se reporta el tiempo de ejecución en segundos y el gap a la mejor solución conocida BKS.

El cuadro 2.2 muestra los resultados obtenidos para cada una de las estrategias anteriores. Al final de la tabla puede observarse el promedio de los tiempos y de los gaps para cada una de las estrategias.

Instancia	(I)		(II)		(III)		(IV)		(V)	
	Tiempo	Gap	Tiempo	Gap	Tiempo	Gap	Tiempo	Gap	Tiempo	Gap
pc20-5-1-l	1.61	-2.88	1.92	-5.00	2.53	-5.00	3.09	-5.00	5.13	-2.88
pc20-5-2-s	1.63	0.00	1.94	-19.45	2.43	0.00	3.21	0.00	4.95	0.00
pc20-5-2-l	1.53	-1.42	1.88	-0.55	2.31	0.00	3.01	0.00	4.99	0.00
pc50-5-1-l	13.11	-8.73	18.47	-2.97	23.52	-1.73	36.05	-4.26	68.91	-3.29
pc50-5-1b-s	21.33	-15.04	31.43	-11.16	49.02	-5.90	95.06	-5.90	164.99	-3.14
pc50-5-1b-l	25.22	-3.94	47.25	-1.30	65.33	-0.51	115.87	-0.72	235.68	-0.23
pc50-5-2-s	9.69	-3.12	14.60	0.88	17.45	0.88	26.94	0.88	50.20	0.88
pc50-5-2-l	11.94	-5.27	15.50	0.00	18.87	-1.11	25.74	-1.36	45.69	-1.19
pc50-5-2b-s	19.31	-0.77	31.05	-2.07	39.59	0.00	64.03	0.00	134.96	0.00
pc50-5-2BIS-l	10.42	-4.43	16.37	0.00	19.69	-0.38	24.75	0.56	52.23	0.16
pc50-5-2bBIS-l	16.12	-6.85	25.19	-0.70	33.89	0.00	51.13	0.00	106.38	0.00
pc50-5-3-s	10.55	0.00	13.88	-0.95	18.41	0.00	25.55	0.00	52.39	0.00
pc50-5-3-l	12.05	-3.85	15.92	-4.97	19.86	-3.61	27.89	-2.39	52.22	-1.09
pc50-5-3b-s	19.40	-4.78	27.94	0.00	39.56	-1.09	58.26	0.54	127.64	0.00
pc50-5-3b-l	21.42	-3.10	34.52	-0.49	44.50	-0.90	70.41	-0.90	155.28	-0.39
prom.	13.02	-4.28	19.86	-3.25	26.46	-1.29	42.07	-1.24	84.11	-0.74

Cuadro 2.2: Resultados experimentación Clonación

La estrategia elegida debe balancear la calidad de las soluciones obtenidas versus el tiempo empleado en generar "clonaciones". Analizando las distintas opciones, y considerando que en el algoritmo final se utilizan los procedimientos de búsqueda local completos, se ha decidido seleccionar la estrategia (III) que obtiene, en la mayor parte de las instancias, buenas soluciones a tiempos razonables. La estrategia (V) obtiene mejores resultados en los promedios y en la mayor parte de las instancias, pero a un costo computacional mucho más alto.

Evaluación impacto del procedimiento activarClientes

El tercer y último experimento reportado tiene como objetivo mostrar el impacto de la utilización del procedimiento *activarClientes* (definido en la sección 2.4). Como se mencionó en la correspondiente sección, el procedimiento activa/desactiva a los clientes durante la construcción de una solución. Este es uno

de los mecanismos introducidos para afrontar las particularidades existentes en los problemas *PC-CLRP*.

Se reporta en este experimento el contraste entre la desactivación del procedimiento versus la utilización del mismo. Las alternativas consideradas son notadas como:

(I) No aplicar procedimiento *activarClientes*

(II) Aplicar procedimiento *activarClientes* con $parPricing = 0{,}9$

(III) Aplicar procedimiento *activarClientes* con $parPricing = 0{,}8$ (elegida)

(IV) Aplicar procedimiento *activarClientes* con $parPricing = 0{,}6$

(V) Aplicar procedimiento *activarClientes* con $parPricing = 0{,}3$

El cuadro 2.3 reporta la información de las mejores soluciones obtenidas después de efectuar 10 corridas independientes por instancia. Se reporta el costo y el gap a la mejor solución conocida *BKS* para la instancia.

Instancia	(I)		(II)		(III)		(IV)		(V)	
	Costo	Gap	Costo	Gap	Costo	Gap	Costo	Gap	Costo	Gap
pc20-5-1-l	-31949	-4.67	-31839	-5.00	-31839	-5.00	-32852	-1.98	-32726	-2.35
pc20-5-2-s	-2480	-16.52	-2971	0.00	-2971	0.00	-2971	0.00	-2971	0.00
pc20-5-2-l	-44725	0.00	-44667	-0.12	-44725	0.00	-44575	-0.33	-44575	-0.33
pc50-5-1-l	-38192	-1.21	-37218	-3.73	-37989	-1.73	-36421	-5.79	-36809	-4.79
pc50-5-1b-s	-4365	-5.90	-4442	-4.24	-4365	-5.90	-4121	-11.16	-4365	-5.90
pc50-5-1b-l	-61721	-0.72	-61403	-1.23	-61850	-0.51	-60927	-2.00	-62173	0.00
pc50-5-2-s	-4768	-1.38	-4835	0.00	-4878	0.88	-4835	0.00	-4835	0.00
pc50-5-2-l	-43326	-1.10	-42985	-1.88	-43321	-1.11	-42801	-2.30	-42997	-1.85
pc50-5-2b-s	-4067	-10.04	-4521	0.00	-4521	0.00	-4521	0.00	-4486	-0.77
pc50-5-2BIS-l	-13834	-2.65	-13906	-2.15	-14157	-0.38	-14074	-0.97	-14292	0.56
pc50-5-2bBIS-l	-34362	-0.70	-34362	-0.70	-34606	0.00	-34606	0.00	-34606	0.00
pc50-5-3-s	-6050	0.00	-5992	-0.95	-6050	0.00	-5820	-3.80	-6050	0.00
pc50-5-3-l	-37409	-2.84	-37489	-2.63	-37113	-3.61	-38504	0.00	-37381	-2.91
pc50-5-3b-s	-20811	-1.40	-20917	-0.90	-20876	-1.09	-21038	-0.32	-21038	-0.32
pc50-5-3b-l	-59321	-1.44	-58791	-2.32	-59645	-0.90	-59745	-0.74	-59243	-1.57
prom.		-3.30		-1.72		-1.29		-1.96		-1.35

Cuadro 2.3: Resultados experimentación procedimiento *activarClientes*

Como puede observarse en la tabla, el impacto de la utilización del procedimiento es positivo, logrando una reducción importante en el promedio de los gaps. Dentro de las distintas opciones analizadas, utilizar $parPricing = 0{,}8$ (opción III) fue la que mejores resultados ofreció.

Comparación de resultados con trabajos para CLRP

Como ya se mencionó, no se ha encontrado trabajos sobre el problema *PC-CLRP*. Por ello, como paso previo al testeo de las instancias Prize-Collecting, se ha verificado el algoritmo propuesto con las instancias de Prodhon generales y se lo ha comparado con otros trabajos publicados para *CLRP*. Esta comparación permite tener una idea aproximada de la calidad de las soluciones obtenidas para el problema particular

CLRP y verifica que el funcionamiento general del algoritmo sea correcto. Para poder procesar las instancias *CLRP* con el algoritmo diseñado para problemas *PC-CLRP*, se han fijado las ganancias de los clientes con valores muy grandes de manera de forzar la visita de los mismos (simulando el problema general *CLRP*).

Se han comparado los resultados obtenidos con algunas de las últimas heurísticas publicadas en la literatura para *CLRP*: MA|PM en Prins y otros [83]; LRGTS en Prins y otros [85]; SALRP en Yu y otros [104]; *GRASP*xELS en Duhamel y otros [18] (GxE), *Multiple Ant Colony Optimization* - *MACO* en Ting y otros [92] y 2-HGTS en Escobar y otros [35]. Otros trabajos recientes basados en metaheurísticas híbridas que utilizan programación lineal entera (*Integer Linear Programming* - *ILP*), denominados también como matheuristics, no han sido utilizados como comparación debido a que los mismos utilizan técnicas muy diferentes a la utilizada por las metaheurísticas tradicionales.

Los cuadros 2.4 y 2.5 muestran los resultados computacionales obtenidos por el algoritmo y la comparación con los otros trabajos publicados. Se han informado los gaps a los costos de las mejores soluciones conocidas (*BKS*) calculando $gap = 100 * (costo - BKS)/BKS$, siendo *costo* el valor de la función objetivo del nuevo resultado. Para cada instancia, los resultados y los gaps obtenidos por el algoritmo presentado informan la mejor solución generada luego de 10 corridas independientes. Los tiempos de corrida considerados indican el tiempo total de CPU obtenido al sumar los tiempos de las 10 corridas efectuadas por cada instancia. Las instancias utilizadas tienen 5 y 10 posibles localizaciones de depósitos y 20, 50, 100 y 200 clientes.

Los tiempos y los resultados informados en las otras metaheurísticas son los promedios de 5-10 corridas (salvo MA|PM y SALRP que usan una sola corrida por instancia). Es importante tener cuenta al momento de evaluar los resultados que los algoritmos comparados informan el promedio de los tiempos de las distintas corridas, mientras que los resultados presentados para el algoritmo propuesto en este trabajo se informan como la suma de los tiempos de todas las corridas.

Instancia	*BKS*	Costo	Tiempo (s)	Gap
coord20-5-1	54793	54793	4.89	0.00
coord20-5-1b	39104	39104	6.92	0.00
coord20-5-2	48908	48908	3.76	0.00
coord20-5-2b	37542	37542	6.14	0.00
coord50-5-1	90111	90111	25.22	0.00
coord50-5-1b	63242	63242	31.82	0.00
coord50-5-2	88298	88298	34.58	0.00
coord50-5-2b	67308	67449	52.57	0.21
coord50-5-2BIS	84055	84181	35.05	0.15
coord50-5-2bBIS	51822	51851	66.34	0.06
coord50-5-3	86203	86203	23.73	0.00
coord50-5-3b	61830	61830	42.30	0.00
coord100-5-1	274814	276220	171.41	0.51
coord100-5-1b	213615	215100	225.98	0.70
coord100-5-2	193671	195180	102.94	0.78
coord100-5-2b	157095	157280	128.93	0.12
coord100-5-3	200079	201237	111.68	0.58
coord100-5-3b	152441	153437	131.59	0.65
coord100-10-1	287983	297823	226.49	3.42
coord100-10-1b	231763	236649	296.66	2.11
coord100-10-2	243590	245523	175.39	0.79
coord100-10-2b	203988	205258	225.76	0.62
coord100-10-3	250882	254624	176.77	1.49
coord100-10-3b	204317	205209	248.63	0.44
coord200-10-1	477248	480195	1021.99	0.62
coord200-10-1b	378351	378378	1342.82	0.01
coord200-10-2	449571	450608	1084.96	0.23
coord200-10-2b	374330	375783	1453.01	0.39
coord200-10-3	469433	478311	1314.23	1.89
coord200-10-3b	362817	366132	1854.59	0.91
prom.			354.24	0.56

Cuadro 2.4: Resultados CLRP sobre las instancias Prodhon

Instancia	MA|PM	LRGTS	GxE	SALRP	MACO	2-HGTS	PC
coord20-5-1	0.00	0.62	0.00	0.00	0.00	0.00	0.00
coord20-5-1b	0.00	0.00	0.00	0.00	0.00	0.00	0.00
coord20-5-2	0.00	0.00	0.00	0.00	0.00	0.08	0.00
coord20-5-2b	0.00	0.00	0.00	0.00	0.00	0.00	0.00
coord50-5-1	0.05	0.05	0.00	0.00	0.00	0.32	0.00
coord50-5-1b	0.00	0.02	0.00	0.00	0.00	1.31	0.00
coord50-5-2	0.00	0.47	0.39	0.00	0.00	1.18	0.00
coord50-5-2b	0.87	0.58	0.00	0.05	0.00	1.74	0.21
coord50-5-2BIS	0.00	0.15	0.00	0.00	0.00	0.00	0.15
coord50-5-2bBIS	0.00	0.33	0.00	0.00	0.00	0.51	0.06
coord50-5-3	0.00	0.00	0.00	0.29	0.00	0.00	0.00
coord50-5-3b	0.00	0.00	0.00	1.41	0.00	0.00	0.00
coord100-5-1	2.59	1.14	0.78	0.81	0.51	0.28	0.51
coord100-5-1b	1.42	0.59	1.05	1.12	0.33	0.60	0.70
coord100-5-2	0.98	1.48	0.31	0.23	0.40	0.49	0.78
coord100-5-2b	0.15	0.44	0.18	0.04	0.08	0.11	0.12
coord100-5-3	0.83	0.94	0.13	0.08	0.48	0.50	0.58
coord100-5-3b	0.58	1.49	0.06	0.02	0.18	0.80	0.65
coord100-10-1	9.93	1.36	4.67	1.06	1.09	0.62	3.42
coord100-10-1b	16.61	1.63	16.32	1.06	1.55	2.69	2.11
coord100-10-2	0.63	1.28	0.08	0.91	0.69	0.89	0.79
coord100-10-2b	0.52	0.22	0.00	0.65	0.75	0.13	0.62
coord100-10-3	1.11	3.10	1.05	0.00	1.36	1.53	1.49
coord100-10-3b	0.24	0.77	0.38	0.34	0.23	0.74	0.44
coord200-10-1	1.31	0.93	1.93	0.79	0.33	0.00	0.62
coord200-10-1b	0.45	0.60	1.05	1.38	0.14	0.00	0.01
coord200-10-2	0.50	0.84	0.45	0.28	0.42	0.02	0.23
coord200-10-2b	0.18	0.81	0.22	0.63	0.17	0.17	0.39
coord200-10-3	1.85	1.54	1.91	0.95	1.22	0.00	1.89
coord200-10-3b	0.56	0.67	0.65	0.24	0.71	0.12	0.91
gap prom.(%)	1.38	0.69	1.07	0.41	0.36	0.49	0.56
tiempo prom. (s)	76.69	17.48	258.17	422.36	191.28	176	354.24

Cuadro 2.5: Comparación con metaheurísticas recientes (gaps a BKS)

Resultados para PC-CLRP

Para poder estudiar el problema *PC-CLRP* ha sido necesario generar instancias de prueba. La información adicional requerida para el presente problema son las ganancias o beneficios que los clientes proporcionan cuando son atendidos. Se han generado nuevas instancias basadas en el conjunto de 30 instancias de Prodhon [86]. Los valores de ganancia requeridos fueron generados aleatoriamente utilizando un algoritmo especialmente diseñado. Los detalles de dicho algoritmo pueden encontrarse en el apéndice A.2.

Los cuadros 2.6 y 2.7 muestran los resultados computacionales obtenidos usando el algoritmo propuesto sobre el conjunto de nuevas instancias generadas. Para cada instancia Prodhon fueron generadas dos nuevas instancias: una intentando contener en su solución óptima un número reducido de clientes atendidos (instancias finalizas en '-s') y otra intentando tener un gran número de clientes servidos (instancias finalizadas en '-l').

Como puede observarse en los resultados, el valor de la función objetivo (costo) para el problema *PC-CLRP* es siempre menor o igual a 0. Esto es así porque en la función objetivo se están minimizando costos y maximizando ganancias. Se expresa esta idea minimizando la suma de los costos y la resta de los beneficios. En soluciones óptimas o eficientes, los beneficios deberán siempre superar a los costos, y la peor solución en *PC-CLRP* será no atender a ningún cliente, obteniendo costo 0. En los cuadros 2.6 y 2.7 se informa con la marca (*) los casos en los que el algoritmo *Branch&Cut*, que se presenta en el capítulo siguiente, haya demostrado la optimalidad del resultado obtenido.

Instancia	Costo	Tiempo (s)	Depósitos	Vehículos	Clientes Atendidos
pc20-5-1-s	-6435*	6.16	1	2	8
pc20-5-1-l	-33516*	7.61	2	4	17
pc20-5-1b-s	-768*	6.34	1	1	10
pc20-5-1b-l	-56122*	8.58	1	2	19
pc20-5-2-s	-2971*	5.17	1	3	9
pc20-5-2-l	-44725*	6.55	2	4	16
pc20-5-2b-s	-114*	8.34	1	2	19
pc20-5-2b-l	-59744*	9.69	1	2	19
pc50-5-1-s	-1453	21.10	2	7	32
pc50-5-1-l	-38661	47.50	2	9	40
pc50-5-1b-s	-4639	79.24	2	5	47
pc50-5-1b-l	-62173	95.39	2	5	47
pc50-5-2-s	-4835	32.74	1	5	22
pc50-5-2-l	-43809	48.46	2	10	42
pc50-5-2b-s	-4521	61.75	2	4	39
pc50-5-2b-l	-62867	87.42	2	6	44
pc50-5-2BIS-s	-786	21.74	1	4	19
pc50-5-2BIS-l	-14212	30.99	1	5	24
pc50-5-2bBIS-s	-4383*	43.77	1	2	20
pc50-5-2bBIS-l	-34606*	86.29	2	4	39
pc50-5-3-s	-6050	28.74	1	5	23
pc50-5-3-l	-38504	51.48	2	10	44
pc50-5-3b-s	-21107	63.22	1	3	28
pc50-5-3b-l	-60192	98.39	2	5	47

Cuadro 2.6: Resultados 1 Prize-Collecting

Instancia	Costo	Tiempo (s)	Depósitos	Vehículos	Clientes atend.
pc100-5-1-s	-41917	168.17	1	12	51
pc100-5-1-l	-230396	342.95	2	21	90
pc100-5-1b-s	-28315	250.20	1	6	50
pc100-5-1b-l	-145023	416.91	2	10	91
pc100-5-2-s	-20861	154.92	1	13	55
pc100-5-2-l	-141791	230.33	2	22	93
pc100-5-2b-s	-51662	256.97	1	6	55
pc100-5-2b-l	-177450	323.88	2	11	99
pc100-5-3-s	-11394	148.11	1	13	55
pc100-5-3-l	-119756	316.06	2	21	91
pc100-5-3b-s	-41503	263.46	1	6	54
pc100-5-3b-l	-161475	567.94	2	11	98
pc100-10-1-s	-7804	115.20	1	8	36
pc100-10-1-l	-111218	255.65	2	15	64
pc100-10-1b-s	-26664	213.26	1	4	37
pc100-10-1b-l	-147440	366.16	2	8	70
pc100-10-2-s	-5180	123.04	1	9	37
pc100-10-2-l	-130013	260.58	2	17	70
pc100-10-2b-s	-15662	199.56	1	4	38
pc100-10-2b-l	-157225	405.55	2	8	70
pc100-10-3-s	-2306	100.81	1	7	32
pc100-10-3-l	-122919	261.68	2	17	68
pc100-10-3b-s	-25428	231.40	1	4	36
pc100-10-3b-l	-158249	356.24	2	8	68
pc200-10-1-s	-24852	688.58	1	18	81
pc200-10-1-l	-309573	1486.62	2	36	155
pc200-10-1b-s	-65092	1146.13	1	9	81
pc200-10-1b-l	-391121	2317.02	2	15	138
pc200-10-2-s	-63260	792.43	1	19	84
pc200-10-2-l	-345084	1507.64	2	34	146
pc200-10-2b-s	-108406	1096.32	1	9	84
pc200-10-2b-l	-412262	2405.61	2	16	146
pc200-10-3-s	-37886	715.54	1	17	74
pc200-10-3-l	-300642	1819.34	2	32	138
pc200-10-3b-s	-64310	953.66	1	8	75
pc200-10-3b-l	-384762	2326.08	2	16	147

Cuadro 2.7: Resultados 2 Prize-Collecting

2.11. Conclusiones

En este capítulo se presentó una nueva metaheurística para la versión Prize-Collecting del problema de Localización y Ruteo con capacidades. El método utilizado es un algoritmo de optimización colaborativo de múltiples colonias de hormigas basado en el *ACS* introducido en Dorigo y Gambardella [29]. Este algoritmo utiliza 3 *ACS* especializados para los problemas involucrados: localización, clusterizado y ruteo.

El método fue testeado usando el conjunto de instancias Prodhon que contiene instancias de pequeño, mediano y gran tamaño. Se compararon los resultados obtenidos con otras metaheurísticas para el problema *CLRP*. Las soluciones obtenidas muestran que el algoritmo es capaz de encontrar buenas soluciones y alcanzar los mejores resultados obtenidos en algoritmos previos dedicados al problema *CLRP*.

Para poder testear la versión Prize-Collecting del problema se han generado 60 nuevas instancias basadas en el mismo conjunto de instancias de Prodhon. Si se comparan los resultados obtenidos por la

metaheurística con los que se presentan en el capítulo siguiente puede observarse que se han logrado soluciones de muy buena calidad. Todos los óptimos que han sido probados como tales utilizando el algoritmo *Branch&Cut* fueron encontrados por la metaheurística o el resultado encontrado es muy cercano al óptimo. Si se considera además que los gaps a las cotas inferiores del resto de las instancias es en general bastante reducido, se puede inferir que las soluciones reportadas por la metaheurística se encontrarán cercanas a las soluciones óptimas de cada una de las instancias.

La experimentación efectuada demuestra que las distintas estrategias diseñadas para resolver la variante Prize-Collecting del problema fueron muy efectivas identificando numerosas instancias con clientes desactivados.

Como primera aproximación al problema con métodos heurísticos se priorizó la obtención de soluciones de buena calidad frente a la reducción de los tiempos computaciones requeridos. Como trabajo futuro se podría intentar mejorar las estrategias de búsqueda local, analizando el impacto de cada uno de los operadores o considerando nuevas combinaciones de los mismos. En ambos casos, se tendría como objetivo reducir los tiempos de cómputo sin degradar notablemente la calidad de las soluciones. El software *IRACE*, utilizado como herramienta para seleccionar los mejores valores de los parámetros de la metaheurística presentada, podría ser de gran utilidad para encarar dicha tarea. La metaheurística presentada en este trabajo se utiliza también como heurística inicial del algoritmo *Branch&Cut* que se presentará en el próximo capítulo. La calidad de las soluciones aquí obtenidas son comparadas luego con los resultados generados por medio del algoritmo exacto.

Capítulo 3

ALGORITMO BRANCH&CUT PARA EL PROBLEMA PC-CLRP

Muchos problemas de optimización combinatoria pueden ser modelados mediante formulaciones de Programación Lineal Entera Mixta (*Mixed Integer Linear Programming - MILP*). Existen numerosas maneras de representar un mismo problema de optimización combinatoria mediante modelos *MILP*.

En este capítulo se presenta un nuevo modelo *ILP* (subconjunto de formulaciones *MILP* que utiliza solamente variables enteras) para el problema *PC-CLRP* y un método exacto basado en un algoritmo *Branch&Cut* para intentar resolverlo.

3.1. Programación Lineal

Un *problema de programación lineal* (*Linear Programming - LP*) puede formularse de la siguiente manera:

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

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

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

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

Los problemas de optimización lineal con variables continuas 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 $\mathcal{P}$. 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 [59] 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 y combinan variantes de ambos métodos para resolver los problemas lineales.

3.2. Programación Lineal Entera Mixta

Desafortunadamente, no todos los problemas de optimización pueden ser modelados como problemas de programación lineal con variables continuas. En muchos casos, es necesaria 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, se obtiene lo que se denomina un *problema de optimización lineal entera mixta* (*MILP*):

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

$$c^T x^* = \text{mín}\{c^T x \mid Ax \leq b, x_i \text{ entero para todos los } i \in I \} \tag{3.1}$$

En la definición anterior, si se pide 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 como problemas de programación lineal entera mixta. 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 dos posibilidades, puede modelarse tal dicotomía usando una variable binaria x que tome

$$x = \begin{cases} 1 & \text{si el evento ocurre} \\ 0 & \text{en caso contrario} \end{cases}$$

Numerosos problemas de optimización combinatoria pertenecen a la clase $\mathcal{NP}-Hard$. Gracias al estudio de la *cápsula convexa* de las soluciones 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. Para mayor detalle ver Nemhauser y Wolsey [76].

3.3. Algoritmos de Plano de Corte

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

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

se define $conv(S)$ como la *cápsula convexa* de S, es decir, el conjunto de todos los puntos que son combinación convexa de los puntos de S. Se define a un *poliedro* como el conjunto de puntos definidos mediante un conjunto finito de desigualdades lineales. Por lo tanto, se puede definir $conv(S)$ como el menor poliedro que contiene a 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 problema de programación lineal asociado y se obtendría 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 $\mathcal{NP}-Hard$ tal descripción incluiría un conjunto exponencial de restricciones salvo que $\mathcal{P} = \mathcal{NP}$. 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 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 definen de la siguiente manera:

Dado el problema de programación lineal entera mixta 3.1, 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 3.1 (a) se ve una relajación lineal de un determinado conjunto de puntos S. En la figura 3.1 (b) se puede observar la cápsula convexa asociada a dicho conjunto de puntos S representantes de un determinado problema de programación lineal entera.

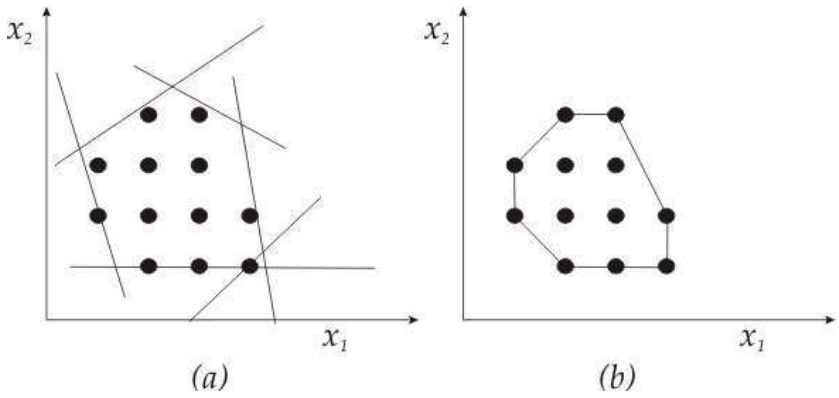

Figura 3.1: Algoritmos de Plano de Corte. (a) Relajación Lineal. (b) Cápsula convexa.

Dada la definición 3.1, la idea básica de los algoritmos de plano de corte puede resumirse en el pseudocódigo 3.7.

Algoritmo 3.7 Algoritmo de Planos de Corte

1: *Inicialización:* 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: *Relajación:* Optimizar el sistema
$c^T x^* = \text{mín}\{c^T x \mid A^{'} x \leq b^{'}, x \in \mathbb{R}^n\}$
Si la solución óptima x^* corresponde a una solución factible del problema entero original, entonces mostrar x^* y parar.
3: *Separación:* Buscar una restricción (f, f_0), $f \in \mathbb{R}^n$, $f_0 \in \mathbb{R}$ con $f^T x^* > f_0$ y
$f^T y \leq f_0$ para todos los $y \in \{x \mid Ax \leq b, x_i \textit{ entero para todos } i \in I\}$.
Agregar la desigualdad $f^T x \leq f_0$ al conjunto de restricciones del sistema $(A^{'}, b^{'})$ anterior.
ir al paso 2.

Es decir, se resuelve mediante programación lineal la relajación del problema original. Si la solución resultante es factible para el problema de programación lineal entera, 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 y se pueda volver a obtener otra solución. De esta manera, si tiene éxito, 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. Para mayor detalle ver Nemhauser y Wolsey [76].

El éxito de este método depende de, principalmente, encontrar familias de desigualdades válidas apropiadas y en lograr *algoritmos de separación* eficientes para identificarlas. Dada una solución obtenida mediante la relajación de un problema de programación lineal entero, los *algoritmos de separación* para una familia de desigualdades válidas intentan identificar desigualdades que se encuentran violadas por dicha solución.

Los cortes o desigualdades válidas pueden ser generadas utilizando 2 enfoques diferentes:

- Utilizando cortes y desigualdades de propósito general que son aplicables a cualquier problema *MILP*. Los conocidos cortes de Gomory, desarrollados por Gomory [46] en 1958, permiten generar desigualdades válidas para un problema utilizando información existente en la solución de una relajación *LP*. Se basa en propiedades de integralidad sobre las variables que no forman parte de la base del *LP*. Se ha demostrado que bajo ciertas condiciones este método de generación de planos de corte converge a la solución óptima del problema. Sin embargo, en la práctica dicha convergencia suele ser extremadamente lenta y es por ello que sólo se utilizan los cortes de Gomory como complemento de otras familias de cortes. Se han propuesto también otras familias de desigualdades válidas generales. Las más conocidas son los cortes cover, cortes disyuntivos, cortes clique, cortes de flujo, etc (Nemhauser y Wolsey [76]). La mayoría de los paquetes de software de programación lineal entera hoy incluyen este tipo de cortes. Si bien las desigualdades válidas y los algoritmos de separación correspondientes tienen en la teoría propiedades interesantes, en la práctica el éxito de uso depende del tipo de problema y de su adecuada utilización como parte de un método de resolución más completo.

- Estudiando y aprovechando la estructura particular del problema.
  Cada problema tiene características propias que pueden ser utilizadas para generar desigualdades especializadas. Dantzig y otros [25] fueron los primeros en aplicar este enfoque para el problema del Viajante de Comercio (*TSP*). Los autores lograron en 1954, con el enfoque propuesto, resolver una instancia de 49 ciudades (gran tamaño para la época) y sentaron las bases del estudio poliedral como una de las técnicas más efectivas para resolver problemas de optimización con métodos exactos. Los primeros estudios poliedrales fueron los desarrollados para el *Set-Packing Problem - SPP* por Padberg [79] y para el problema *TSP* por Gröetschel y Padberg [49, 50].
  La búsqueda efectiva de desigualdades válidas y facetas para diferentes problemas debe ser acompañada del diseño de buenos y eficientes algoritmos de separación. El trabajo de Gröetschel y otros [51] muestra uno de los resultados teóricos más importantes que relacionan la complejidad entre los algoritmos de separación y los problemas de optimización. En dicho trabajo, se prueba que un problema de optimización puede ser resuelto en tiempo polinomial sí y sólo sí el problema de separación

de las desigualdades válidas violadas es también polinomial. Este resultado muestra la complejidad subyacente que existe en la tarea de describir la cápsula convexa de un problema y en el desarrollo de los correspondientes algoritmos de separación.

Dependiendo del problema, puede suceder que se arribe a un estado donde no se encuentren más desigualdades válidas para aplicar. Es decir, que no se hayan podido identificar más desigualdades para agregar en el algoritmo de *planos de corte*. En casos donde no se conoce la caracterización completa de la cápsula convexa asociada, es necesaria la aplicación de otras herramientas que permitan seguir avanzando en la resolución óptima del problema. Es aquí donde toman importancia otras técnicas relacionadas con los algoritmos *Branch&Bound* donde se aplica, en determinadas situaciones, la idea de dividir y conquistar. La combinación de los algoritmos de planos de corte y los algoritmos *Branch&Bound* da lugar a los algoritmos *Branch&Cut*.

3.4. Algoritmos Branch&Bound

Los algoritmos *Branch&Bound* tienen como objetivo dividir el problema original en subproblemas más fáciles de resolver, que son luego evaluados para encontrar la mejor solución del problema original. La manera en que se divide el problema es mediante la utilización de un árbol de enumeración (usualmente binario) que permite comenzar desde el problema original (nodo raíz del árbol) e ir dividiéndolo en problemas más pequeños utilizando desigualdades lineales. Iterativamente se van calculando cotas inferiores (*Lower Bound* - *LB*) y superiores (*Upper Bound* - *UB*) a cada uno de los problemas analizados (Jünger y otros [58], Johnson y otros [56]). Estas cotas son utilizadas para intentar recorrer el árbol de manera eficiente. Los nodos cuyas cotas inferiores superan la cota superior global se sabe que no generaran sub-árboles con soluciones óptimas del problema y por lo tanto pueden ser omitidos (el nodo y sus descendientes). Las cotas inferiores y superiores globales pueden calcularse de varias maneras. Para las cotas inferiores cualquier relajación del problema corriente puede producir una cota válida. En la práctica suele utilizarse la relajación lineal del *LP* o un método denominado Relajación Lagrangiana que permite relajar a su vez ciertas desigualdades del *LP* introduciéndolas en la función objetivo. Las cotas superiores, por su parte, pueden ser calculadas utilizando heurísticas iniciales que generan una solución válida para el problema. Pueden también ser calculadas durante la optimización. Cada vez que una solución del *LP* es también solución del problema principal se obtiene una cota superior. Se puede intentar ir más allá mediante el uso de heurísticas primales que generen soluciones válidas del problema a partir de la solución del *LP* en curso.

La partición del espacio de soluciones, realizado en la etapa de branching, debe dividir el problema en subproblemas con espacios de soluciones factibles. Cada nueva región generada, correspondiente a un subproblema, se genera agregando restricciones al LP en curso. Estos nuevos subproblemas son agregados a la lista de problemas sin explorar. La regla de branching más simple y más utilizada por los paquetes de software generales de programación lineal entera es la regla de branching por variables. Para aplicarla se debe buscar alguna de las variables enteras que tomen un valor fraccional en la solución del LP procesado y generar dos nuevos subproblemas (hijos en el árbol de enumeración) donde se restringe los valores de la variable en un rango que deje fuera al valor fraccional. Llevado a un ejemplo, suponiendo que se encuentra una variable x_i cuyo valor fraccional es d, se genera un nuevo subproblema con la restricción $x_i \leq \lfloor d \rfloor$ y otro con la restricción $x_i \geq \lceil d \rceil$. Este procedimiento se aplica de manera recursiva a cada nodo del árbol de enumeración. También, suele utilizarse como regla de branching la partición en subproblemas utilizando desigualdades lineales. Por ejemplo, para el problema $CVRP$ suele utilizarse como regla de branching desigualdades basadas en conjunto de clientes. Se busca en el LP analizado un conjunto de clientes que tenga un valor fraccional f de rutas que lo atiende y se generan 2 nuevos hijos: uno que requiera como mínimo $\lceil f \rceil$ rutas y otro que requiera como máximo $\lfloor f \rfloor$ rutas.

El pseudocódigo 3.8 muestra los detalles del algoritmo general *Branch&Bound*.

Algoritmo 3.8 Algoritmo Branch&Bound para un problema de minimización

Input: Una formulación $MILP$ definida de la forma $Ax \leq b$

1: *Inicialización:* Crear una lista L con el nodo raíz del árbol de enumeración representando a la relajación del problema ingresado. Calcular la cota superior global $Z^{ub} = \infty$. Inicializar la cota inferior $Z^{lb} = -\infty$.

2: *Parada:* Si las cotas inferior Z^{lb} y superior global Z^{ub} coinciden, parar y reportar x^{ub} como la solución óptima con costo Z^{ub}. Si la lista L está vacía, parar y reportar que el problema no tiene solución.

3: *Selección de nodos:* Elegir un nodo de la lista L sin procesar y borrarlo de la lista.

4: *Relajación:* Resolver la relajación lineal del subproblema que corresponde al nodo elegido. Sea x^* el valor óptimo y Z^* el valor de su función objetivo.

- Actualizar, si corresponde, la cota inferior global Z^{lb}.
- Si el problema no es factible, eliminar el nodo y volver al paso 2.
- Si $Z^* \geq Z^{ub}$, ninguna solución de este nodo o de sus descendientes puede llegar a ser mejor que la solución con función objetivo Z^{ub}. Podar (eliminar el nodo) y volver al paso 2.
- Si la solución de la relajación es solución factible del problema, actualizar $Z^{ub} = \text{mín}\{Z^{ub}, Z^*\}$, actualizar Z^{ub} y volver al paso 2.

5: *Branching:* Generar 2 o más nuevos nodos del árbol (dependiendo de la regla de branching) y agregarlos a L. Volver al paso 2.

El paso *Selección de nodos* (3) determina cuál es el próximo nodo a procesar. La estrategia utilizada en este paso influirá notablemente en el tamaño del árbol de enumeración analizado y por consiguiente en el desempeño final del algoritmo *Branch&Bound*. Entre las estrategias más utilizadas se pueden nombrar la estrategia *Depth First Search* (seleccionar siempre el último nodo de la lista), *Breadth First Search* (seleccionar siempre el primer nodo de la lista), *Best Bound Search* (seleccionar siempre el nodo con menor valor de relajación lineal *LP* de la lista) o combinaciones de las anteriores.

La estrategia de branching utilizada (5) es también determinante del desempeño general del algoritmo. Hay diferentes maneras de seleccionar la variable (o las desigualdades) sobre la que se generarán los nuevos nodos. Entre los métodos más usados pueden nombrarse *mínima infactibilidad*, *máxima infactibilidad*, *pseudocostos*, *pseudocostos reducidos*, *strong branching*, etc. Más información sobre los distintos métodos utilizados puede encontrarse en la revisión de Achterberg y otros [1].

En la práctica, el método *Branch&Bound* por sí solo no proporciona buenos resultados para la mayor parte de los problemas, pero da lugar a una de las ideas centrales de los algoritmos *Branch&Cut*.

3.5. Algoritmos Branch&Cut

Los algoritmos *Branch&Cut* surgen a partir de la combinación de ideas basadas en los algoritmos *Branch&Bound* y los algoritmos de *planos de corte*. Las primeras combinaciones de métodos *Branch&Bound* y de desigualdades válidas específicas para un determinado problema fueron presentadas en el trabajo de Miliotis [72], aplicado al problema *TSP*. A principios de los años 80, el trabajo de Crowder y otros [24] fue muy exitoso aplicando un algoritmo *Branch&Bound* con una etapa de generación de cortes previa al branching (en el nodo raíz del árbol de enumeración). Este trabajo logró muy buenos resultados disminuyendo con su método el árbol de enumeración a explorar. El uso de *facetas* del poliedro asociado mediante algoritmos de *planos de corte* en combinación con métodos *Branch&Bound* fueron inicialmente publicados por Grötschel, Jünger y Reinelt [48] aplicándolo al problema *Lineal Ordering Problem - LOP* . El término *Branch&Cut* fue introducido por Padberd y Rinaldi [77, 78], aplicado a un algoritmo para el problema *TSP*.

Un esquema de un algoritmo *Branch&Cut* para un problema de minimización puede observarse en el pseudocódigo 3.9.

Algoritmo 3.9 Algoritmo Branch&Cut para un problema de minimización

Input: Una formulación *MILP* definida de la forma $Ax \leq b$

Inicialización: Crear una lista L con el nodo raíz del árbol de enumeración representando una relajación del problema. Calcular la cota superior global $Z^{ub} = \infty$ o utilizar un algoritmo heurístico para inicializarla (heurística inicial). Inicializar la cota inferior $Z^{lb} = -\infty$.
Parada: Si las cotas inferior Z^{lb} y superior global Z^{ub} coinciden, entonces devolver la mejor solución factible encontrada x^{ub} como la óptima del problema y parar. Si no existen nodos abiertos del árbol para procesar ($L = \emptyset$), parar e informar que el problema no tiene solución.
Selección de nodos: Elegir un nodo de la lista L sin procesar y borrarlo de la lista.
Relajación: Resolver la relajación lineal del subproblema que corresponde al nodo elegido. Sea x^* el valor óptimo y Z^* el valor de su función objetivo.

- Actualizar, si corresponde, la cota inferior global Z^{lb}.
- Si el problema no es factible, eliminar el nodo y volver al paso 2.
- Si $Z^* \geq Z^{ub}$, ninguna solución de este nodo o de sus descendientes puede llegar a ser mejor que la solución con función objetivo Z^{ub}. Podar (eliminar el nodo) y volver al paso 2.
- Si la solución de la relajación es solución factible del problema, actualizar $Z^{ub} = \text{mín}\{Z^{ub}, Z^*\}$, actualizar Z^{ub}, eliminar de L todos los nodos i con cota inferior $Z^i \geq Z^{ub}$ y volver al paso 2.

Separación: Si conviene, aplicar el algoritmo de separación para encontrar desigualdades válidas violadas. Si se encuentran desigualdades violadas por x^*, agregarlas a la relajación anterior y volver al paso 4.
Branching: Generar 2 o más nuevos nodos del árbol (dependiendo de la regla de branching) y agregarlos a L. Volver al paso 2.

La idea básica de estos algoritmos 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 *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 *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* (Padberd y Rinaldi [77, 78]). 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 presentado algoritmos exitosos *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.

3.6. Modelos de programación lineal entera para PC-CLRP

Se han propuesto 3 modelos de programación lineal entera para el problema *PC-CLRP*. En la introducción se ha presentado el primer modelo (PC2) inspirado en un modelo de programación lineal entera de 2 índices. En esta sección se presenta un modelo alternativo de 2 índices (PC2E) que sirve como base del algoritmo *Branch&Cut* detallado en 3.9. El el apéndice A.3 se propone un tercer modelo (PC3) basado en un modelo de programación lineal entera de 3 índices.

3.6.1. Modelo 2 índices inicial

El modelo presentado en la sección 1.1 representa el primer abordaje al problema *PC-CLRP* desde un enfoque exacto. Para verificar la corrección de dicha formulación se implementó un algoritmo *Branch&Cut* simplificado, utilizando como relajación inicial a las desigualdades (1.2), (1.3) y (1.5). Se implementaron procedimientos de separación adecuados para identificar las desigualdades (1.4), (1.6) y (1.7) violadas en cada iteración. Las mismas fueron incorporadas a medida que se las necesitaba debido a que el número total de restricciones de estas familias crece de manera exponencial en función de la cardinalidad del conjunto de clientes J. Los algoritmos se implementaron en C++ sobre el optimizador comercial de programación lineal entera CPLEX 12.6.1 [54]. Se aplicó el algoritmo sobre el conjunto de instancias más pequeñas (instancias de 20 clientes y 5 localizaciones) tomadas de las nuevas instancias generadas para *PC-CLRP* basadas en las instancias de Prodhon. En el apéndice A.2 se detallan las instancias y el método utilizado para generarlas. Se consiguió alcanzar el resultado final óptimo del algoritmo *Branch&Cut* en solo 3 instancias en más de 10 horas de ejecución.

Se pudo verificar que el árbol de posibilidades generado por el *Branch&Cut* tenía un tamaño considerable, por lo que se estudió su perfeccionamiento. Se obtuvo como mejora sustancial la incorporación de una nueva familia de desigualdades tendientes a eliminar la simetría de soluciones. Al ser un modelo basado en un grafo dirigido, cada ruta puede ser representada de manera equivalente invirtiendo la dirección de los ejes involucrados. Esto genera la poco deseable situación de tener 2^l soluciones equivalentes por cada solución válida, siendo l la cantidad de rutas utilizadas en una solución. Para eliminar esta solución se utilizó la familia de desigualdades de eliminación de simetría:

$$x(E(S)) + x(l:S) + x(S:k) + x_{il} + x_{ki} \leq \sum_{j \in S} x(\delta^+(j)) + 2 \qquad \begin{matrix} \forall_{S \subseteq J} \\ l,k \in J \end{matrix} \quad \begin{matrix} i \in I \\ l > k \end{matrix} \tag{3.2}$$

Al igual que con las familias de desigualdades (1.4), (1.6) y (1.7), esta familia tiene una cantidad de restricciones que crece exponencialmente en función del tamaño de J. Por lo tanto, se implemento una rutina de separación específica para detectar las desigualdades violadas que deben incorporarse en cada iteración del algoritmo *Branch&Cut*.

Sin embargo, estas modificaciones permitieron mejorar sólo levemente el desempeño del algoritmo *Branch&Cut* implementado. Las instancias, aún las más pequeñas (20 clientes y 5 posibles localizaciones de depósitos), son muy difíciles de resolver con esta formulación. Por lo tanto, se analizó como alternativa convertir la formulación anterior en una formulación basada en un grafo no dirigido para reducir el tamaño de la formulación. La primera ventaja de este enfoque es el de utilizar un número menor de variables (se reduce a la mitad la cantidad de variables utilizadas para representar ejes). Sin embargo, la principal ventaja es la de eliminar los problemas de simetría mostrados anteriormente. Esto elimina la posibilidad de considerar repetidas veces soluciones equivalentes.

En la sección siguiente se presenta la nueva formulación.

3.6.2. Modelo 2 índices extendido

La nueva formulación está basada en un grafo no dirigido $G = (V, E)$ con un conjunto de nodos V, donde $V = I \cup J$ representa, al igual que antes, la unión de las posibles localizaciones de depósitos y el conjunto de clientes, y E representa el conjunto de ejes no dirigidos que conecta a los clientes entre sí y a los clientes con las posibles localizaciones.

El nuevo modelo utiliza un eje para representar la conexión entre dos nodos. Este eje será representado en la formulación de programación lineal entera por una variable binaria. Sin embargo, surge un problema cuando se intenta representar rutas que contienen un único cliente. Es decir, rutas que comienzan en un depósito, visitan un cliente y retornan al depósito. Si se utiliza la representación explicitada anteriormente

con variables binarias se deja de poder representar rutas de un solo cliente. Por lo tanto, se analizan 2 alternativas de representación.

La primera es utilizar variables enteras para representar los ejes, en lugar de utilizar variables binarias, permitiendo que las mismas tomen los valores 0, 1 y 2. Las variables valdrán 2 sólo cuando representen rutas de un solo cliente.

La segunda alternativa, utilizada en Belenguer y otros [16], es utilizar un nuevo conjunto de variables binarias especializadas para representar rutas de un solo cliente. Debido al análisis efectuado en [16] y en Contardo y otros [21] y a ciertas ventajas relacionadas con la utilización de un modelo de programación lineal entera binario se ha decidido utilizar el segundo enfoque. Las nuevas variables utilizadas serán notadas con la letra y para mantener la notación de [21]. Las variables binarias que determinan qué localizaciones son utilizadas se representan en este nuevo modelo utilizando la letra z, respetando nuevamente la notación del trabajo referido. Se cumple que $x_{ij} = 1$ si el eje $(i, j) \in E$ se usa en alguna ruta y 0 en caso contrario; $y_{ij} = 1$ si el cliente j es usado en una ruta de un solo cliente desde la localización i o 0 en caso contrario; $z_i = 1$ si la localización i es utilizada y 0 en caso contrario.

Una modificación adicional fue introducida en este nuevo modelo para indicar si los clientes se encuentran visitados o no visitados. En el modelo anterior, se consideraba a un cliente como visitado si el flujo saliente del nodo que lo representaba valía 1. En este nuevo modelo se utiliza una variable adicional. Se define $w_{ij} = 1$ si el cliente j es atendido a través del depósito i o 0 en caso contrario. Un cliente j no atendido tendrá $\sum_{i \in I} w_{ij} = 0$ y 1 en caso contrario.

Se mantiene la notación utilizada en (PC2) y se redefine la notación requerida para expresar el modelo sobre el conjunto de ejes no dirigidos E. Para cada subconjunto $U \subseteq V$ se define $E(U) = \{(u, w) \in E : u, w \in U\}$ y $\delta(U) = \{(u, w) \in E : u \in U, w \notin U\}$. Para cada par de subconjuntos disjuntos U y W se define $(U : W) = \{(u, w) \in E : u \in U, w \in W\}$. Para un conjunto dado $M \subseteq E$ se define $x(M) = \sum_{e \in M} x_e$ y para un conjunto $M' \subseteq \delta(I)$ se define $y(M') = \sum_{e \in M'} y_e$. Para cada cliente j se establece que $w(j) = \sum_{i \in I} w_{ij}$ y para un conjunto de clientes S se define $w(S) = \sum_{j \in S} w(j)$. Por último, para un conjunto de clientes $S \subseteq J$, se define $d(S) = \sum_{j \in S} d_j$ y $b(S) = \sum_{j \in S} b_j$.

La formulación (PC2E) para el modelo de 2 índices extendido es la siguiente:

$$min \sum_{e \in E} c_e x_e + 2 \sum_{e \in \delta(I)} c_e y_e + \sum_{i \in I} O_i z_i + \sum_{e \in \delta(I)} \frac{F}{2} x_e + \sum_{e \in \delta(I)} F y_e - \sum_{j \in J} b_j w(j) \tag{3.3}$$

sujeto a

$$x(\delta(j)) + 2y(I:j) \leq 2 \qquad j \in J \tag{3.4}$$

$$x(E(S)) \leq w(S) - \sum_{j \in S} \frac{d_j w(j)}{Q} \qquad S \subseteq J \tag{3.5}$$

$$x_{ij} + y_{ij} \leq z_i \qquad i \in I \quad j \in J \tag{3.6}$$

$$x_{ij} + y_{ij} \leq w_{ij} \qquad i \in I \quad j \in J \tag{3.7}$$

$$x(I:j) + y(I:j) \leq 1 \qquad j \in J \tag{3.8}$$

$$\sum_{j \in J} w_{ij} \leq |J| z_i \qquad i \in I \tag{3.9}$$

$$\sum_{j \in J} d_j w_{ij} \leq W_i z_i \qquad i \in I \tag{3.10}$$

$$x(\delta(j)) + 2y(I:j) = 2w(j) \qquad j \in J \tag{3.11}$$

$$x_{jk} \leq 1 - (w_{ij} - w_{ik}) \qquad k, j \in J$$

$$i \in I \tag{3.12}$$

$$x_e \in \{0,1\} \qquad e \in E \tag{3.13}$$

$$y_e \in \{0,1\} \qquad e \in \delta(I) \tag{3.14}$$

$$z_i \in \{0,1\} \qquad i \in I \tag{3.15}$$

$$w_{ij} \in \{0,1\} \qquad i \in I \quad j \in J \tag{3.16}$$

La función objetivo (3.3) representa la minimización de los costos y la maximización simultánea de los beneficios. El costo se define como la suma de los costos de ruta, los costos fijos de apertura de los depósitos utilizados y el costo de los vehículos usados, contabilizados utilizando el flujo saliente del conjunto de depósitos. Los beneficios se calculan sumando los premios de cada cliente activo (clientes con valor 1 en las variables w).

Las desigualdades (3.4) son las restricciones de grado para los nodos clientes. Junto a las restricciones (3.11), determinan el grado de cada cliente y al mismo tiempo sincronizan el valor de las variables w con el flujo activo en cada cliente. Las restricciones (3.5) son una nueva versión de las desigualdades de capacidad para los vehículos, especializada en el nuevo problema *PC-CLRP*. Se ha llamado a esta nueva versión restricciones de PC-Capacidad.

Las desigualdades (3.6) establecen que los ejes incidentes a un depósito sólo deben ser usados si el depósito está activo. Se puede ver que $x_{ij} + y_{ij} \leq 1$ siempre se cumple debido a que las variables x_{ij} y y_{ij} no pueden valer simultáneamente 1. Las desigualdades (3.7), por su parte, establecen que las variables w_{ij} deben activarse para los clientes que inician y terminan sus rutas en el depósito i. La familia de restricciones (3.8) elimina soluciones ilegales donde una ruta comienza en un depósito, atiende un cliente y termina luego en otro depósito.

Por su parte, las desigualdades (3.9) establecen que si hay variables w activadas para un depósito, entonces el mismo debe encontrarse activado. Las restricciones (3.10) imponen las restricciones de capacidad para los depósitos. En este modelo, la utilización de las variables w habilita la posibilidad de utilizar estas restricciones en lugar de las restricciones utilizadas en Belenguer y otros [16] y Contardo y otros [21] para el problema de Localización y Ruteo con Capacidades general. En dichos trabajos, las familias de restricciones utilizadas para asegurar el cumplimiento de la capacidad de los depósitos tienen un número de elementos que crece exponencialmente con el tamaño de $|J|$ (depot capacity constraints o facility capacity inequalities).

Las restricciones (3.12) sincronizan junto a las restricciones (3.7) la lógica de las variables w con los ejes de rutas. Han sido bautizadas como restricciones de lógica de ruta. Establecen que si un eje se encuentra activo, entonces los valores de las variables w asociadas a los clientes del eje deben coincidir en cada uno de los depósitos. Se utilizan 2 restricciones por eje y por depósito. El uso de estas restricciones garantiza la eliminación de soluciones donde una ruta comienza en un depósito y termina en otro. Nuevamente, [16] y [21] utilizan para ello restricciones que tienen un número de elementos que crece exponencialmente con el tamaño de $|J|$ (path-elimination constraints). El uso de las variables w elimina tal necesidad.

Por último, las desigualdades (3.13)-(3.16) son las restricciones de integralidad.

Es importante señalar que las familias de restricciones (3.5) contienen un número de restricciones que crece exponencialmente de acuerdo al tamaño de J. Para poder utilizar esta formulación de manera eficiente se usan estas restricciones dentro de un algoritmo de planos de corte. Se identifican las restricciones de capacidad violadas y se las agrega cuando las mismas son necesitadas. Además de relajar las restricciones de integralidad (3.13)-(3.16), se ha decidido relajar inicialmente las restricciones (3.12) ya que su número crece en orden cuadrático en función del tamaño de J y después de cierta experimentación se ha verificado que es más eficiente agregarlas a medida que las mismas se necesitan.

3.7. Desigualdades válidas

Además de las desigualdades (3.5) y (3.12), hay otras familias de desigualdades válidas que también son agregadas como cortes en el algoritmo de planos de corte. Las mismas serán presentadas en esta sección.

3.7.1. Desigualdades válidas w-z

Los cortes w-z para la formulación (PC2E) refuerzan la vinculación entre las variables w y z.

$$w_{ij} \leq z_i \qquad i \in I \quad j \in J \tag{3.17}$$

Si bien las desigualdades (3.9) se encuentran dominadas por (3.17), estas últimas son agregadas como cortes a medida que se las necesita en el algoritmo de planos de corte. Sólo las desigualdades (3.9) son incorporadas a la formulación inicial para no recargar la formulación y tornarla más difícil de optimizar.

3.7.2. Desigualdades válidas PC-Capacidad y PC-Subtour

Al igual que lo efectuado en Belenguer y otros [16] para los cortes y-Capacidad, se puede mejorar o fortalecer las desigualdades PC-Capacidad (3.5) utilizando las variables y. De esta forma, las desigualdades PC-Capacidad aquí presentadas tendrán la forma

$$y(S' : I) + x(E(S)) \leq w(S) - \sum_{j \in S \setminus S'} \frac{d_j w(j)}{Q} \qquad S' \subset S \quad S \subseteq J \tag{3.18}$$

Se han utilizado 2 versiones de las desigualdades PC-Capacidad. La recién enunciada y la siguiente versión (PC-Subtour) que utiliza como parte derecha de la desigualdad a la suma de las variables w de los clientes en S menos uno. Esta versión es un caso particular de las desigualdades PC-Capacidad que se aplica a los conjuntos S que contengan una demanda total inferior a la capacidad de un vehículo. Este caso es una nueva versión de las estudiadas restricciones de Subtour. En [16] se utilizan restricciones similares (y-Subtour) para el problema *CLRP*. En el presente trabajo se presenta una versión adecuada para el problema *PC-CLRP*. La elección de qué variable w se deja afuera dependerá de los valores encontrados en dichas variables y se tratará de utilizar un cliente j cuyo w_j sea lo más grande posible.

$$y(S' : I) + x(E(S)) \leq w(S'') \qquad S' \subseteq S'' \quad S'' \subset S \quad S \subseteq J \quad |S| = |S''| + 1 \quad d(S) \leq Q \tag{3.19}$$

Es importante destacar que las restricciones PC-Subtour tienen especial utilidad porque no son un caso particular de las desigualdades PC-Capacidad. Las mismas no están dominadas en los casos en que $\sum_{j \in S \setminus S'} d_j/Q < 1$.

Se puede mostrar lo anterior con el ejemplo graficado en la figura 3.2. Suponiendo clientes con demanda unitaria, vehículos con capacidad máxima 6 y los valores de los ejes tal como se muestran en la figura. Las desigualdades de PC-Capacidad no están violadas porque

$$x(E(\{1,2,3\})) = 1/2 + 1/2 + 1 = 2 \leq w(\{1,2,3\}) - \frac{3}{4}\frac{1}{6} - \frac{1}{6} - \frac{3}{4}\frac{1}{6} = \frac{5}{2} - \frac{10}{24} = 2{,}0833 \quad \text{tomando } S' = \emptyset$$

Sin embargo, tomando $S'' = \{1,3\}$ la desigualdad de PC-Subtour sí se encuentra violada

$$2 = x(E(\{1,2,3\})) < w(\{1,3\}) = \frac{3}{4} + \frac{3}{4} = 1{,}5 \quad \text{tomando } S' = \emptyset \text{ y } S'' = \{1,3\}$$

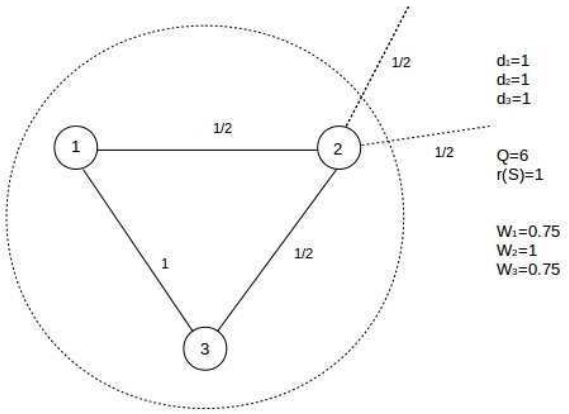

Figura 3.2: Restricción PC-Subtour violada

Es necesario ahora demostrar la validez de estas 2 versiones de desigualdades.

Proposición 3.1. *Las desigualdades PC-Capacidad* (3.18) *son válidas para el problema* PC-CLRP.

Demostración. Se utilizarán argumentos similares a los presentados en Belenguer y otros [16] para demostrar la validez de las desigualdades y-Capacidad en el problema *CLRP*. Puede verse que si $S' = \emptyset$, se está en presencia de las desigualdades (3.5). Por lo tanto, (3.18) incluye a las desigualdades (3.5). Por otra parte, si $S' \neq \emptyset$ las desigualdades (3.18) dominan a (3.5) debido al término $y(S' : I)$. Si se considera una solución factible para *PC-CLRP* y se toma $S'' \subseteq S'$ como el conjunto de clientes activos de S' que forman parte de rutas unitarias (rutas donde se atiende sólo a un cliente y se vuelve al depósito), entonces $|S''| = w(S'') = y(S' : I)$. Se sabe que al menos se necesitaran $k(S \setminus S'')$ vehículos para atender los clientes en $S \setminus S''$ (tomando $k(S) = \lceil d(S)/Q \rceil$) y, por consiguiente, $\lceil \sum_{j \in S \setminus S''} d_j w(j)/Q \rceil$ vehículos para atender los

clientes activos en $S \setminus S''$. Además, se sabe también que estos viajes no están conectados con los viajes de los clientes en S'' porque los mismos ya están atendidos como viajes unitarios. Entonces, utilizando (3.5) se tiene que

$x(E(S)) = x(E(S \setminus S'')) \leq w(S) - w(S'') - \sum_{j \in S \setminus S''} d_j w(j)/Q \leq w(S) - w(S'') - \sum_{j \in S \setminus S'} d_j w(j)/Q$

y se obtiene lo buscado. □

Proposición 3.2. *Las desigualdades PC-Subtour* (3.19) *son válidas para el problema* PC-CLRP.

Demostración. Por definición $d(S) \leq Q$, por lo tanto se sabe que un solo vehículo es suficiente para satisfacer la demanda de los clientes de S. Como se está considerando el problema *PC-CLRP*, se sabe que $x(E(S)) \leq |S| - 1$, es decir que no puede haber subtours y a lo sumo un vehículo puede recorrer los clientes activos de S. Si $S' = \emptyset$, la desigualdad (3.19) se cumple trivialmente ya que $w(S'') \leq |S| - 1$. Suponiendo que $S' \neq \emptyset$, en una solución factible para *PC-CLRP* se puede tomar $S^* \subseteq S'$ como el conjunto de clientes de S' que forma parte de rutas unitarias, se tiene que $|S^*| = w(S^*) = y(S' : I)$. El resto de los clientes de $S \setminus S^*$ será atendido por un solo vehículo. Si se supone que hay al menos un cliente activo en $S \setminus S^*$, entonces

$x(E(S)) = x(E(S \setminus S^*)) \leq w(S) - w(S^*) - 1$

Pero como (3.19) se debe cumplir para cualquier S y S', entonces

$x(E(S)) = x(E(S \setminus S^*)) \leq w(S'') - w(S^*)$

Esta última reducción puede hacerse porque se sabe por definición que $S' \subseteq S''$, y por ende $S^* \subseteq S''$. Sumando $w(S^*) = y(S' : I)$ a ambos lados de la desigualdad se obtiene lo buscado.

□

3.7.3. Desigualdades y-Capacidad

Las desigualdades y-Capacidad tal como fueron definidas en [16] han sido de utilidad en el algoritmo *Branch&Cut* presentado. Si bien estas desigualdades parecerían a simple vista estar dominadas por las desigualdades PC-Capacidad, las mismas son válidas para el problema presentado y sirven para reforzar y mejorar la relajación del programa lineal en curso. Aunque no tienen en cuenta qué clientes se encuentran activos, las mismas son pedidas para todos los conjuntos $S \subseteq J$. Por lo tanto, se pedirán también para el conjunto de clientes activos en una determinada solución.

$$y(S' : I) + x(E(S)) \leq |S| - k(S \setminus S') \qquad S' \subset S \quad S \subseteq J \tag{3.20}$$

utilizando $k(S) = \lceil d(S)/Q \rceil$

Estas desigualdades no están dominadas por las desigualdades PC-Capacidad porque la parte derecha de (3.20) puede ser más restrictiva. Esto es así debido a que $k(S)$ se obtiene redondeando el valor de la cantidad mínima de vehículos al valor entero inmediato superior. Las desigualdades PC-Capacidad, por su parte, obtienen sólo una aproximación fraccional a dicho valor porque incluyen a las variables w para considerar los clientes activos.

Proposición 3.3. *Las desigualdades y-Capacidad* (3.20) *son válidas para el problema* PC-CLRP.

Demostración. La demostración es la misma que la efectuada en Belenguer y otros [16] para el problema *CLRP*. La aplicación a *PC-CLRP* es trivial si se observa que $w(S) \leq |S| \quad \forall S \subseteq J$ y por consiguiente, la parte derecha de (3.20) es totalmente válida para *PC-CLRP*.

□

3.7.4. Desigualdades válidas 3-Path

Las desigualdades 3-Path refuerzan la interrelación entre las variables x y las variables w. Son un caso particular de las desigualdades de PC-Subtour (3.19) tomando $S = \{j, k, l\} \quad S' = \emptyset \quad S'' = \{j, k\}$.

$$x_{jk} + x_{kl} + x_{jl} + \leq \sum_{i \in I} (w_{ij} + w_{ik}) \qquad j, k, l \in J \tag{3.21}$$

Debido a las desigualdades y-Capacidad se sabe que $x_{jk} + x_{kl} + x_{jl} \leq 2$. Por lo tanto, con estas desigualdades se exige que para cada terna de clientes activos conectados, 2 de sus variables w estén activas con valor 1. La elección del par de variables w es indiferente, por lo tanto, en el algoritmo de separación se intentará seleccionar las variables w con menor valor.

3.7.5. Desigualdades válidas Co-Circuit y 1 Co-Circuit

Los cortes 1 Co-Circuit son un caso particular de los llamados cortes Co-Circuit introducidos por Barahona y Grötschel en [14] y luego usados en Belenguer y otros [16] para el problema *CLRP*. Si no se consideran las variables y que corresponden a viajes de un solo cliente, se sabe que en cualquier solución válida, los nodos (tanto clientes como localizaciones) deberán tener 0 o una cantidad par de ejes adyacentes activos. Por lo tanto, se pueden utilizar las siguientes restricciones Co-Circuit.

$$x(\delta(S) \setminus F) \geq x(F) - |F| + 1 \qquad \forall S \subseteq J \cup I, \quad \forall F \subseteq \delta(S), \quad |F| \text{ impar} \tag{3.22}$$

Proposición 3.4. *Las desigualdades Co-Circuit* (3.22) *son válidas para el problema* PC-CLRP.

Demostración. Siguiendo el razonamiento presentado en Belenguer y otros [16] para este tipo de desigualdades en el problema *CLRP*, si $x(F) < |F|$ (3.22) se satisface de manera trivial porque el lado derecho de la desigualdad se vuelve menor o igual a 0. Para el caso en que $x(F) = |F|$, como $|F|$ es impar por definición, al menos un eje de $\delta(S) \setminus F$ se debe usar y de esta manera se cumple la restricción. Es importante destacar que esta familia de desigualdades es siempre válida para el problema *PC-CLRP* porque la posibilidad de que algunos clientes estén desactivados no influye en la validez de la misma, que utiliza solo la información del flujo saliente del conjunto S.

□

En el caso de las restricciones 1 Co-Circuit, se utiliza $S = \{v\}$ para cada $v \in J \cup I$ y se busca conjuntos F de cardinalidad $|F| = 1$.

3.7.6. Desigualdades válidas PC-path-elimination

Las desigualdades path-elimination previenen soluciones con rutas que comienzan en un depósito y finalizan en otro. En la formulación presentada dichas soluciones no son factibles debido a la utilización de las variables w y las desigualdades de lógica de rutas. Sin embargo, se analizó si era posible que en soluciones del *LP* no factibles del problema, el uso de las desigualdades path-elimination reforzaba la formulación mejorando las soluciones de la relajación lineal.

Se adaptó las desigualdades path-elimination mejoradas definidas en Belenguer y otros [16] y se obtuvo las PC-path-elimination:

$$y(S \cup \{l, j\} : I) + x(E(S \cup \{l, j\})) + x(\{j\} : I') + x(\{l\} : I \setminus I') \leq w(S) + w(\{j\}) + w(\{l\})$$
$$\forall j, l \in J, \forall S \subseteq J \setminus \{l, j\}, \forall I' \subset I \qquad (3.23)$$

Sin embargo, los resultados computacionales mostraron que el algoritmo de separación no encontraba desigualdades de este tipo, mostrando que la formulación propuesta eliminaba completamente las rutas entre depósitos.

Se ha decidido no separar estas desigualdades en el algoritmo *Branch&Cut* debido a que los pocos casos donde se identificaba alguna desigualdad correspondía más a cuestiones de redondeo e inestabilidad numérica que a desigualdades genuinamente violadas.

3.7.7. Desigualdades válidas derivadas de CVRP

El problema de ruteo de vehículos con capacidades (*CVRP*) puede considerarse como un caso particular del problema de localización y ruteo con capacidades general (*CLRP*). Esto es posible si se considera un solo depósito y se eliminan los costos de apertura de depósito O y los costos de utilización de vehículos F.

El problema *CVRP* ha sido muy estudiado y existen muchos trabajos al respecto tanto sobre métodos exactos como heurísticos. Pueden citarse algunas revisiones que muestran los avances recientes del problema [45], [93] y [95].

Con respecto a los métodos exactos basados en programación lineal entera se puede citar el algoritmo *Branch&Cut* para *CVRP* presentado en Lysgaard y otros [67]. En dicho trabajo se utilizan numerosas familias de desigualdades válidas con el que se logran muy buenos resultados. Se ha estudiado la adaptación de las mismas al problema *PC-CLRP* y se ha logrado obtener nuevas versiones basadas en las familias de desigualdades de Capacidad (presentadas anteriormente como desigualdades PC-Capacidad, PC-Subtour y y-Capacidad), de desigualdades multistar generalizadas grandes (GLM) y de desigualdades Hypotour.

Para hacerlo se han tenido que mapear las soluciones del problema *PC-CLRP* a soluciones del problema *CVRP*. Dada una solución *PC-CLRP*, si todos los depósitos son contraídos en un único depósito, acumulando los ejes adyacentes de manera adecuada, se obtiene una solución factible para *Prize-Collecting Capacitated Vehicle Routing Problem - PC-CVRP*. Para mapear este último al problema *CVRP* se debería, además, considerar la posibilidad de que un cliente sea no atendido y mapear las variables considerando las x e y aquí utilizadas. Suponiendo que la formulación de 2 índices de [67] utiliza variables u_{ij} para representar a los ejes (i, j), se debe entonces considerar el mapeo $u_{ij} = x_{ij}$ para los ejes $(i, j), i, j \in J$ y el mapeo $u_{dj} = \sum_{i \in I} (x_{ij} + 2y_{ij})$ para los ejes $(d, j), j \in J$. Como puede observarse en este mapeo, la formulación de 2 índices de [67] utiliza variables enteras $[0, 1, 2]$ que, como se menocionó al presentar la formulación (PC2E) para *PC-CLRP*, es la otra alternativa de representación para poder expresar rutas de un solo cliente.

A continuación se describirán las nuevas versiones de desigualdades GLM y Hypotour.

Desigualdades válidas PC-GLM

Las desigualdades Multistar fueron inicialmente definidas por Araque y otros [6] para *VRP* con demandas unitarias. En el presente trabajo se ha utilizado la familia de desigualdades *generalized large multistar* (GLM) introducidas en Gavish [41] para el Capacitated Minimal Spanning Tree Problem y luego utilizadas en Gouveia [47] y Lysgaard y otros [67] para *CVRP*. Se presenta la definición utilizada en [67]

$$x(\delta(N)) \geq 2\sum_{i\in N}\frac{d_i}{Q} + 2\sum_{j\in\bar{N}}\frac{d_j}{Q}x(N:\{j\}) \qquad N\subseteq J \quad \bar{N}\cup N = J$$

donde $N \subset J$ es un conjunto de clientes (núcleo). Los ejes del corte $x(\delta(N))$ se encuentran representados utilizando la notación de [67], donde solamente se utilizan variables enteras x con valores $[0, 1, 2]$. Estas desigualdades pueden utilizarse en el problema *PC-CLRP* debido a que las mismas pueden reescribirse permitiendo la posibilidad de que algunos o todos los nodos de N no estén activos. Para que esto se cumpla se las ha reescrito de la siguiente manera

$$\sum_{j\in\bar{N}}\frac{d_j}{Q}x(N:\{j\}) + y(N:I) + x(E(N)) \leq |N| - \frac{d(N)}{Q} \qquad N\subseteq J \quad \bar{N}\cup N = J$$

Se ha utilizado que para $S \subseteq J$, $x(\delta(S)) = 2|S| - 2y(S : I) - 2x(E(S))$. Esto es válido para el problema *CLRP*. Sin embargo, para que las desigualdades sean válidas en *PC-CLRP* es necesario utilizar las igualdades (3.11) del modelo (PC2E). Con ellas se puede utilizar que $x(\delta(S)) = 2w(S)-2y(S:I)-2x(E(S))$ y obtener las nuevas desigualdades PC-GLM

$$\sum_{j\in\bar{N}}\frac{d_j}{Q}x(N:\{j\}) + y(N':I) + x(E(N)) \leq w(N) - \sum_{j\in N\setminus N'}\frac{d_j}{Q}w(\{j\}) \qquad N'\subset N \quad N\subseteq J \quad \bar{N}\cup N = J \tag{3.24}$$

Para que la restricción anterior sea válida se ha tenido que modificar el último término sumando sólo las demandas de los clientes activos en N y por ende se ha modificado también el término $y(N' : I)$. De lo contrario, con el lado derecho propuesto previamente, la desigualdad deja de ser válida en determinados casos.

Proposición 3.5. *Las desigualdades PC-GLM* (3.24) *son válidas para el problema* PC-CLRP.

Demostración. Puede pasarse el primer término de (3.24) al lado derecho de la desigualdad obteniendo

$$y(N':I) + x(E(N)) \leq w(N) - \sum_{j\in N\setminus N'}\frac{d_j}{Q}w(\{j\}) - \sum_{j\in\bar{N}}\frac{d_j}{Q}x(N:\{j\}) \qquad N'\subset N \quad N\subseteq J \quad \bar{N}\cup N = J \tag{3.25}$$

Esta desigualdad tiene una estructura similar a las desigualdades PC-Capacidad (3.18) con el lado derecho más restringido.

Sin pérdida de generalidad se va a suponer que $N' = \emptyset$ ya que la demostración de la participación de $y(N' : I)$ es equivalente a la mostrada para PC-Capacidad.

Si se tiene una solución factible para el problema *PC-CLRP*, y sea N^* el conjunto de clientes activos en la misma, se cumple que $w(N) = |N^*|$ y por las desigualdades y-Capacidad se sabe que

$$x(E(N^*)) \leq |N^*| - k(N^*)$$

Si se logra determinar que los últimos 2 términos de (3.25) cumplen que

$$\sum_{j \in N^*} \frac{d_j}{Q} w(\{j\}) + \sum_{j \in \bar{N}} \frac{d_j}{Q} x(N : \{j\}) \leq k(N^*)$$

se sabría que (3.25) es válida.

En el caso que los $j \in \bar{N}$ cumplan $x(N : \{j\}) = 1$ lo anterior se cumple debido a las desigualdades y-Capacidad (3.20), ya que los caminos de N^* se extienden con nodos de $\bar{N}$ y las capacidades de las rutas no pueden estar excedidas.

En caso que algún $j \in \bar{N}$ cumpla que $x(N : \{j\}) = 2$, lo anterior no se cumple. Sin embargo, en dichos casos la cantidad de ejes activos en el lado izquierdo $E(N^*)$ disminuye en 1 por cada j que cumpla lo anterior debido a que el algún camino de N^* se interrumpe saliendo de N^*, pasando por j y luego volviendo a ingresar a N^*. En dichas situaciones, la desigualdad anterior se cumple trivialmente lográndose lo buscado para (3.25). □

Desigualdades válidas PC-2E-Hypotour

Se ha utilizado la familia de desigualdades *2-edges extended hypotour* (2EH) introducidas por Augerat [9] y luego utilizadas en Lysgaard y otros [67]. Dichas desigualdades son de la forma

$$x(\delta(W)) + 2x(F) \geq 2x_{e1} + 2x_{e2} \qquad F \subset E \quad e_i = \{u_i, v_i\} \quad \text{para } i = 1, 2 \quad v_1, v_2 \notin W$$

donde $W \subset J$ es un conjunto de clientes y $e_1, e_2 \in \delta(W)$ son ejes. $x(\delta(W))$ representa a los ejes del corte de W utilizando la notación de [67], donde solamente se utilizan variables enteras x con valores $[0, 1, 2]$. Las desigualdades 2EH expresan, en su formulación original, que una solución factible para el problema *CVRP* con $x(\delta(W)) = 2$ y $x_{e1} = x_{e2} = 1$ debe contener activo al menos un eje de F. Esto está basado en el hecho de que cualquier ruta que visite los clientes de $W \cup \{v_1, v_2\}$ de manera consecutiva, comenzando y terminando en v_1, v_2 debe contener algún eje de F. La identificación de dicho F estará basada en el hecho

de que cualquier camino hamiltoniano que pase a través de $W \cup \{v_1, v_2\}$ se debe extender por 2 caminos hasta el depósito, con conjuntos de nodos disjuntos salvo la intersección final en el depósito. La demanda total de dicho ciclo no debe exceder a la capacidad del vehículo. Estas desigualdades pueden utilizarse en el problema *PC-CLRP* debido a que, como se verá en breve, las mismas pueden reescribirse permitiendo la posibilidad de que algunos nodos de W no estén activos. Para ello se las ha tenido que reescribir de la siguiente manera

$$x(E(W)) + y(W : I) - x(F) + x_{e1} + x_{e2} \leq |W| \qquad F \subset E \quad e_i = \{u_i, v_i\} \quad \text{para } i = 1, 2 \quad v_1, v_2 \notin W$$

Se ha utilizado que $x(\delta(S)) = 2|S| - 2y(S : I) - 2x(E(S))$. Esto es válido para el problema *CLRP*. Para hacerlo válido para *PC-CLRP* y ajustar aún más la restricción hay que tomar en cuenta las igualdades (3.11) del modelo (PC2E). A partir de ellas se puede utilizar que $x(\delta(S)) = 2w(S) - 2y(S : I) - 2x(E(S))$ y obtener las desigualdades PC-2E-Hypotour

$$\begin{aligned} x(E(W)) + y(W : I) - x(F) + x_{e1} + x_{e2} \leq w(W) \qquad & F \subset E \quad e_i = \{u_i, v_i\} \quad \text{para } i = 1, 2 \quad v_1, v_2 \notin W \\ & W \subset J \quad e_1, e_2 \in \delta(W) \end{aligned} \tag{3.26}$$

Proposición 3.6. *Las desigualdades PC-2E-Hypotour* (3.26) *son válidas para el problema* PC-CLRP.

Demostración. Sea $ax \leq a_0$ una desigualdad válida para *PC-CLRP* con un vector de coeficientes enteros a, y F un conjunto de ejes tales que $ax = a_0 \Rightarrow x(F) \geq 1$.

Entonces, la desigualdad de hypotour $ax - x(F) \leq a_0 - 1$ es claramente válida para el problema *PC-CLRP*. Tomando $L = W \cup \{v_1, v_2\}$ y utilizando las desigualdades válidas PC-Subtour (3.19) se puede ver que las siguientes desigualdades son también válidas:

$$y(L' : I) + x(E(L)) \leq w(L'') \qquad L' \subseteq L'' \quad L'' \subset L \quad |L| = |L''| + 1 \tag{3.27}$$

$$y(L' : I) + x(E(L \setminus \{v_1\})) + x_{e1} \leq w(L'') \qquad L' \subseteq L'' \quad L'' \subset L \quad |L| = |L''| + 1 \tag{3.28}$$

$$y(L' : I) + x(E(L \setminus \{v_1, v_2\})) + x_{e1} + x_{e2} \leq w(L'') \qquad L' \subseteq L'' \quad L'' \subset L \quad |L| = |L''| + 1 \tag{3.29}$$

Las desigualdades anteriores valen para cualquier conjunto $L' \subseteq L'' \subset L$. Asumiendo que se puede derivar una desigualdad de hypotour a partir de (3.29) con $L' = W$, $L'' = L' \cup \{v_1\}$, $L = L'' \cup \{v_2\}$ y, utilizando un conjunto F adecuado, se puede expresar la desigualdad PC-2E-Hypotour como

$$y(W : I) + x(E(L \setminus \{v_1, v_2\})) - x(F) + x_{e1} + x_{e2} = y(W : I) + x(E(W)) - x(F) + x_{e1} + x_{e2} \leq w(L'') - 1$$

Se sabe que $w(L'') = w(L') + w(\{v_1\}) \leq w(L') + 1$ por lo tanto $w(L'') - 1 \leq w(L') = w(W)$ cumpliéndose lo buscado.

□

3.8. Cortes de Optimalidad

Las desigualdades o cortes de optimalidad son restricciones que sólo deberán ser satisfechos por las soluciones óptimas del problema y no por cualquier solución válida del mismo. Restringen el espacio de soluciones factibles eliminando soluciones válidas que, de acuerdo a determinadas características de las mismas, se sabe no podrán ser soluciones óptimas del problema.

3.8.1. Cortes Facility-degree

Los cortes Facility-degree introducidos en Belenguer y otros [16] son cortes de optimalidad que establecen la necesidad de utilizar una única variable y por depósito. Esta familia de cortes sólo es válida si los costos de los ejes cumplen la desigualdad triangular (TI-property). Se pueden describir para rutas de un solo cliente

$$y(i : S) \leq z_i \qquad S \subseteq J \quad d_h + d_j \leq Q \quad \forall h \neq j \in S \quad i \in I \tag{3.30}$$

En este caso puede verse que las restricciones son trivialmente válidas para soluciones óptimas ya que, al cumplirse que para todo par de clientes la suma de sus demandas no pueden exceder la capacidad del vehículo, en caso de haber más de una ruta unitaria las mismas deberían ser combinadas en una ruta de más de un cliente. De lo contrario la solución analizada no sería óptima.

Este tipo de cortes pueden también ser enunciados para rutas generales

$$2y(i : S') + x(i : S') + x(E(S)) \leq 2z_i + w(S') \quad S' \subset S \subseteq J \quad |S| = |S'| + 1 \quad i \in I \quad d(S) \leq Q \tag{3.31}$$

Esta es la adaptación para *PC-CLRP* de los cortes de optimalidad para rutas generales presentados en Belenguer y otros [16]. En caso de que $z_i = 0$ la desigualdad se cumple de manera trivial, debido a que $x(E(S)) \leq w(S) - 1 \leq w(S')$. Si $z_i = 1$, debido a la propiedad de desigualdad triangular a lo sumo puede haber una ruta que parta del depósito i y que atienda los clientes de S. Es decir, $2y(i : S') + x(i : S') + x(E(S)) \leq w(S) + 1 \leq w(S') + 2$ porque como máximo una ruta puede estar completamente contenida en $S \cup \{i\}$ (contando partida y llegada), y es por eso que se cumple lo buscado.

3.8.2. Cortes de depósitos en equilibrio

Se presentan ahora 4 familias de desigualdades que se usan como cortes de optimalidad y explotan distintas particularidades del problema *PC-CLRP*.

Debido a que el problema de Localización y Ruteo estudiado incorpora la noción de beneficios, se puede exigir para cada depósito activo que los costos que dicho depósito y sus rutas generan sean inferiores a los beneficios que otorgan los clientes atendidos desde el mismo. Esto puede expresarse con la siguiente familia de desigualdades.

$$\sum_{j \in J} b_j w_{ij} \geq O_i z_i + \frac{F}{2} x(\delta(i)) + F y(\delta(i)) + \sum_{e \in \delta(i)} c_e x_e + \sum_{e \in \delta(i)} 2 c_e y_e \qquad i \in I \tag{3.32}$$

Se pide que la suma de las ganancias de los clientes activos para un depósito supere el costo de apertura del depósito, los costos de rutas utilizadas y los costos de ruteo sobre los ejes salientes y entrantes al depósito. Es importante remarcar que no se están contemplando los costos internos de las rutas debido a que no es posible expresarlos con el modelo elegido.

3.8.3. Cortes de clientes en equilibrio

La segunda familia de cortes de optimalidad se basa en analizar el entorno de cada cliente. Con esta familia de desigualdades se pretende eliminar la posibilidad que se utilicen 2 ejes consecutivos en una ruta si los mismos no cumplen las ecuaciones de equilibrio que se describen a continuación. Para un cliente i se puede decir que su beneficio b_i debe ser mayor o igual al costo de los ejes en su corte $x(\delta(j))$. Sin embargo, esto por sí solo no alcanza debido a que hay situaciones donde eliminar un cliente de una ruta puede provocar mayores costos al resto de la misma. Formalizándolo, sea una ruta $r = ..., j, i, k, ...$, donde se tienen los ejes x_{ji}, x_{ik} activos, si se decide eliminar el nodo i entonces se obtendrá la nueva ruta $r' = ..., j, k, ...$ y los 2 ejes activos anteriores pasan a estar inactivos mientras que el eje x_{jk} se activa. Por lo tanto, se debe considerar el costo c_{jk} en la ecuación de equilibrio (véase figura 3.3). Los beneficios menos los costos de r' sólo serán superiores a los de r si se cumple que $b_i < c_{ji} + c_{ik} - c_{jk}$. En consecuencia, si se lo expresa como desigualdad lineal se puede decir que los dos ejes adyacentes al cliente i no podrán estar activos simultáneamente si se cumple la condición anterior. O sea, se verifica la siguiente desigualdad:

$$x_{ji} + x_{ik} \leq 1 \qquad i \in J \quad j, k \in V \quad j \neq k \quad b_i < c_{ji} + c_{ik} - c_{jk} \tag{3.33}$$

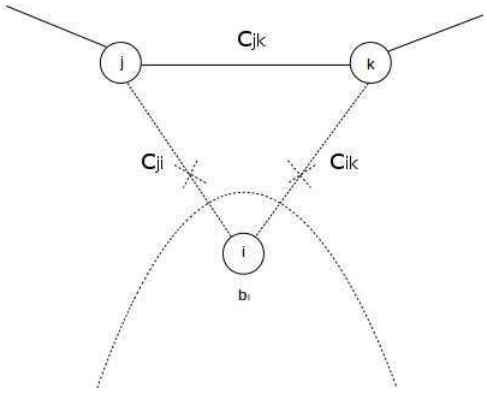

Figura 3.3: Restricción clientes en equilibrio

3.8.4. Cortes de activaciones implicadas 1

Al igual que las dos familias anteriores, la familia de desigualdades de activaciones implicadas es una familia de cortes de optimalidad que analiza que se satisfaga un equilibrio entre la ganancia que aporta un cliente y los costos que el mismo ocasiona. La primera familia de cortes de activaciones implicadas analiza los clientes cuya ganancia supera el costo de una ruta unitaria desde un determinado depósito. En este caso, lo que se pretende es forzar la activación de un cliente si el depósito adecuado está activado. La restricción puede escribirse de la siguiente manera:

$$\frac{W_i z_i - \sum_{l \in J \ l \neq k} d_l w_{il} - d_k}{W_i} \leq \sum_{j \in I} w_{kj} \qquad k \in J \quad i \in I \quad b_k >= F + 2c_{ik} \tag{3.34}$$

Esta restricción puede interpretarse de la siguiente manera. Si un cliente tiene una ganancia que supera al costo de una ruta unitaria con un depósito (contando el costo del vehículo) y, a su vez, el depósito i está activado y tiene suficiente capacidad como para albergar al cliente k, entonces se puede decir que el cliente debe estar activado (la suma de sus variables w debe ser mayor a 0). El lado izquierdo de la desigualdad (lhs) está escalado por la capacidad total del depósito para lograr que $lhs \leq 1$. Si la capacidad es superada, el lado izquierdo será negativo y el lado derecho puede tomar cualquier valor.

3.8.5. Cortes de activaciones implicadas 2

La idea anterior puede extenderse para ser aplicada a depósitos activados con una única ruta. Es decir, en este caso la capacidad a considerar es la de un vehículo, y por consiguiente, la ganancia debe sólo superar

al costo de una ruta unitaria. Esta familia de cortes asume que los costos cumplen la desigualdad triangular (TI-property). La desigualdades de activaciones implicadas 2 pueden escribirse como:

$$\frac{Qz_i - \sum_{l \in J \ l \neq k} d_l w_{il} - d_k}{Q} \leq \sum_{j \in I} w_{kj} \qquad k \in J \quad i \in I \quad b_k >= 2c_{ik} \quad d_k < W_i \tag{3.35}$$

Nuevamente el lado izquierdo es escalado para que se cumpla $lhs \leq 1$. Esta familia de cortes es bastante restrictiva ya que sólo se activa para depósitos activos con clientes atendidos que no superan la capacidad de un vehículo. Sin embargo, en algunas instancias se ha podido verificar su uso.

3.9. Algoritmo Branch&Cut

La formulación del problema *PC-CLRP*, las desigualdades y cortes adicionales presentados en la secciones 3.7 y 3.8 han sido embebidas en un algoritmo de planos de corte para producir una cota inferior válida LB. El algoritmo de planos de corte es la parte fundamental del algoritmo *Branch&Cut* que se propone para resolver el problema *PC-CLRP* de manera exacta. El esquema de planos de corte funciona de la siguiente manera. Inicialmente se construye un programa lineal LP con la función objetivo y un subconjunto de restricciones que se describirá en la sección siguiente. Luego, en cada iteración se resuelve el LP y se busca si hay alguna de las desigualdades válidas conocidas violadas por la solución óptima del LP. Si es así, se las agrega al LP en curso y se vuelve a iterar. El algoritmo de planos de corte termina cuando no se encuentra ninguna desigualdad violada.

Hay determinadas particularidades adicionales y detalles que diferencian a los algoritmos *Branch&Cut* de los de planos de corte. Las mismas serán analizadas en las siguientes secciones.

3.9.1. Relajación inicial

La relajación inicial incluye la función objetivo (3.3), las restricciones de grado (3.4), las restricciones (3.6), (3.7), (3.8), (3.9), (3.10) y (3.11), los cortes de optimalidad (3.32), las cotas en las variables y algunas restricciones adicionales que fueron agregadas para reforzar la relajación lineal y mejorar su cota inferior (LB):

- Caso particular de las desigualdades PC-Capacidad (3.18) con $S = J$.
- Se han agregado los cortes Facility-degree (3.30) para cada depósito $i \in I$ tomando $S = \{j \in J : d_j \leq Q/2\}$. Además, se han ordenado los clientes por demandas de menor a mayor y se ha construido un nuevo S recorriendo la lista ordenada. Se agregan clientes a S mientras se cumpla que la demanda de los últimos 2 clientes agregados sea menor o igual a la capacidad del vehículo Q. De esta forma

se cumple por construcción que $d_h + d_j \leq Q \quad \forall h \neq j \in S$. Esto genera nuevos cortes Facility-degree para cada depósito que son agregadas a la relación inicial.

- Restricciones que garantizan que cada depósito abierto sea usado:

$$x(\delta(i)) + 2y(\delta(i)) \geq 2z_i \qquad i \in I \tag{3.36}$$

Los cortes de clientes en equilibrio (3.33), de activaciones implicadas 1 (3.34) y de activaciones implicadas 2 (3.35) presentados anteriormente pueden ser agregadas a la formulación como parte de la relajación inicial o como cortes incorporados a medida que se los necesita en el algoritmo de planos de cortes. Se han analizado ambas alternativas y se ha decidido utilizarlos como cortes, ya que de esa manera se logra una relajación inicial con menos restricciones y, por ende, más rápida de optimizar. Al agregar las desigualdades como cortes, pudo comprobarse además que sólo un número reducido de los mismas era utilizado. Esta es una justificación adicional para utilizar las desigualdades como cortes en lugar de agregarlas a la relajación inicial.

3.9.2. Preprocesamiento

Al finalizar la construcción del primer *LP* y antes de optimizarlo, en los algoritmos *Branch&Cut* es útil analizar si es factible fijar variables utilizando condiciones lógicas extraídas de la instancia en proceso. En el algoritmo presentado, en una etapa de preprocesamiento se han verificado diversas condiciones que, en caso de cumplirse, determinan el fijado de variables para el resto del proceso. El objetivo principal de este enfoque es la simplificación del programa lineal fijando variables a valores estáticos, ya sea activándolas o desactivándolas.

Desactivado de las variables w

Si para cada par de ejes (i, j), (i, k) adyacentes al nodo cliente i se cumple la restricción $b_i < c_{ij} + c_{ik} - c_{jk}$ mostrada anteriormente en la sección "Restricciones de clientes en equilibrio", o se cumple que $d_i > Q$, es decir, que la demanda del cliente supera la capacidad de los vehículos; entonces se pueden desactivar las variables w_{si} $\forall_{s \in I}$ durante el curso de la optimización.

$$\forall_{s \in I} \quad w_{si} = 0 \qquad si \quad \forall_{j,k \in J} \ (b_i < c_{ij} + c_{ik} - c_{jk}) \quad o \quad d_i > Q \quad i \in J \tag{3.37}$$

Si un cliente i es desactivado para el resto de la optimización (desactivando sus variables w), lo mismo puede hacerse con todos ejes incidentes a dicho nodo. Es decir, se desactivarán las variables

$$\forall_{j \in J} \quad x_{ij} = 0 \qquad i \in J \tag{3.38}$$

$$\forall_{j \in I} \quad x_{ij} = 0 \quad y_{ji} = 0 \qquad i \in J \tag{3.39}$$

También puede desactivarse la variable w respectiva al depósito s si se cumple $d_i > W_s$, siendo W_s la capacidad del depósito s.

$$w_{si} = 0 \qquad si \quad d_i > W_s \quad i \in J \quad s \in I \tag{3.40}$$

Desactivado de las variables y

Si para algún cliente j y un depósito k se cumple que el beneficio $b_j < F + 2c_{jk}$, es decir, que el beneficio es menor a los costos de la ruta, entonces se podrá desactivar la variable y_{kj} por el resto de la optimización ya que la misma no participará de ninguna solución óptima.

$$y_{kj} = 0 \qquad si \quad b_j < F + 2c_{jk} \tag{3.41}$$

Desactivado de las variables z

Se podrá desactivar la variable z asociada a un depósito si se verifica que su costo de apertura es superior a las ganancias acumuladas de todos los clientes más el costo de usar un vehículo. No se puede asumir más de un vehículo utilizado ya que no se sabe cuántos clientes estarán efectivamente activos.

$$z_i = 0 \qquad si \quad \sum_{j \in J} b_j < O_i + F \tag{3.42}$$

Desactivar un depósito también implica la posibilidad de desactivar todos los ejes incidentes al mismo. Suponiendo que desactivó el depósito i, entonces pueden desactivarse también las variables

$$\forall_{j \in J} \quad x_{ij} = 0 \quad y_{ij} = 0 \tag{3.43}$$

3.9.3. Algoritmos de separación

En esta sección se detallarán los algoritmos de separación utilizados para identificar las desigualdades válidas violadas, introducidas en el capítulo anterior. En primer lugar se presentará la heurística general de cut lifting, introducida en Contardo y otros [21]. Esta heurística utiliza la estructura de algunas desigualdades para descomponer su algoritmo de separación en 2 subproblemas más fáciles de resolver. Luego se presentarán los algoritmos de separación utilizados para las distintas familias de desigualdades.

Heurística de cut lifting

Sea un polítopo $\mathcal{X} = \{x \in \mathbb{R}^n, Ax \leq b\}$ y se toma $\mathcal{Y} = conv(\mathcal{X} \cap \mathbb{Z}^n)$ como la cápsula convexa de los puntos enteros de $\mathcal{X}$. Dada una función $f : \mathbb{R}^n \to \mathbb{R}$ y un escalar $g \in \mathbb{R}$, se puede decir que la tupla (f, g) es una desigualdad válida para $\mathcal{Y}$ si $f(x) \leq g$ para todo $x \in \mathcal{Y}$. Dadas las funciones $f : \mathbb{R}^n \to \mathbb{R}$ y $h : \mathbb{R}^n \to \mathbb{R}$, se puede notar como $[f + h]$ a la función $[f + h](x) = f(x) + h(x)$. Si se asume que se tiene una familia de desigualdades válidas para $\mathcal{Y}$ definida por $\mathcal{F} = \{([\alpha_j + \beta_{jk}], \gamma_j) : j = 1 \dots \mathcal{J}, k = 1 \dots \mathcal{K}_j\}$ con $\beta_{jk}(\cdot) \geq 0$ para todo j, k; y suponiendo que $\mathcal{F}_1 = \{(\alpha_j, \gamma_j) : j = 1 \dots \mathcal{J}\}$ es fácil de separar, o equivalentemente que para cualquier $\epsilon \geq 0$ y $x \in \mathcal{X}$ es fácil de resolver el problema de decisión

$$\exists j \in \{1, \dots, \mathcal{J}\} \text{ tal que } \alpha_j(x) > \gamma_j - \epsilon \tag{P1}$$

entonces, se asume que para un $j \in \{1, \dots, \mathcal{J}\}$ y $x \in \mathcal{X}$ dados, el problema

$$\begin{aligned} &\max_k \beta_{jk}(x) \\ &\text{sujeto a} \quad k \in \mathcal{K}_j \end{aligned} \tag{P2}$$

es también fácil de resolver, o que al menos una buena cota inferior puede calcularse de manera eficiente. Entonces, dado un $x \in \mathcal{X}$, la siguiente heurística trata de encontrar un corte válido $([\alpha_j + \beta_{jk}], \gamma_j) \in \mathcal{F}$ que sea violado por x.

(I) Fijar $\epsilon > 0$ y usar un algoritmo de separación para el problema (P1) de manera de encontrar uno o más j tales que $\alpha_j(x) > \gamma_j - \epsilon$. Llámense a estos cortes ϵ–$\mathcal{F}_1$.

(II) Para cada j encontrado en el paso anterior resolver el problema (P2), obteniendo k.

Si $\alpha_j(x) + \beta_{jk}(x) > \gamma_j$ se ha encontrado una desigualdad violada.

Este procedimiento no exacto descompone el problema en dos subproblemas que, para determinadas familias de desigualdades, pueden llegar a corresponderse con algoritmos de separación conocidos. El problema (P2) puede, por lo general, ser resuelto como un problema knapsack $0 - 1$. Sin embargo, en las implementaciones presentadas en este trabajo, el mismo será siempre resuelto mediante heurísticas.

Separación de desigualdades CVRP

Como se ha mostrado anteriormente, cualquier solución factible del problema *CLRP* puede ser transformada a una solución factible de *CVRP*. Sin embargo, debido a que en este problema los clientes pueden estar atendidos o no (Prize-Collecting), es natural suponer que la solución de una relajación lineal de

un problema *PC-CLRP* puede transformarse a una solución de una relajación lineal para un problema *PC-CVRP*, pero la misma no puede convertirse directamente a *CVRP*. Es decir, en una solución de una relajación lineal no factible del problema *PC-CLRP* se puede ver que la suma de las variables adyacentes de algunos clientes puede ser inferior a 2 y de esta manera no ser una solución de una relajación lineal válida para el problema *CVRP* general.

Para separar las desigualdades de Capacidad, Hypotour y GLM se han utilizado los procedimientos provistos por las rutinas CVRPSEP de Lysgaard [66]. Se han analizado estos procedimientos de código abierto y no se han encontrado operaciones en los algoritmos que invaliden su uso para instancias que no cumplan el grado 2 anteriormente mencionado. Por lo tanto, se las ha modificado para que se adapten a cada una de las nuevas familias de desigualdades y con ello se han obtenido excelentes resultados.

Para poder hacer uso de las rutinas CVRPSEP se ha tenido que mapear los resultados lineales al formato de las formulaciones de 2 índices presentadas en el trabajo originario de Lysgaard y otros [67]. Allí, se utilizan variables enteras $[0, 1, 2]$ para representar los ejes entre los clientes y el depósito.

Sea $(\bar{x}, \bar{y}, \bar{w}, \bar{z})$ la solución óptima del *LP* en curso para el problema *PC-CLRP* y $G[\bar{x}, \bar{y}]$ el grafo soporte de la misma inducido por los ejes $\bar{x}$ y $\bar{y}$ con valor positivo. $\bar{w}$ y $\bar{z}$ representan a las variables w y z con valor positivo en el *LP* corriente. Si $\bar{u}_{ij}$ representa en CVRPSEP a los ejes (i, j) y d representa al único depósito, se debe entonces considerar el mapeo $\bar{u}_{ij} = \bar{x}_{ij}$ para los ejes $(i, j), i, j \in J$ y el mapeo $\bar{u}_{dj} = \sum_{i \in I} (\bar{x}_{ij} + 2\bar{y}_{ij})$ para los ejes $(d, j), j \in J$. En este último mapeo puede observarse cómo se acumula el flujo de los ejes incidentes a los distintos depósitos para convertirlo en una instancia de *CVRP*.

Separación de desigualdades PC-Capacidad, PC-Subtour y y-Capacidad

La separación de las desigualdades PC-Capacidad, PC-Subtour y y-Capacidad se ha dividido en 2 rutinas diferentes de separación. En la primera se separan las desigualdades PC-Capacidad y PC-Subtour, mientras que en la segunda se separan las desigualdades y-Capacidad.

En ambas rutinas se utiliza la heurística de cut lifting detallada anteriormente para hacer uso de los algoritmos eficientes de separación de desigualdades de Capacidad para el problema *CVRP*. El problema (P1) corresponde justamente a la separación de dichas desigualdades. Suponiendo que se encuentra un conjunto S que resuelve el problema de separación de cortes ϵ–*Capacidad*. El problema (P2) corresponde a encontrar un $S' \subset S$ que maximice $y(S' : I)$ minimizando al mismo tiempo el lado derecho de las desigualdades. Este problema puede escribirse como un problema de la mochila 0–1 (knapsack)

$$\begin{aligned} & \max_{\mu} \sum_{j \in S} \mu_j y(I:j) \\ \text{sujeto a} \quad & \sum_{j \in S} d_j \mu_j \leq rhs \\ & \mu \in 0, 1^{|S|} \end{aligned}$$

donde rhs referencia a los lados derechos de las correspondientes desigualdades PC-Capacidad, PC-Subtour y y-Capacidad. El problema podría entonces resolverse a optimalidad utilizando algún software específico, como ser el algoritmo COMBO de Martello y otros [70]. Sin embargo, dado la simplicidad del problema y del uso de características particulares de cada desigualdad se ha decidido resolver (P2) utilizando heurísticas basadas en la información de las instancias.

Para la separación de las desigualdades PC-Capacidad y PC-Subtour se utilizan las rutinas CVRPSEP mencionadas anteriormente. Primero se testeó la eficacia de implementarlas utilizando las rutinas tal cual habían sido diseñadas originalmente. Al llamar a la rutina correspondiente, se obtienen distintos conjuntos de clientes S sobre los cuales se pueden generar las desigualdades de Capacidad. Se observó que los conjuntos generados frecuentemente no violaban las desigualdades PC-Capacidad y PC-Subtour, por lo que se consideró la modificación de las rutinas para mejorar dicha situación. Para poder adaptarlas fue necesario suministrar a la rutina de separación el valor de las variables $\bar{w}$. De esa manera, se pudo modificar el chequeo interno de la rutina para verificar conjuntos de clientes S acordes a los lados derechos de las desigualdades PC-Capacidad (3.18) y PC-Subtour (3.19).

Para la separación de cortes de Capacidad, las rutinas de separación CVRPSEP implementan 4 heurísticas aplicadas secuencialmente. Los detalles de implementación de éstas pueden encontrarse en [67] aunque a continuación se describen las ideas fundamentales utilizadas.

1. La primera heurística calcula las componentes conexas del grafo soporte $G[\bar{x}, \bar{y}]$. Se calculan las distintas componentes conexas $S_1, \ldots, S_p$ y se verifica para cada $i = 1, \ldots, p$, la violación de las desigualdades de capacidad (respectivamente PC-Capacidad para la implementación adaptada) sobre el conjunto de nodos S_i y sobre el conjunto $J \setminus S_i$. Si se encuentra alguna desigualdad violada en este paso se sale del algoritmo de separación informando el resultado.

2. Luego, se aplica un proceso de contracción (shrinking) para disminuir la cantidad de nodos a analizar y mejorar el desenvolvimiento de las heurísticas restantes. Sea $GS = (VS, ES)$ el grafo obtenido,

y $GS_c = (VS_c, ES_c)$ el grafo obtenido sin el nodo depósito. Cada uno de los nodos de VS_c son denominados super-vértices. La segunda heurística separa las llamadas desigualdades de Capacidad Fraccionales. Estas desigualdades son obtenidas si se reemplaza, en el lado derecho de las desigualdades de capacidad, $k(S)$ por $d(S)/Q$ (por su parte, las desigualdades de PC-Capacidad ya son desigualdades fraccionales). Aquí, CVRPSEP utiliza una heurística que ejecuta sucesivas veces un algoritmo exacto que resuelve problemas de máximo flujo (max-flow / min-cut) sobre el grafo GS_c. La heurística utiliza apropiadamente el algoritmo tradicional de máximo flujo para separar distintos cortes de capacidad fraccional.

3. La tercera heurística aplica un método goloso que construye de manera iterativa conjuntos S que combinan de distinta manera los super-vértices de VS_c. Este método intenta encontrar desigualdades de capacidad violadas para conjuntos de nodos de mayor tamaño.

4. La última heurística recorre las desigualdades de capacidad encontradas en los pasos anteriores e intenta generar nuevas desigualdades efectuando modificaciones a las mismas.

Cuando se generan desigualdades PC-Subtour se calcula el conjunto S' como el conjunto de todos los clientes menos el que contiene el menor valor de $\bar{y}$, es decir $S' = S \setminus \{ arg\,\text{mín}_{i \in S}\, \bar{y}_i\}$. El lado derecho, por su parte se calcula como $S'' = S \setminus \{ arg\,\text{máx}_{i \in S'}\, \bar{w}_i\}$ (el cliente restado no puede ser el mismo que el dejado afuera para calcular S').

Para las desigualdades PC-Capacidad, el problema (P2) se resuelve utilizando una construcción heurística de S'. Se recorren los clientes tomando $\bar{y}(j) = \sum_{i \in I} \bar{y}_{ij} \quad \bar{w}(j) = \sum_{i \in I} \bar{w}_{ij} \quad j \in S$ y se incorporan a S' sólo los clientes j que cumplen $\bar{y}(j) > \bar{w}(j) - \frac{d_j}{Q}\bar{w}(j)$. De esta manera, se construye S' con los clientes que aumentan la violación de la desigualdad.

La segunda rutina de separación efectúa la identificación de las desigualdades y-Capacidad (3.20) violadas. Esta rutina hace uso de la rutina CVRPSEP original. Para resolver el problema (P2) correspondiente, se aplica una heurística similar a la aplicada anteriormente para PC-Capacidad y PC-Subtour. Se analizan los distintos conjuntos S identificados.

(a) Si $d(S) <= Q$, el conjunto S' se calcula, al igual que antes, como $S' = S \setminus \{ arg\,\text{mín}_{i \in S}\, \bar{y}_i\}$ y el lado derecho como $|S| - 1$ (implementando de esta manera desigualdades y-Subtour).

(b) Si $d(S) > Q$, se calcula S' aplicando una heurística constructiva similar a la implementada en Belenguer y otros [16]. Se ordenan los clientes en orden decreciente en función de los valores

$\bar{y}(j) = \sum_{i \in I} \bar{y}_{ij} \quad j \in S$. Se recorren los clientes ordenados y se construye iterativamente S' hasta que deje de cumplirse $k(S \setminus S') = k(S)$.

Con los conjuntos S y S' se construye la desigualdad y-Capacidad (y-Subtour respectivamente) y se verifica su violación.

Separación de desigualdades w-z

Las desigualdades w-z (3.17) se separan por simple enumeración de las variables activas. Para ello, se verifica la violación de las restricciones w-z sobre cada uno de los depósitos activos z_i y las variables representantes de clientes activos w_{ij}.

Separación de restricciones de lógica de ruta

Las restricciones de lógica de ruta (3.12) se separan por simple enumeración de las variables activas. Para ello, se verifica para cada eje activo la violación de las restricciones. El costo del algoritmo de separación es $O(|J|^2|I|)$. Los resultados computaciones demuestran que el algoritmo de separación es eficiente y tiene poco impacto en el tiempo de cómputo del algoritmo general.

Separación cortes Facility-degree

Algunos cortes Facility-degree (3.30) han sido incluidos en la relajación inicial. Sin embargo, los cortes Facility-degree (3.31) deben ser separados e incorporados al algoritmo de planos de corte cuando son necesarios.

El algoritmo de separación implementado para los cortes (3.31) utiliza las ideas del trabajo de Contardo y otros [21]. Se implementa una heurística que repite para cada depósito i el siguiente procedimiento: se inicializa un conjunto $S = \emptyset$. Iterativamente se va incorporando a S el cliente $j \notin S$ que maximice la cantidad $2\bar{y}_{ij} + \bar{x}_{ij} + \bar{x}(S : j)$. El algoritmo termina si $d(S) > Q$ o una desigualdad (3.31) ha sido identificada.

Además del procedimiento anterior, tanto los cortes Facility-degree (3.30) como los cortes Facility-degree (3.31) son verificados para cada conjunto S construido que cumpla $d_h + d_j \leq Q \quad \forall h \neq j \in S$ o que cumpla $d(S) \leq Q$ dentro de las rutinas de separación de los cortes PC-Capacidad y y-Capacidad.

Separación de desigualdades 3-Path

La separación de las desigualdades 3-Path (3.21) se realiza analizando todos los subconjuntos de 3 clientes que tengan sus variables w con valores fraccionales. Este algoritmo se ha implementado teniendo el especial cuidado de no repetir conjuntos y es por ello que el mismo tiene costo $O(|J|^3)$. En la práctica,

el algoritmo de separación ha mostrado ser eficiente y tiene poco impacto en el tiempo de cómputo del algoritmo general. Esto se debe fundamentalmente a que sólo se analizan los clientes i activos $\bar{w}(i) > 0$ y los ejes activos adyacentes $\bar{x}_e > 0, \quad e \in \delta(i)$.

Separación de desigualdades 1 Co-Circuit y Co-Circuit

La rutina de separación de desigualdades Co-Circuit (3.22) verifica primero si se encuentran violadas algunas desigualdades 1 Co-Circuit. Si es así, se sale informando las mismas.

Para separar las desigualdades 1 Co-Circuit se ha implementado un algoritmo que analiza los ejes activos adyacentes a cada nodo (clientes y depósitos). Para cada nodo $i \in J \cup I$ se verifica si existe un eje $e \in \delta(i)$ tal que $\bar{x}_e > (\sum_{f \in \delta(i)} \bar{x}_f)/2$. En caso de existir un eje con dichas características, se toma $S = \{i\}$, $F = \{e\}$ y se genera la restricción 1 Co-Circuit correspondiente.

La separación de las desigualdades Co-Circuit generales se realiza siguiendo los lineamientos de Belenguer y otros [16] para la separación de cortes similares en el problema *CLRP*. Inicialmente, el método fue presentado por Letchford y otros [63] para separar las desigualdades Blossom en el problema *b-matching*.

El procedimiento de separación se basa en la utilización de un algoritmo exacto para el cálculo del árbol de cortes mínimos. Para ello, se han utilizado las librerías C++ *Library for Efficient Modeling and Optimization in Networks - LEMON* versión 1.3.1 [28] que implementan el algoritmo de Gomory-Hu para tal fin. Las librerías *LEMON* forman parte de la iniciativa de código abierto COIN-OR.

Para comprender mejor el funcionamiento del algoritmo de separación conviene reescribir las desigualdades Co-Circuit (3.22) de la siguiente manera:

$$\sum_{e \in \delta(S) \setminus F} x_e + \sum_{e \in F} (1 - x_e) \geq 1 \qquad \forall S \subseteq J \cup I, \quad \forall F \subseteq \delta(S), \quad |F| \text{ impar} \tag{3.44}$$

Con el algoritmo de Gomory-Hu se calcula el árbol de cortes mínimos T sobre el grafo inducido por los ejes $e \in E$ tales que $\bar{x}_e > 0$, determinando las capacidades por eje como $h_e = \text{mín}\{\bar{x}_e, 1 - \bar{x}_e\}$. Cada eje f del árbol resultante representa un corte mínimo. Sea S^f el conjunto de nodos que definen un corte mínimo asociado al eje f del árbol T, el peso de este corte es igual a la parte izquierda de la desigualdad anterior (3.44). Se recorren todos los ejes f del árbol T verificando para cada uno la violación de cortes Co-Circuit, es decir, cortes con peso menor a 1.

El proceso anterior determina los conjuntos S candidatos a violar cortes Co-Circuit. Para poder formular las desigualdades resta encontrar los conjuntos de ejes $F \subseteq \delta(S)$. Entonces, para cada corte se necesita

encontrar un conjunto F que minimice el lado izquierdo de (3.44). Para ello se toma $F = \{e \in \delta(S) : \bar{x}_e \geq 0{,}5\}$.

Si $|F|$ es impar, el conjunto formado es el buscado. Por otra parte, si $|F|$ es par, se debe eliminar un eje de F o agregar uno de $\delta(S) \setminus F$ para hacer que $|F|$ sea impar. El eje elegido será el que minimiza la variación del lado izquierdo de (3.44).

Al finalizar, se devuelven las desigualdades de Co-Circuit correspondientes a cada par de conjuntos S y F encontrados que violan (3.44).

Separación desigualdades PC-path-elimination

Para separar las restricciones PC-path-elimination se han utilizado las ideas de Belenguer y otros [16]. Para cada par de nodos $\{j,l\} \quad j,l \in J$ se deben calcular un conjunto S y otro I' que maximicen la parte izquierda de la restricción (3.23).

Puede verse que la parte izquierda puede dividirse en 2 partes. La primera, $A(j,l,I') = x(\{j\} : I') + x(\{l\} : I \setminus I')$ depende sólo de la elección de I' mientras que la segunda $B(j,l,S) = y(S \cup \{l,j\} : I) + x(E(S \cup \{l,j\}))$ depende de la elección de S. Sea $(\bar{x},\bar{y},\bar{w},\bar{z})$ la solución óptima del LP en curso. Es fácil ver que la primera parte alcanza su máximo si $I' = \{i \in I : \bar{x}_{ij} \geq \bar{x}_{il}\}$. Si $I' = I$ o $I' = \emptyset$, entonces no puede haber PC-path-elimination violados para el par de nodos $\{j,l\}$. En caso contrario, es necesario ver cómo calcular S para maximizar $B(j,l,S)$.

Sumando las restricciones (3.11) del modelo (PC2E) sobre cada nodo de $S \cup \{l,j\}$ se sabe que

$$2\bar{w}(S) + 2\bar{w}(\{l\}) + 2\bar{w}(\{j\}) = 2\bar{y}(S \cup \{l,j\} : I) + 2\bar{x}(E(S \cup \{l,j\})) + \bar{x}(\delta(S \cup \{l,j\})) \tag{3.45}$$

De las ecuaciones (3.45) puede verse que maximizar $B(j,l,S)$ equivale a minimizar $\bar{x}(\delta(S \cup \{l,j\}))$. Por eso, el problema de maximizar S se reduce a encontrar conjuntos de clientes $S' = S \cup \{l,j\}$ de manera tal que $\bar{x}(\delta(S'))$ sea mínimo. Para ello, se ha utilizado un algoritmo de máximo flujo (max-flow / min-cut) que se aplica sobre un grafo apropiadamente definido. La implementación de dicho algoritmo utiliza las rutinas CVRPSEP de Lysgaard [66]. Se ha modificado la subrutina que calcula máximo flujo para que considere las partes derechas de las restricciones PC-path-elimination (3.23) tal como han sido definidas para el problema *PC-CLRP*.

Sea $G[\bar{x}]$ el grafo soporte construido con los ejes $\bar{x}_e > 0 \quad e \in E$. Se construye un grafo G' agregándole al grafo soporte un nodo fuente s que se conecta a los nodos clientes l y j con ejes de capacidad infinita. Se agrega también otro nodo destino t conectado a los nodos depósitos también con ejes de capacidad infinita. Calculando el mínimo corte que separa los nodos s y t en el grafo G' se obtiene el conjunto de nodos del corte. Se toma S' como los nodos del lado izquierdo del corte (lado de s) menos el nodo fuente s. Si el corte

$\bar{x}(\delta(S'))$ es menor a $2A(j, l, I')$, entonces la restricción PC-path-elimination definida para $S = S' \setminus \{l, j\}$, el par de clientes $\{l, j\}$ y los depósitos I', se encuentra violada. De lo contrario, no hay restricción PC-path-elimination violada para los clientes $\{l, j\}$.

Este algoritmo de separación fué utilizado en los experimentos, sin embargo, como fue mencionado anteriormente, estas desigualdades no han sido incorporadas al algoritmo *Branch&Cut* final, dejándose las desigualdades PC-path-elimination fuera del mismo.

Separación desigualdades PC-GLM

El algoritmo de separación de las desigualdades PC-GLM (3.24) utiliza la heurística de cut lifting previamente detallada. De esta manera se busca hacer uso de los algoritmos eficientes de separación de desigualdades GLM presentes en las rutinas CVRPSEP de Lysgaard [66]. El problema (P1) corresponde justamente a la separación de dichas desigualdades utilizando CVRPSEP. Suponiendo que se encuentra un conjunto N que resuelve el problema de separación de cortes ϵ–PC–GLM. El problema (P2) corresponde a encontrar un $N' \subset N$ que maximice $y(N' : I)$ minimizando al mismo tiempo el lado derecho de las desigualdades. En este caso, se calcula N' como el conjunto de todos los clientes menos el que contiene el menor valor de $\bar{y}$, es decir $N' = N \setminus \{ \, arg\,\text{mín}_{i \in N} \, \bar{y}_i \}$.

Se ha modificado la rutina CVRPSEP para que contemple la violación del lado derecho de la desigualdad PC-GLM (3.24). Esta rutina implementa un algoritmo de separación polinomial basado en la resolución exacta de problemas de máximo flujo (max-flow / min-cut).

Separación desigualdades PC-2E-Hypotour

Para implementar el algoritmo de separación de las desigualdades PC-2E-Hypotour (3.26) se ha utilizado la rutina CVRPSEP de Lysgaard [66] diseñada para separar desigualdades Hypotour en el problema $CVRP$. El algoritmo retorna de manera explícita los coeficientes de cada uno de los ejes involucrados en el corte. Para poder construir el corte PC-2E-Hypotour adaptado, ha sido necesario reconstruir con dicha información el conjunto de clientes W. Esto es factible si se analiza cada uno de los ejes involucrados y se contabiliza la aparición de cada nodo en el corte. Se incrementa el contador en los ejes con coeficientes positivos y se decrementa el mismo en los ejes con coeficientes negativos. Al finalizar, se puede verificar que los nodos con valor en el contador mayor o igual a 2 se corresponden con los nodos del conjunto W.

Se ha modificado la rutina CVRPSEP para que contemple la violación del lado derecho de la desigualdad PC-2E-Hypotour (3.26) utilizando las variables $\bar{w}$. El algoritmo de separación puede verse en detalle en

Lysgaard y otros [67]. El mismo aplica diversas heurísticas para construir conjuntos candidatos W. El costo total del algoritmo de separación es $O(|J|^2)$.

Separación cortes de optimalidad

Los cortes de optimalidad de clientes en equilibrio (3.33), los cortes de activaciones implicadas 1 (3.34) y los cortes de activaciones implicadas 2 (3.35), como se mencionó en la sección 3.9.1, pueden ser incorporadas directamente a la relajación inicial o utilizadas como cortes identificados en el algoritmo de planos de cortes. Se presentan aquí los algoritmos de separación utilizados.

El algoritmo de separación para los cortes (3.33) analiza, para cada cliente, todos los pares de ejes adyacentes al mismo que pueden llegar a generar una desigualdad violada. Este algoritmo se ha implementado teniendo el especial cuidado de no repetir conjuntos y es por ello que el mismo tiene costo $O(|J|^3)$. En la práctica, el algoritmo de separación ha mostrado ser eficiente y tiene poco impacto en el tiempo de cómputo del algoritmo general.

El algoritmo que se utilizó para separar los cortes (3.34) y (3.35) es un simple algoritmo de separación por enumeración. Se verifica para todos los clientes que cumplen la condición de superar los costos con sus ganancias. Se suman las variables involucradas y se verifica si cada una de las desigualdades es violada. El costo de este algoritmo de separación es $O(|J||I|)$ y en la práctica resulta ser muy eficiente.

3.9.4. Estrategia de separación

La estrategia de separación constituye uno de los aspectos más importantes de un algoritmo *Branch&Cut*. Siguiendo las estrategias utilizadas en trabajos previos para algoritmos *Branch&Cut* en problemas de ruteo, se ha decidido independizar la estrategia efectuada en el nodo raíz del árbol de enumeración (nodo 0) de la efectuada en el resto de los nodos.

Para poder evaluar diferentes secuencias de separación de las distintas familias de cortes y, además, poder considerar distintas estrategias de reoptimización al identificar cortes de una familia, se incorporó al algoritmo la posibilidad de activar y desactivar cada uno de los cortes por línea de comando. Además se incorporó un parámetro para estipular el orden utilizado por la estrategia de separación en el nodo raíz y otro parámetro para especificar el orden del resto de los nodos. Por último, se incorporó también la posibilidad de indicar mediante un parámetro de linea de comando la acción a tomar inmediatamente después de finalizar la separación de cada una de las familias de desigualdades. Las posibles acciones a considerar son:

- No hacer nada y pasar a evaluar la próxima familia de desigualdades.
- Si se identificó algún corte en la separación de la última familia o en las previas, re-optimizar.
- Si se identificó algún corte en la separación de la última familia o en las previas y la violación es mayor a 0,1, re-optimizar.
- Si se identificó algún corte en la separación de la última familia o en las previas y la violación es mayor a 0,2, re-optimizar.
- Si se identificaron más de 100 cortes en la separación de la última familia o en las previas, re-optimizar.

Con estos parámetros se ha podido testear gran cantidad de combinaciones de estrategias de separación. Para poder identificar la mejor, como herramienta de soporte, se utilizó el software *IRACE* versión 1.06 (empleado también en el capítulo 2). Como se mencionó en el capítulo anterior, esta herramienta permite buscar de manera automática las mejores configuraciones para un problema de optimización. Utilizando el mecanismo creado para estipular por línea de comando las distintas estrategias de separación en el algoritmo *Branch&Cut* propuesto, se ejecutó *IRACE* para evaluar distintas combinaciones válidas. Las mejores estrategias reportadas por *IRACE* fueron luego testeadas para verificar su eficiencia y se realizaron ajustes hasta obtener la estrategia final presentada.

La estrategia elegida para el nodo raíz ejecuta las rutinas de separación en el siguiente orden:

I. Se buscan desigualdades violadas w-z (3.17), luego cortes de optimalidad de activaciones implicadas 1 (3.34), y 2 (3.35).

II. Se buscan restricciones de lógica de ruta (3.12).

III. Se analiza la violación de desigualdades Co-Circuit (3.22).

IV. Se buscan desigualdades 3-Path (3.21). Si hasta este punto se encontró algún corte, re-optimizar y reiniciar la separación.

V. Se buscan desigualdades violadas PC-Capacidad (3.18) y PC-Subtour (3.19).

VI. Se buscan desigualdades violadas y-Capacidad (3.20).

VII. Se buscan cortes de optimalidad Facility-Degree (3.31).

VIII. Se buscan cortes de optimalidad de clientes en equilibrio (3.33).

IX. Se buscan desigualdades PC-2E-Hypotour (3.26).

X. Se buscan desigualdades PC-GLM (3.24). Si hasta este punto se encontró algún corte, re-optimizar y reiniciar la separación. Si no hay cortes, pasar a la etapa de branching.

La estrategia que finalmente se eligió para el resto de los nodos ejecuta las rutinas de separación en el siguiente orden:

I. Se buscan desigualdades violadas w-z (3.17), luego cortes de optimalidad de activaciones implicadas 1 (3.34), y 2 (3.35).

II. Se buscan restricciones de lógica de ruta (3.12).

III. Se buscan desigualdades violadas y-Capacidad (3.20). Si hasta este punto se encontró algún corte, re-optimizar y reiniciar la separación.

IV. Se analiza la violación de desigualdades Co-Circuit (3.22). Si en la separación de Co-Circuit se encontró algún corte con una violación de al menos 0,2, re-optimizar y reiniciar la separación.

V. Se buscan cortes de optimalidad de clientes en equilibrio (3.33).

VI. Se buscan desigualdades 3-Path (3.21). Si hasta este punto se encontró algún corte, re-optimizar y reiniciar la separación.

VII. Se buscan desigualdades PC-GLM (3.24).

VIII. Se buscan desigualdades PC-2E-Hypotour (3.26). Si hasta este punto se encontró algún corte, re-optimizar y reiniciar la separación. Si no hay cortes, pasar a la etapa de branching.

Como puede observarse, en la separación de nodos no se buscan desigualdades violadas PC-Capacidad ni cortes Facility-Degree. Esto disminuye los tiempos de separación y, según pudo observarse en los experimentos, no ocasiona un impacto considerable en las cotas del algoritmo *Branch&Cut* completo. Se ha estipulado que las rutinas de separación de los cortes PC-GLM y PC-2E-Hypotour se llamen como máximo 10 veces por nodo. Esta restricción ha sido impuesta para limitar el esfuerzo computacional empleado en cada nodo del árbol de enumeración.

3.9.5. Estrategia de branching

Uno de los aspectos fundamentales a considerar en un algoritmo *Branch&Cut* es el algoritmo de branching utilizado. Existen diferentes opciones a la hora de diseñar la estrategia de branching. En este algoritmo, en función de la experimentación realizada, se decidió efectuar branching jerárquico sobre variables y desigualdades.

Sea $(\bar{x}, \bar{y}, \bar{w}, \bar{z})$ la solución óptima del LP en curso al momento de aplicar el branching. Para aplicar branching jerárquico, en primer lugar se buscan variables de apertura de depósito $\bar{z}$ que sean fraccionales. Se hace branching sobre la variable más cercana al valor 0,5.

Si no hay variables $\bar{z}$ fraccionales, se pasa a analizar 2 estrategias diferentes. Primero, se analiza el flujo saliente de cada depósito. Es decir, se analiza para cada $i \in I$ la suma de las variables $\bar{x}_{ij}$ con $j \in J$. Se selecciona el depósito que tenga flujo total más cercano a un número impar. En otras palabras, para cada depósito i con flujo saliente f_i, se selecciona el depósito cuyo flujo está más cerca de $\lfloor f_i/2 \rfloor * 2 + 1$.

Antes de decidir el branching sobre las desigualdades de flujo saliente de depósitos, se pasa a analizar la segunda estrategia. La misma busca para cada cliente $j \in J$ la suma de las variables $\bar{w}_{ij}$. En caso de encontrar algunos clientes cuya suma sea fraccional, se elige el que tenga la suma más cercana a 0,5.

Juntando la infomación calculada en ambas estrategias se verifica si el menor flujo obtenido por la estrategia 1 es inferior al parámetro general *branch*23 y, además, dicho flujo es inferior a la menor diferencia obtenida en la estrategia 2. Si es así, se efectúa el branching sobre la estrategia 1 y se generan 2 hijos: el primero con la desigualdad $\sum_{j \in J} \bar{x}_{ij} \leq \lfloor f_i/2 \rfloor * 2$ y el segundo con la desigualdad $\sum_{j \in J} \bar{x}_{ij} \geq \lfloor f_i/2 \rfloor * 2 + 2$, siendo f_i el flujo del depósito elegido i.

En caso contrario, si hay algún cliente con variables w fraccionales, se hace branching sobre la estrategia 2. Se generan 2 hijos: el primero con la desigualdad $\sum_{i \in I} \bar{w}_{ij} = 0$ y el segundo con la desigualdad $\sum_{i \in I} \bar{w}_{ij} = 1$. Este branching tiene como objetivo analizar subproblemas con determinado cliente activo/inactivo.

Si ninguna de las condiciones anteriores se cumple, entonces se entrega el control a CPLEX para que efectúe branching sobre variables. Para el branching sobre variables de CPLEX se ha establecido la jerarquía:
1. variables x 2. variables w 3. variables y

Se ha configurado CPLEX para que utilice Strong Branching y seleccione de esa manera la variable que mejor resultados ofrezca (esto sólo se aplica al branching controlado por CPLEX y no a las estrategias manuales enunciadas anteriormente). Para ello, CPLEX selecciona un conjunto de variables candidatas y realiza algunas iteraciones de su optimizador lineal simulando la selección de cada una de ellas. Luego se queda con la variable que mejores resultados ha mostrado y que más ha reducido la cota inferior (LB) de la

optimización. Si bien este mecanismo de branching es costoso, el mismo produce un árbol de enumeración mucho más reducido. Se han obtenido mejores resultados globales del algoritmo aplicando esta técnica frente a las otras alternativas que proporciona CPLEX (pseudocostos, pseudocostos reducidos y variable de máxima infactibilidad).

El esquema completo de la estrategia de branching implementada puede observarse en el pseudocódigo 3.10.

Algoritmo 3.10 Estrategia de Branching

Input: $(\bar{x}, \bar{y}, \bar{w}, \bar{z})$ solución LP en curso

1: Buscar alguna variable $\bar{z}$ fraccional. Si existe hacer branching sobre la más cercana a 0.5. Salir.
2: *Estrategia 1:* Buscar algún depósito $i \in I$ con flujo entero impar o fraccional ($f_i = \sum_{j \in J} \bar{x}_{ij}$).
Elegir el i más cercano a $\lfloor f_i/2 \rfloor * 2 + 1$ y hacer $dif_1 = |\lfloor f_i/2 \rfloor * 2 + 1 - f_i|$
3: *Estrategia 2:* Buscar algún cliente $j \in J$ tal que la $f_j = \sum_{i \in I} \bar{w}_{ij}$ sea fraccional.
Elegir el cliente j con f_j más cercano a 0,5 y hacer $dif_2 = |f_j - 0{,}5|$
4: **if** existe(i) **and** $dif_1 < branch23$ **and** $dif1 < dif2$ **then**
5: *Branching 1:* Si se encontró un depósito i generar un hijo con las desigualdad $\sum_{j \in J} \bar{x}_{ij} \leq \lfloor f_i/2 \rfloor * 2$ y otro con $\sum_{j \in J} \bar{x}_{ij} \geq \lfloor f_i/2 \rfloor * 2 + 2$. Salir.
6: **else**
7: *Branching 2:* Si se encontró un cliente j generar un hijo con las desigualdad $\sum_{i \in I} \bar{w}_{ij} = 0$ y otro con $\sum_{i \in I} \bar{w}_{ij} = 1$. Salir.
8: **end if**
9: Pasar el control a CPLEX para hacer branching sobre variables

Diferentes parámetros $branch23$ fueron testeados. El que mejor resultados ofreció en promedio es el valor 0,45. En la sección Resultados Computacionales 3.10 se mostrarán los resultados de las diferentes alternativas analizadas.

En contraposición a lo analizado, se ha testeado también la alternativa de hacer branching sobre conjuntos de nodos. Esta técnica ha sido utilizada con éxito en algoritmos *Branch&Cut* para varias familias de problemas de ruteo como *CVRP* y *CLRP* y se la conoce como *branching on cutsets*. Se basa en buscar conjuntos de nodos tal que los ejes del corte sumen una cantidad fraccional o impar. Sea un conjunto de clientes $S \subseteq J$, y suponiendo que $\delta(S) = f$, se generan 2 nuevos nodos: uno con la restricción $\delta(S) \leq \lfloor f/2 \rfloor * 2$ y el otro con la restricción $\delta(S) \leq \lfloor f/2 \rfloor * 2 + 2$. En Belenguer y otros [16] se aplica esta idea convirtiendo las desigualdades y-Capacidad en igualdades, para luego hacer branching a partir de las mismas sobre las variables slack generadas. Esto tiene como ventaja la posibilidad de instruir a CPLEX para que efectúe Strong Branching sobre las variables slack y seleccionar el branching sobre la variable que ofrezca mejores perspectivas futuras. Sin embargo, tanto la técnica de generar conjuntos de clientes como la de convertir las

desigualdades y-Capacidad en igualdades no mostraron, en la implementación para el problema *PC-CLRP* presentada, mejores resultados que el branching jerárquico detallado.

3.9.6. Selección de nodos

Se ha configurado CPLEX para que utilice la estrategia de selección de nodos *Best Bound Search*. Esto significa que CPLEX va a seleccionar como nodo siguiente el que tenga la menor cota inferior LB. Se sabe que esta técnica produce el árbol de enumeración más pequeño.

3.9.7. Heurística inicial

Se utilizó como heurística inicial el algoritmo metaheurístico presentado en el capítulo 2. El algoritmo *Branch&Cut* implementado permite recibir como parámetro la mejor cota superior conocida (Z^{ub}) para la instancia evaluada. Se utiliza como cota superior UB la mejor solución obtenida mediante el algoritmo metaheurístico de colonia de hormigas descripto en el capítulo 2.

En caso de no suministrarse un parámetro con una cota superior, el algoritmo comienza su ejecución inicializando la cota superior en $+\infty$ (0 en caso de tratarse de una instancia *PC-CLRP* y $+\infty$ en caso de ser una instancia *CLRP*).

3.9.8. Heurística primal

Si al procesar una instancia la cota superior suministrada como parámetro no es lo suficientemente buena o no se ha suministrado la misma, es necesario tratar de mejorarla lo más rápido posible para evitar explorar sectores del árbol *Branch&Cut* irrelevantes para la determinación del mejor resultado. Para esto, las heurísticas primales (llamadas también heurísticas de mejoramiento) han mostrado ser muy útiles. Las mismas permiten intentar generar soluciones enteras factibles utilizando la información provista por la solución de la relajación lineal del LP en curso. Es importante tratar de encontrar soluciones enteras factibles que permitan mejorar las cotas superiores. Sin embargo, el tiempo computacional empleado en esta tarea no debe ser excesivo. La utilización de heurísticas primales debe balancear la efectividad de las soluciones encontradas versus el esfuerzo computacional efectuado para buscarlas.

El algoritmo *Branch&Cut* presentado utiliza una heurística primal de 3 fases que itera 6 veces sobre las mismas generando diferentes alternativas y posibles soluciones. Para ello, en cada iteración se genera un número aleatorio r con distribución uniforme $[0, 1]$. Posteriormente, se construye una solución factible utilizando la información del LP y la semilla aleatoria r. Luego, se trata de mejorar esta solución mediante una técnica de mejoramiento basada en una adaptación de la tradicional heurística de ahorros de Clark y

Wright [20]. Por último, se aplican operadores de búsqueda local para tratar de mejorar la solución hasta encontrar un óptimo local. Los detalles generales de la heurística pueden observarse en el algoritmo 3.11.

Algoritmo 3.11 Heurística Primal

Input: x^* solución relajación LP

1: **for** each $k = 1 \dots 6$ **do**
2: $r \leftarrow randomUniform(0 \dots 1)$
3: $s \leftarrow construccionC\&W(x^*)$
4: $s \leftarrow mejoramientoC\&W(s)$
5: $s \leftarrow mejoramientoBusquedaLocal(s)$
6: **if** $costo(s) < costoMejorSolucion$ **then**
7: $actualizarMejorSolucion(s)$
8: **end if**
9: **end for**

Fase constructiva de la heurística primal

El algoritmo constructivo utiliza la información del LP en curso para generar una solución factible del problema. La idea del mismo es construir una solución válida que luego se pueda mejorar en la fase siguiente con la heurística de Clarke y Wright. El algoritmo 3.12 detalla los pasos efectuados en la fase constructiva.

Inicialmente, al igual que lo efectuado en la heurística clásica de Clarke y Wright, cada cliente es asignado a una ruta unitaria. El depósito elegido para radicar cada ruta es aquél que tiene una capacidad máxima compatible con la demanda del cliente y que se encuentra a mínima distancia del mismo. En este paso pueden quedar clientes sin clusterizar. Los clientes no clusterizados continúan en ese estado por el resto de la fase constructiva.

Luego, se recorren todos los ejes activos que conectan clientes con depósitos. Se seleccionan los que tienen valor 1 en el LP y se utiliza dicha información para reclusterizar (asignar cada ruta/cliente a un nuevo depósito). Al finalizar, puede ocurrir que alguno de los depósitos se encuentre con capacidad excedida. Si es así, se arreglan los clientes que sean necesarios hasta que las capacidades de los depósitos sean adecuadas. Para ello, se relocalizan algunos clientes a clusters con capacidad disponible.

El paso siguiente es ordenar las variables x (que representan ejes que interconectan clientes entre sí) en orden decreciente en función de los valores que las mismas tienen en el LP en curso. En caso de haber variables con el mismo valor, se desempata dicha igualdad (para determinar el orden de las mismas) utilizando una variable aleatoria basada en una distribución uniforme $[0, 1]$. De esta manera, se busca

favorecer primero las variables que representan conexiones más fuertes en el LP (variables con valores lineales más cercanos a 1).

Se recorre luego la lista ordenada de ejes. Se analizan sólo los ejes cuyas variables superen el valor $1 * r$, siendo r la semilla generada al principio de la iteración. Para cada eje, si los clientes conectados por el mismo forman parte del mismo cluster, se intenta combinar las rutas a las que los mismos pertenecen. Para ello, se utiliza el proceso de combinación o merge definido en la heurística de Clark y Wright. Dadas las rutas $r_1 = r_{10}, ..., r_{1k}$, y $r_2 = r_{20}, ..., r_{2l}$, se analizan las 4 posibles combinaciones $r*_1 = r_{10}, ..., r_{1k}, r_{20}, ..., r_{2l}$, $r*_2 = r_{20}, ..., r_{2l}, r_{10}, ..., r_{1k}$, $r*_3 = r_{1k}, ..., r_{10}, r_{20}, ..., r_{2l}$, $r*_4 = r_{10}, ..., r_{1k}, r_{2l}, ..., r_{20}$. Se selecciona la combinación más económica (*mejorCombinacion*). Si la combinación es válida (las rutas combinadas no superan la capacidad máxima del vehículo) se actualizan las mismas disminuyendo en 1 la cantidad total de rutas.

Al finalizar la fase constructiva se cuenta con una solución válida para el problema *PC-CLRP*.

Algoritmo 3.12 construccionC&W

Input: x^* solución relajación LP
Output: solución factible

```
1: inicializarC&W
2: for each x_ij   i ∈ I   j ∈ J   / val(x_ij) >= 1 do
3:    clusterizar(j, i)
4: end for
5: arreglarCapacidadesExcedidas
6: l ← ∅
7: for each x_ij   i, j ∈ J   / val(x_ij) >= 1 * r do
8:    Apilar(x_ij, l)
9: end for
10: ordenarDecreciente(l) usando valores de x y variable aleatoria uniforme [0, 1] en caso de empates
11: for each i = 1 ... longitud(l) do
12:    if cluster(head(l(i)) = cluster(tail(l(i))) then
13:       b ← mejorCombinacion(head(l(i)), tail(l(i)))
14:       if combinacionValida(b) then
15:          combinarRutas(b)
16:       end if
17:    end if
18: end for
```

Fase mejoramiento C&W

La etapa de mejoramiento utiliza la heurística de Clark y Wright para tratar de combinar todas las rutas posibles. Se itera hasta que no se puedan hacer más combinaciones. En cada iteración se computan todas las combinaciones posibles válidas entre pares de rutas. Se elige la combinación que produce el mayor ahorro (disminución del costo total de la solución). Se aplica la combinación elegida utilizando la misma lógica presentada para la etapa de inicialización. En el algoritmo 3.13 pueden observarse los pasos enunciados.

Algoritmo 3.13 mejoramientoC&W

Input: s solución factible
Output: solución mejorada

```
1:  hayCombinacion ← TRUE
2:  while hayCombinacion do
3:    mayorAhorro ← 0
4:    hayCombinacion ← FALSE
5:    for each i, j ∈ {1 ... totalRutas}   i ≠ j do
6:      b ← mejorCombinacion(ruta(i), ruta(j))
7:      if combinacionValida(b) and mayorAhorro > costoCombinacion(b) then
8:        mayorAhorro ← costoCombinacion(b)
9:        mejorCombinacion ← b
10:       hayCombinacion ← TRUE
11:     end if
12:   end for
13:   if hayCombinacion then
14:     combinarRutas(mejorCombinacion)
15:   end if
16: end while
```

Fase mejoramiento por búsqueda local

Esta fase aplica los operadores de búsqueda local diseñados para la metaheurística presentada en el capítulo 2. Los operadores son aplicados utilizando diferentes estrategias dependiendo del número de nodo del árbol *Branch&Cut*.

Sea n el número de nodo del árbol, se aplican 3 estrategias diferentes:

- Si n mod 3 = 0: se aplica primero el procedimiento $fastLocalSearch$ presentado en 2.8 y luego el procedimiento completo $localSearch$ limitando el operador $CrossExchange$ a cadenas de 3 nodos.
- Si n mod 3 = 1: se aplica el procedimiento completo $localSearch$ limitando el operador $CrossExchange$ a cadenas de 4 nodos.
- Si n mod 3 = 2: se aplica primero un procedimiento $IntraDepots$ aplicando sólo los operadores de búsqueda local entre depósitos (*intra-depots Relocate*, *intra-depots Exchange* y *Route-Change*). Luego, se aplica el procedimiento $fastLocalSearch$ y, por último, el procedimiento completo $localSearch$ desactivando el operador $CrossExchange$.

Se testeó la alternativa propuesta frente a numerosas opciones siendo la combinación presentada la que mejores resultados otorgó.

3.10. Resultados Computacionales

El algoritmo *Branch&Cut* fue implementado en C++ usando el optimizador comercial CPLEX 12.6.1 [54] y el compilador GNU C++ versión 4.6.3. Los tests fueron corridos en una máquina con procesador Intel Core i7-3770, 3.40 Ghz con 16GB de memoria bajo el sistema operativo Linux kernel 3.8.

Se han deshabilitado todas las familias de cortes disponibles en CPLEX para poder analizar correctamente el impacto de los cortes introducidos y las distintas estrategias implementadas por el algoritmo *Branch&Cut*. A modo de prueba, y sólo para verificar si se podía llegar a mejorar las cotas de las instancias de mayor tamaño, se intentó activar algunos de los cortes de CPLEX. Sin embargo, la mayoría de los cortes no provocó mejoras en el desempeño global del algoritmo. Los únicos cortes que intermitentemente provocaron algunas mejoras fueron los cortes Lift and Project y Implied Bounds. Los mismos fueron desactivados finalmente para todas las corridas reportadas.

Como se mencionó en el capítulo anterior, en la revisión bibliográfica efectuada, no se han encontrado referencias al problema *PC-CLRP* tal cual se lo presenta en este trabajo. Debido a ello, no se han podido conseguir instancias específicamente diseñadas para este problema. Para poder evaluar el algoritmo propuesto, se utiliza un conjunto de 30 instancias diseñadas para el problema general *CLRP* denominadas en la literatura como instancias de Prodhon [86]. Las mismas pueden descargarse del sitio http://prodhonc.free.fr/Instances/instances_us.htm y se encuentran detalladas en el apéndice A.1. Utilizando el conjunto de instancias de Prodhon se generó un nuevo conjunto de instancias diseñado específicamente para el problema *PC-CLRP*. El algoritmo utilizado para la generación de las nuevas instancias se encuentra detallado en el apéndice A.2. Estas instancias pueden descargarse del sitio http://www.dc.uba.ar/People/negrotto.

Antes de pasar a mostrar los resultados generales del algoritmo, se reportarán parte de los experimentos efectuados para diseñar cada una de las secciones del algoritmo *Branch&Cut*. Para llevar a cabo estos experimentos se ha utilizado el conjunto de 8 instancias de pequeño tamaño (instancias con 20 clientes). Estas instancias forman parte del conjunto de nuevas instancias generadas a partir de las instancias de Prodhon.

Se han realizado también experimentos y corridas sobre el problema general *CLRP* para analizar el desenvolvimiento del algoritmo aplicado a dicho problema. Esto permite comparar el algoritmo presentado

con los resultados de los trabajos de Belenguer y otros [16] y Contardo y otros [21]. Para poder utilizar el algoritmo presentado en el problema general *CLRP* se han fijado las ganancias de los clientes a valores lo suficientemente altos como para que los mismos sean siempre activados en las soluciones analizadas. Además, para que el algoritmo sea competitivo, se ha activado la separación de las familias de desigualdades Multistar, Framed Capacity y Comb (ver Lysgaard y otros [67]). Estas desigualdades son separadas con las rutinas de separación CVRPSEP de Lysgaard [66]. Se han desactivado los cortes de optimalidad de clientes en equilibrio (3.33), activaciones implicadas 1 (3.34) y activaciones implicadas 2 (3.35) porque utilizan información propia del problema *PC-CLRP*. Por último, se ha agregado una restricción adicional a la relajación inicial que permite aumentar significativamente la cota inferior *LB* en el nodo raíz del árbol. Esta desigualdad pide que la capacidad de los depósitos abiertos supere la suma de las demandas de los clientes:

$$\sum_{i \in I} W_i z_i \geq \sum_{j \in J} d_j \tag{3.46}$$

El primero de los experimentos que se presenta a continuación tiene como objetivo evaluar las distancias existentes entre las cotas inferiores *LB* y las cotas superiores *UB* en el nodo raíz del árbol de enumeración *Branch&Cut*. Luego, se analiza la eficiencia de las distintas familias de cortes presentadas. Posteriormente, se comparan varias estrategias de separación seleccionadas y, por último, se realiza una comparación entre diferentes estrategias de branching.

Resultados en algoritmo de planos de corte en el nodo raíz

Para poder analizar la efectividad de los cortes y de la formulación presentada, es importante observar el estado de las cotas inferiores *LB* y superiores *UB* del problema al finalizar la etapa de planos de corte en el nodo raíz del árbol de enumeración. La distancia o gap existente entre ambas cotas tendrá un impacto directo en el tamaño del árbol. Dada la naturaleza del problema presentado, se espera de antemano una mayor variabilidad y distancia entre las cotas, en comparación con el problema *CLRP*. Esto se debe a que el problema *PC-CLRP* utiliza una función multiobjetivo, al igual que el problema *CLRP*, pero con el agregado de un término cuya suma se quiere maximizar. Además, este término se basa en los beneficios que otorgan los clientes y los mismos pueden estar activos o no. Esto aumenta aún más la variabilidad de la función objetivo. En consecuencia, se espera que los gaps sean superiores a los observados en el problema *CLRP*.

A continuación se presentan los resultados obtenidos en el nodo raíz para el problema *PC-CLRP* sobre el conjunto de instancias de 20 clientes. Luego, se presentan también los resultados del algoritmo aplicado a las instancias de 20 clientes del problema *CLRP*. Las cotas superiores *UB* han sido pasadas como parámetros del algoritmo utilizando los mejores resultados obtenidos con la metaheurística presentada en el capítulo 2.

En todos los casos se reporta la información de la solución obtenida presentando la cota inferior *LB*, la cota superior pasada *UB*, el gap a la cota superior para la instancia (calculado como $gap = 100 * (LB - UB)/UB$) y el tiempo utilizado en segundos. El cuadro 3.1 muestra los resultados obtenidos para cada una de las instancias de 20 clientes.

Instancia	UB	LB	Gap %	Tiempo (s)
pc20-5-1-s	-6435*	-11933.57	85.45	0.24
pc20-5-1-l	-33516*	-39824.31	18.82	0.80
pc20-5-1b-s	-768*	-2854.58	271.69	0.38
pc20-5-1b-l	-56122*	-58528.17	4.29	0.31
pc20-5-2-s	-2971*	-7030.85	136.65	1.55
pc20-5-2-l	-44725*	-48929.84	9.40	1.87
pc20-5-2b-s	-114*	-1060.25	830.04	0.55
pc20-5-2b-l	-59744*	-60928.72	1.98	0.12

Cuadro 3.1: Resultados nodo raíz

Una primera observación que puede hacerse es que las instancias generadas para tener soluciones óptimas con un número reducido de clientes activos (instancias terminadas en -s) tienen un gap muchísimo mayor que las generadas para tener un número grande de clientes atendidos. Esto puede deberse a la variabilidad que se mencionaba anteriormente y la influencia de las ganancias en la función objetivo. Para poder analizar esta situación se ha decidido analizar la composición de cada una de las cotas inferiores *LB*, separándolas en los costos de rutas, costos de depósitos, costos de vehículos y las ganancias por clientes atendidos. Se informa en el cuadro 3.2 los valores de cada uno de los costos comparando la diferencia o gap entre el costo respectivo en la cota inferior y la cota superior. En el cuadro 3.3 se informan los costos totales y las ganancias. En las últimas 2 columnas se informa la cota inferior obtenida *LB* y el gap a la cota superior *UB*.

Instancia	UB	Costo Ruteo			Costo Depósitos			Costo Vehículos		
		UB	LB Root	Gap %	UB	LB Root	Gap %	UB	LB Root	Gap %
pc20-5-1-s	-6435	11074	14810,78	33,74	6091	12922.22	112.15	2000	3259.46	62.97
pc20-5-1-l	-33516	25596	16118,62	-37,03	13588	14098.96	3.76	4000	3565.63	-10.86
pc20-5-1b-s	-768	10937	15677,8	43,35	6995	7157.51	2.32	1000	1672.11	67.21
pc20-5-1b-l	-56122	21183	18491,2	-12,71	6995	7188.91	2.77	2000	1946.38	-2.68
pc20-5-2-s	-2971	12813	12980,33	1,31	6616	10218.84	54.46	3000	2833.73	-5.54
pc20-5-2-l	-44725	17655	14984,17	-15,13	14620	12316.69	-15.75	4000	3312.01	-17.20
pc20-5-2b-s	-114	18718	16770,38	-10,41	8068	7670.61	-4.93	2000	1759.53	-12.02
pc20-5-2b-l	-59744	19388	18028,88	-7,01	8068	8090.15	0.27	2000	1901.94	-4.90

Cuadro 3.2: Resultados con participación de distintos costos en la cota inferior

Instancia	UB	Costo total				Ganancia			Total	
		UB	LB Root	Dif	Gap %	UB	LB Root	Gap %	LB	Gap %
pc20-5-1-s	-6435	19165.00	30992.46	11827.46	61.71	25600	42926.03	67.68	-11933.57	85.45
pc20-5-1-l	-33516	43184.00	33783.21	-9400.79	-21.77	76700	73607.51	-4.03	-39824.3	18.82
pc20-5-1b-s	-768	18932.00	24507.42	5575.42	29.45	19700	27362.00	38.89	-2854.58	271.69
pc20-5-1b-l	-56122	30178.00	27626.49	-2551.51	-8.45	86300	86154.66	-0.17	-58528.17	4.29
pc20-5-2-s	-2971	22429.00	26032.90	3603.90	16.07	25400	33063.74	30.17	-7030.84	136.65
pc20-5-2-l	-44725	36275.00	30612.87	-5662.13	-15.61	81000	79542.72	-1.80	-48929.85	9.40
pc20-5-2b-s	-114	28786.00	26200.52	-2585.48	-8.98	28900	27260.77	-5.67	-1060.25	830.04
pc20-5-2b-l	-59744	29456.00	28020.97	-1435.03	-4.87	89200	88949.72	-0.28	-60928.75	1.98

Cuadro 3.3: Resultados con participación de ganancias en la cota inferior

Según puede observarse en los cuadros anteriores, tanto los costos como las ganancias están sobre-estimados en las instancias con pocos clientes atendidos. El modelo propuesto tiende a activar muchos más clientes que los que finalmente se utilizan en la solución óptima para este tipo de instancias. Al activarse clientes en exceso se sobre-estiman las ganancias, y por consiguiente, también los costos ya que los mismos están íntimamente relacionados a la infraestructura necesaria para satisfacer los requerimientos de los clientes. Una línea de investigación futura podría focalizarse en intentar mejorar el modelo presentado, poniendo especial énfasis en ajustar la selección inicial de clientes activos.

Sin embargo, según puede observarse en el cuadro 3.4, al aplicarse un branching reducido que enumera sólo los primeros 50 nodos, se obtienen rápidamente cotas mucho más ajustadas y cercanas. Incluso, la instancia que tenía el mayor gap encuentra su valor óptimo en el nodo 19. En el cuadro también se muestran los resultados de aplicar el algoritmo *Branch&Cut* completo para mostrar el esfuerzo necesario para llegar a los valores óptimos de cada instancia.

Instancia	UB	B&C Parcial (50 nodos)				B&C Completo		
		LB	Gap %	Tiempo (s)	Nodos	Gap %	Tiempo (s)	Nodos
pc20-5-1-s	-6435*	-7488.94	16.38	2.09	50	0.00	6.38	261
pc20-5-1-l	-33516*	-35465.63	5.82	3.76	50	0.00	35.09	613
pc20-5-1b-s	-768*	-1355.50	76.50	2.83	50	0.00	2.98	77
pc20-5-1b-l	-56122*	-56122.00	0	0.69	15	0.00	0.70	15
pc20-5-2-s	-2971*	-4035.23	35.82	3.31	50	0.00	6.49	242
pc20-5-2-l	-44725*	-46305.41	3.53	3.88	50	0.00	12.36	400
pc20-5-2b-s	-114*	-114.00	0	1.75	19	0.00	1.77	19
pc20-5-2b-l	-59744*	-59744.00	0	0.30	15	0.00	0.30	15

Cuadro 3.4: Resultados Branch&Cut parcial y completo

Por lo tanto, si bien en algunas instancias los gaps son muy grandes, esto se logra reducir luego de algunas iteraciones del algoritmo *Branch&Cut*. De acuerdo a la estrategia de branching implementada, primero se efectúa branching sobre las variables fraccionales z (variables que representan a los depósitos activos). Según pudo observarse en las instancias anteriores, los gaps se reducen principalmente en los primeros nodos del árbol de enumeración y fundamentalmente al efectuar branching sobre las variables z.

Para corroborar lo que se enunciaba al inicio de esta sección, se muestran a continuación los resultados de aplicar el algoritmo *Branch&Cut* sobre las instancias generales *CLRP* de 20 clientes. Se informan los resultados obtenidos en el nodo raíz y los obtenidos al aplicar el algoritmo *Branch&Cut* completo. Como puede observarse en el cuadro 3.5, las cotas en las instancias generales son mucho más ajustadas. Esto muestra que el impacto de tener que seleccionar el conjunto de clientes a activar y su correspondiente suma de ganancias en la función objetivo, genera una variabilidad muy grande que provoca que las cotas inferiores en el nodo raíz se encuentren a una distancia importante de las cotas superiores.

Instancia	UB	Nodo raíz			B&C Completo		
		LB	Gap %	Tiempo (s)	Gap %	Tiempo (s)	Nodos
coord20-5-1	54793*	50504.84	7.82	0.17	0.00	7.02	167
coord20-5-1b	39104*	39104.00	0.00	0.08	0.00	0.08	0
coord20-5-2	48908*	43891.35	10.41	0.14	0.00	2.44	98
coord20-5-2b	37542*	37542.00	0.00	0.05	0.00	0.05	0

Cuadro 3.5: Resultados nodo raíz y Branch&Cut completo sobre CLRP

Eficiencia de las familias de cortes

En esta sección se analizarán los cortes utilizados para la resolución de cada una de las instancias de 20 clientes y la eficiencia relativa de cada una de las familias de desigualdades utilizadas. En el cuadro 3.6 se muestra la cantidad de cortes separados de cada una de las familias durante la ejecución del algoritmo *Branch&Cut* completo. Las mismas son y-Cap (y-Capacidad 3.20), PC-Cap (PC-Capacidad 3.18 + PC-Subtour 3.19), DD (Depot-Degree o Facility-Degree 3.31), 3Path (3-Path 3.21), CoC (Co-Circuit 3.22), CliEq (clientes en equilibrio 3.33), LogR (lógica de ruta 3.12), w-z (desigualdades w-z 3.17), PC-Hyp (PC-2E-Hypotour 3.26), PC-GLM (PC-GLM 3.24) y Ac-Imp (Activaciones Implicadas 1 3.34 y 2 3.35).

Instancia	y-Cap	PC-Cap	DD	3Path	CoC	CliEq	LogR	w-z	PC-Hyp	PC-GLM	Ac-Imp
pc20-5-1-s	415	9	10	43	20	22	184	60	163	52	1
pc20-5-1-l	1063	28	9	10	21	21	494	67	265	98	1
pc20-5-1b-s	247	30	28	22	19	23	60	70	452	19	0
pc20-5-1b-l	84	26	33	7	13	0	16	53	268	10	0
pc20-5-2-s	408	73	22	90	40	14	145	37	149	45	0
pc20-5-2-l	677	33	3	14	17	8	236	46	118	40	1
pc20-5-2b-s	95	19	16	10	11	21	12	35	630	7	0
pc20-5-2b-l	51	16	8	2	4	4	23	31	82	8	0
Total	3040	234	129	198	145	113	1170	399	2127	279	3
%	38,79	2,99	1,65	2,53	1,85	1,44	14,93	5,09	27,14	3,56	0,04

Cuadro 3.6: Cantidad de desigualdades encontradas al aplicar el algoritmo Branch&Cut completo

A simple vista puede observarse el predominio de los cortes y-Capacidad. Los mismos representan la mayor parte de los cortes separados, contabilizando para estas instancias el 38,79 % del total de desigualdades separadas en el algoritmo *Branch&Cut*. Le siguen en cantidad las desigualdades PC-2E-Hypotour y las restricciones de lógica de ruta. Puede observarse también que se han identificado muy pocos cortes de optimalidad de Activaciones Implicadas y se ha comprobado que las mismas no han sido muy efectivas en el

conjunto de instancias de 20 clientes. Se supone que esto se debe a la estructura que presentan las instancias creadas. Las ganancias han sido generadas aleatoriamente en el rango de los costos involucrados, y esto provoca que este tipo de cortes de optimalidad no sea demasiado útil. Se supone que si se usaran instancias donde las ganancias superaran ampliamente a los costos, las mismas deberían ser más efectivas. Lo opuesto es probable que ocurra con los cortes de clientes en equilibrio. En este caso las ganancias deberían ser muy inferiores a los costos para que los cortes tomen un papel más preponderante. En el cuadro anterior puede verse que se han identificado 113 cortes de este tipo para todas las instancias de 20 clientes. Sin embargo, habría que analizar el impacto de las mismas en el algoritmo *Branch&Cut*.

Para poder estudiar la incidencia de cada una de las familias de desigualdades presentadas, se muestra a continuación un análisis efectuado sobre el nodo raíz del árbol *Branch&Cut* (se utiliza sólo el algoritmo de planos de corte). El algoritmo de planos de corte se aplica a la relajación inicial (presentada en la sección 3.9.1) del modelo junto a los cortes necesarios para eliminar soluciones enteras no factibles. Se ha definido una versión *base* que contiene las familias de desigualdades necesarias para eliminar tales soluciones: desigualdades y-Capacidad (3.20) y restricciones de lógica de ruta (3.12). Se podrían haber utilizado las desigualdades PC-Capacidad (3.18) en lugar de las desigualdades y-Capacidad, sin embargo estas últimas son más efectivas y logran obtener una cota inferior muchas más ajustada.

Se han definido 8 versiones adicionales a la versión *base* para estudiar las distintas familias de cortes y su eficacia en el problema estudiado:

I. *base*+PC-Capacidad

II. *base*+Facility-Degree

III. *base*+Restricciones de clientes en equilibrio

IV. *base*+PC-2E-Hypotour

V. *base*+PC-GLM

VI. *base*+Co-Circuit

VII. *base*+Restricciones w-z

VIII. *base*+3-Path

La estrategia de separación utilizada en las pruebas sigue los lineamientos presentados en la sección 3.9.4. Se deshabilitan todos los cortes no involucrados en el experimento. En el cuadro 3.7 y 3.8 puede

observarse los resultados de las distintas corridas. Además de reportarse los gaps a las cotas superiores se ha definido el *gap cerrado* como $Gap\ C = 100 * (LB_1 - LB_0)/(UB - LB_0)$, donde LB_0 es la cota inferior de la versión *base* y LB_1 es la cota inferior de la versión estudiada. El objetivo de este nueva métrica es analizar la incidencia o aporte que proporciona cada una de las familias de cortes estudiadas.

Instancia	Base Gap	Base+PC-Cap		Base+DD		Base+CliEq		Base+PC-Hyp	
		Gap	Gap C	Gap	Gap C	Gap	Gap C	Gap	Gap C
pc20-5-1-s	172,94	101,86	41,10	172,94	0,00	172,94	0,00	152.86	11.61
pc20-5-1-l	32,21	20,90	35,14	32,21	0,00	32,21	0,00	30.99	3.81
pc20-5-1b-s	864,68	561,71	35,04	864,68	0,00	864,68	0,00	671.92	22.29
pc20-5-1b-l	11,65	6,99	40,01	11,65	0,00	11,64	0,05	8.45	27.46
pc20-5-2-s	313,39	165,23	47,27	313,39	0,00	313,39	0,00	271.34	13.42
pc20-5-2-l	19,48	11,74	39,72	19,48	0,00	19,48	0,00	17.21	11.65
pc20-5-2b-s	4585,30	2109,98	53,98	4585,30	0,00	4585,30	0,00	3338.26	27.20
pc20-5-2b-l	7,21	3,04	57,89	7,21	0,00	7,21	0,00	3.04	57.89
prom	750,86	372,68	43,77	750,86	0,00	750,86	0,01	561.76	21.92

Cuadro 3.7: Eficiencia en versiones I, II, III y IV

Instancia	Base Gap	Base+GLM		Base+CoC		Base+w-z		Base+3Path	
		Gap	Gap C	Gap	Gap C	Gap	Gap C	Gap	Gap C
pc20-5-1-s	172,94	95.81	44.60	161.21	6.78	172.61	0.19	164.07	5.13
pc20-5-1-l	32,21	20.62	35.98	31.68	1.66	32.22	-0.01	32.04	0.54
pc20-5-1b-s	864,68	554.27	35.90	821.32	5.01	776.45	10.20	812.55	6.03
pc20-5-1b-l	11,65	6.66	42.81	10.60	8.99	9.91	14.90	10.01	14.08
pc20-5-2-s	313,39	182.27	41.84	269.84	13.90	321.40	-2.56	292.48	6.67
pc20-5-2-l	19,48	10.82	44.42	17.59	9.70	18.81	3.43	17.63	9.48
pc20-5-2b-s	4585,30	2801.77	38.90	4093.98	10.72	3868.43	15.63	4302.26	6.17
pc20-5-2b-l	7,21	2.56	64.57	7.24	-0.41	5.92	17.89	6.23	13.63
promedio	750,86	459.35	43.63	676.68	7.04	650.72	7.46	704.66	7.72

Cuadro 3.8: Eficiencia en versiones V, VI. VII y VIII

Las desigualdades PC-Capacidad y las desigualdades PC-GLM son las familias de desigualdades que más reducen el gap en el algoritmo de planos de corte. Le siguen en importancia los cortes PC-2E-Hypotour que también logran reducir los gaps notablemente. Sin embargo, la reducción es despareja dependiendo de la instancia analizada. El resto de las familias producen reducciones moderadas variables de acuerdo a la instancia analizada. En los cuadros anteriores puede observarse también que las familias de cortes Facility-Degree 3.31 y clientes en equilibrio (3.33) no generan ninguna reducción en los gaps (una sola instancia reduce mínimamente el gap para los cortes de clientes en equilibrio). Sin embargo, en todas las instancias se identifican algunas desigualdades. Coincidiendo con el análisis efectuado anteriormente para los cortes de optimalidad de Implicaciones Activadas, esto puede deberse a la naturaleza de las instancias generadas y puede ser diferente en otro tipo de instancias que contenga ganancias con valores que superen ampliamente a los costos.

Comparación de estrategias de separación

Se pretende ahora mostrar algunas de las comparaciones efectuadas para determinar la mejor estrategia de separación. Como se mencionó anteriormente, se utilizó como herramienta auxiliar el software *IRACE*. Se programaron distintos procesos batch con configuraciones múltiples y se ejecutó *IRACE* para que eligiera la configuración que mejores resultados ofrecía. Se utilizó el subconjunto de 8 instancias de 20 nodos y un subconjunto reducido de instancias de mayor tamaño para que *IRACE* efectuara la selección. El parámetro estipulado para optimizar fue el tiempo total que el algoritmo utiliza para resolver cada instancia. La cantidad de alternativas analizadas supera las 200 estrategias si se considera las estrategias analizadas en forma separada para el nodo raíz y el resto de los nodos de *Branch&Cut*. Las estrategias probaron distintas maneras de llamar a las diferentes rutinas de separación, habilitando y deshabilitando rutinas como así también variando el orden de ejecución de las mismas. Además, gracias al esquema de parametrización implementado, se pudieron testear diferentes alternativas de interrupción y reiniciado del proceso de separación considerando los distintos operadores definidos en la sección 3.9.4 para, en caso de encontrar cortes violados, salir de la rutina de separación. En esta sección se muestran solamente 4 de las muchas estrategias probadas. Las estrategias mostradas forman parte del conjunto de estrategias preseleccionadas antes de concretar la elección de la estrategia definitiva.

Las 4 estrategias de separación utilizadas en la comparación se definen de la siguiente manera:

I. Versión final implementada presentada en la sección 3.9.4.
   Nodo raíz: Z-L-C-3*W-Y-D-B-H-G* Resto Nodos: Z-L-Y*C2B-3*G-H2

II. Versión con estrategia unificada para todos los nodos. Todos los nodos: Z-L-3-Y*W*D-B-C-H2G*

III. Nodo raíz: ZsLsY*W*DsBsCs3sGsHs Resto Nodos: LsZs3sY*W*BsDsG*H*Cs

IV. Nodo raíz: Z-L-Y13-D-B-C2W1H1G* Resto Nodos: 3-B-Z-LsY-W*DsG2H2C2

Se ha utilizado la misma nomenclatura utilizada en la parametrización del algoritmo implementado. La definición de cada una de las letras anteriores para las rutinas de separación es la siguiente:

- Z: desigualdades w-z (3.17) y activaciones implicadas 1 (3.34) y 2 (3.35).
- L: restricciones de lógica de ruta (3.12).
- C: desigualdades de Co-Circuit (3.22).

- 3: desigualdades 3-Path (3.21)
- W: desigualdades PC-Capacidad (3.18) y PC-Subtour (3.19).
- Y: desigualdades y-Capacidad (3.20).
- D: cortes de optimalidad Facility-Degree (3.31).
- B: cortes de optimalidad de clientes en equilibrio (3.33).
- H: desigualdades PC-2E-Hypotour (3.26).
- G: desigualdades PC-GLM (3.24).

La nomenclatura de los saltos de reoptimización es la siguiente:

- -: no reoptimizar y continuar con la siguiente rutina de separación.
- *: si se encontró algún corte reoptimizar.
- 1: si se encontró algún corte con violación superior a 0.1 reoptimizar.
- 2: si se encontró algún corte con violación superior a 0.2 reoptimizar.
- s: si se identificaron más de 100 cortes en la separación de la última familia o en las previas, reoptimizar.

Instancia	B&C estrategia 1			B&C estrategia 2			B&C estrategia 3			B&C estrategia 4		
	Gap	Tiempo	Nodos	Gap	Tiempo	Nodos	Gap	Tiempo	Nodos	Gap	Tiempo	Nodos
pc20-5-1-s	0.00	6.38	261	0.00	8.57	240	0.00	8.13	245	0.00	9.20	269
pc20-5-1-l	0.00	35.09	613	0.00	50.51	636	0.00	40.99	625	0.00	41.79	661
pc20-5-1b-s	0.00	2.98	77	0.00	4.10	81	0.00	5.18	97	0.00	2.47	55
pc20-5-1b-l	0.00	0.70	15	0.00	0.59	13	0.00	0.52	15	0.00	0.48	17
pc20-5-2-s	0.00	6.49	242	0.00	8.54	250	0.00	7.23	243	0.00	7.37	255
pc20-5-2-l	0.00	12.36	400	0.00	11.56	379	0.00	12.35	391	0.00	14.10	371
pc20-5-2b-s	0.00	1.77	19	0.00	3.56	19	0.00	4.97	21	0.00	3.92	13
pc20-5-2b-l	0.00	0.30	15	0.00	0.40	21	0.00	0.32	19	0.00	0.67	15
promedio		8.25	205.25		10.98	204.88		9.96	207		10.00	207

Cuadro 3.9: Resultados comparación estrategias de separación

No se incluyeron en la comparación las estrategias que deshabilitaban cortes o incorporaban las restricciones a la relajación inicial (para las desigualdades w-z, restricciones de lógicas de ruta, cortes de clientes en equilibrio o activaciones implicadas 1 y 2). Como puede observarse en la tabla 3.9 las diferencias entre las distintas estrategias presentadas no son significativas para el conjunto de instancias de 20 clientes. Algunas estrategias tienen mejores resultados en algunas instancias y peores en otras. Para poder elegir la mejor

opción fue necesario incluir también un conjunto reducido del resto de las instancias de mayor tamaño y analizar detenidamente las distintas cotas LB de los nodos raíz. Las mayores diferencias se notan en las estrategias que deshabilitan cortes o en las que efectúan demasiados saltos de reoptimización.

Reglas de branching comparadas

Para elegir la estrategia de branching se han testeado numerosas alternativas. La estrategia finalmente utilizada es la que minimiza el tamaño del árbol de enumeración *Branch&Cut* en la mayor parte de las instancias. Se muestran en esta sección algunas de las estrategias probadas.

Las 4 estrategias de branching utilizadas en la comparación se definen de la siguiente manera:

I. Branching jerárquico presentado en la sección 3.9.5. Parámetro $branch23 = 0{,}45$ (estrategia elegida).

II. Branching jerárquico presentado en la sección 3.9.5. Parámetro $branch23 = 0{,}00$.

III. Branching de variables de CPLEX utilizando Strong Branching y orden de variables: z, x, w, y.

IV. Branching de variables default de CPLEX.

Todas las estrategias testeadas han sido corridas con un tiempo máximo de 10 minutos.

Instancia	B&C estrategia 1			B&C estrategia 2			B&C estrategia 3			B&C estrategia 4		
	Gap	Tiempo	Nodos	Gap	Tiempo	Nodos	Gap	Tiempo	Nodos	Gap	Tiempo	Nodos
pc20-5-1-s	0.00	6.38	261	0.00	6.92	279	0.00	9.19	304	0.00	34.12	3609
pc20-5-1-l	0.00	35.09	613	0.00	35.51	620	0.00	94.79	1204	2.56	600.00	13278
pc20-5-1b-s	0.00	2.98	77	0.00	5.82	141	0.00	5.74	141	0.00	47.72	1493
pc20-5-1b-l	0.00	0.70	15	0.00	0.70	15	0.00	0.77	18	0.00	13.99	365
pc20-5-2-s	0.00	6.49	242	0.00	6.99	269	0.00	9.75	357	0.00	21.96	2707
pc20-5-2-l	0.00	12.36	400	0.00	13.45	401	0.00	28.76	698	0.00	267.43	22640
pc20-5-2b-s	0.00	1.77	19	0.00	1.35	27	0.00	2.19	17	0.00	2.27	39
pc20-5-2b-l	0.00	0.30	15	0.00	0.28	15	0.00	0.53	16	0.00	0.97	34
promedio		8.25	205.25		8.88	220.87		18.96	344.37		123.56	5520.62

Cuadro 3.10: Resultados comparación estrategias de branching

Como puede observarse, la estrategia elegida (estrategia 1) obtiene los mejores tiempos y la menor cantidad de nodos en el árbol de enumeración *Branch&Cut*. La diferencia con los branching propuestos por CPLEX es significativa.

Resultados para el CLRP

Como se mencionó al comienzo de esta sección, se ha ejecutado el presente algoritmo *Branch&Cut* con las instancias generales del problema *CLRP*. El objetivo es poder comparar el algoritmo presentado con los resultados de los trabajos de Belenguer y otros [16] y Contardo y otros [21] y tener referencias del

desenvolvimiento del mismo. Para ello, se han activado algunas rutinas de separación y desactivado otras tal como fue explicado al inicio de la sección.

Si bien existen otros métodos exactos publicados, como los trabajos de Baldacci y otros [13] y Contardo y otros [22], los mismos no se encuentran basados en algoritmos *Branch&Cut*. Se ha decidido utilizar como comparación los trabajos que utilizan algoritmos *Branch&Cut* como método principal, pero es importante destacar que estos dos trabajos obtienen las mejores cotas publicadas hasta el momento para las instancias de Prodhon analizadas.

Los resultados presentados en el cuadro 3.11 muestran, para cada una de las instancias, la mejor cota superior conocida (UB) obtenida del trabajo de Contardo y otros [22] (que contiene las mejores soluciones reportadas hasta el momento mediante algoritmos exactos), la cota inferior obtenida (LB), el gap de la cota inferior a la cota superior, el tiempo total en segundos utilizados por el algoritmo y la cantidad total de nodos del árbol de enumeración. Todas las instancias han sido corridas con un tiempo máximo de 2 horas (7200 segundos).

	UB	LB	Gap %	Tiempo (s)	Nodos
coord20-5-1	54793	54793.00	0	7.02	167
coord20-5-1b	39104	39104.00	0	0.08	0
coord20-5-2	48908	48908.00	0	2.44	98
coord20-5-2b	37542	37542.00	0	0.05	0
coord50-5-1	90111	88019.59	2.32	7200.00	2211
coord50-5-1b	63242	61211.53	3.21	7200.00	403
coord50-5-2	88298	84395.87	4.42	7200.00	2251
coord50-5-2b	67308	64840.26	3.67	7200.00	286
coord50-5-2BIS	84055	82694.83	1.62	7200.00	2874
coord50-5-2bBIS	51822	48641.72	6.14	7200.00	1154
coord50-5-3	86203	84704.52	1.74	7200.00	2363
coord50-5-3b	61830	60672.44	1.87	7200.00	450
coord100-5-1	274814	267538.43	2.65	7200.00	471
coord100-5-1b	213615	208278.19	2.50	7200.00	27
coord100-5-2	193671	190020.42	1.88	7200.00	466
coord100-5-2b	157095	154678.10	1.54	7200.00	15
coord100-5-3	200079	196002.16	2.04	7200.00	658
coord100-5-3b	152441	150541.20	1.25	7200.00	67
coord100-10-1	287983	275090.22	4.48	7200.00	144
coord100-10-1b	231763	225029.56	2.91	7200.00	67
coord100-10-2	243590	233114.87	4.30	7200.00	289
coord100-10-2b	203988	196258.84	3.79	7200.00	53
coord100-10-3	250882	238580.72	4.90	7200.00	176
coord100-10-3b	204317	194048.94	5.02	7200.00	7
coord200-10-1	477248	434111.07	9.03	7200.00	6
coord200-10-1b	378351	346099.77	8.52	7200.00	4
coord200-10-2	449571	436360.08	2.93	7200.00	20
coord200-10-2b	374330	347096.13	7.27	7200.00	3
coord200-10-3	469433	436077.91	7.10	7200.00	5
coord200-10-3b	362817	337404.41	7.00	7200.00	3

Cuadro 3.11: Resultados CLRP sobre las instancias Prodhon

Como se observa en el cuadro, todos los gaps obtenidos son inferiores a 10 %, aún en las instancias de mayor tamaño. En las instancias de 200 clientes se visualiza una drástica reducción del tamaño del árbol de enumeración (cantidad de nodos procesados).

En el cuadro 3.12 se comparan los gaps obtenidos en el presente algoritmo frente a los obtenidos en los trabajos de Belenguer y otros [16] y Contardo y otros [21]. Se informan sólo los gaps, ya que los distintos algoritmos han sido ejecutados sobre computadores de muy diferente configuración. En el trabajo de Belenguer y otros [16] se estipula un tiempo máximo de cómputo de 7200 segundos (2 horas) mientras que en el trabajo de Contardo y otros [21] se utiliza un tiempo máximo de 10800 segundos (3 horas). Los resultados publicados en [16] contienen sólo información sobre las instancias de 20 y 50 clientes. Por su parte, el trabajo [21] reporta los resultados de las instancias de 20, 50 y 100 clientes. El promedio reportado se calcula sobre las instancias informadas.

	UB	Gap Belenguer %	Gap Contardo %	Gap nuevo B&C %
coord20-5-1	54793	0.00	0.00	0.00
coord20-5-1b	39104	0.00	0.00	0.00
coord20-5-2	48908	0.00	0.00	0.00
coord20-5-2b	37542	0.00	0.00	0.00
coord50-5-1	90111	2.26	2.46	2.32
coord50-5-1b	63242	1.37	1.58	3.21
coord50-5-2	88298	1.11	1.56	4.42
coord50-5-2b	67308	1.38	1.19	3.67
coord50-5-2BIS	84055	0.47	0.35	1.62
coord50-5-2bBIS	51822	0.00	0.00	6.14
coord50-5-3	86203	1.80	1.58	1.74
coord50-5-3b	61830	0.00	0.00	1.87
coord100-5-1	274814		2.54	2.65
coord100-5-1b	213615		2.19	2.50
coord100-5-2	193671		1.78	1.88
coord100-5-2b	157095		0.86	1.54
coord100-5-3	200079		1.50	2.04
coord100-5-3b	152441		0.67	1.25
coord100-10-1	287983		3.54	4.48
coord100-10-1b	231763		1.40	2.91
coord100-10-2	243590		1.54	4.30
coord100-10-2b	203988		0.10	3.79
coord100-10-3	250882		3.46	4.90
coord100-10-3b	204317		2.47	5.02
promedio		0.70	1.28	2.59

Cuadro 3.12: Comparación resultados CLRP sobre las instancias Prodhon

Los resultados muestran que el algoritmo aquí presentado obtiene soluciones de similar calidad a las soluciones obtenidas por los trabajos comparados. Si bien los gaps reportados en la mayoría de las instancias es mayor, los mismos se encuentran cercanos a los obtenidos en los trabajos de referencia. Es importante destacar que los algoritmos *Branch&Cut* con los que se efectúa la comparación estan basados en modelos diferentes y utilizan además familias de desigualdades válidas adicionales diseñadas especialmente para *CLRP* o derivadas desde el problema *CVRP* que no han sido implementadas en el algoritmo presentado.

Resultados PC-CLRP

Se presentan ahora los resultados generales para el problema *PC-CLRP*. Se informan las cotas superiores *UB* obtenidas con la metaheurística del capítulo 2, la cota inferior *LB* en el nodo raíz, el gap de dicha cota a la cota superior y los resultados del *Branch&Cut* completo. Para estos últimos se informa la cota

inferior LB final, el gap de la cota inferior a la cota superior, el tiempo total de cómputo en segundos y la cantidad de nodos utilizados en el árbol de enumeración. El gap se calcula, al igual que antes, como $gap = 100 * (LB - UB)/UB$.

Instancia	UB	Nodo Raíz		B&C Completo			
		LB	Gap	LB	Gap	Tiempo (s)	Nodos
pc20-5-1-s	-6435*	-11933.57	85.45	-6435.00	0	6.38	261
pc20-5-1-l	-33516*	-39824.31	18.82	-33516.00	0	35.09	613
pc20-5-1b-s	-768*	-2854.58	271.69	-768.00	0	2.98	77
pc20-5-1b-l	-56122*	-58528.17	4.29	-56122.00	0	0.70	15
pc20-5-2-s	-2971*	-7030.85	136.65	-2971.00	0	6.49	242
pc20-5-2-l	-44725*	-48929.84	9.40	-44725.00	0	12.36	400
pc20-5-2b-s	-114*	-1060.25	830.04	-114.00	0	1.77	19
pc20-5-2b-l	-59744*	-60928.72	1.98	-59744.00	0	0.30	15
pc50-5-1-s	-1453	-13757.10	846.81	-4209.74	189.73	7200.00	4862
pc50-5-1-l	-38661	-52231.44	35.10	-42404.24	9.68	7200.00	2939
pc50-5-1b-s	-4639	-14455.47	211.61	-8771.48	89.08	7200.00	796
pc50-5-1b-l	-62173	-71207.02	14.53	-65890.15	5.98	7200.00	809
pc50-5-2-s	-4835	-12165.71	151.62	-5872.78	21.46	7200.00	7024
pc50-5-2-l	-43809	-52692.18	20.28	-46773.68	6.77	7200.00	4402
pc50-5-2b-s	-4521	-8932.51	97.58	-6964.72	54.05	7200.00	605
pc50-5-2b-l	-62867	-69149.54	9.99	-66900.55	6.42	7200.00	522
pc50-5-2BIS-s	-786	-3288.85	318.43	-881.00*	-	23.30	203
pc50-5-2BIS-l	-14212	-21686.03	52.59	-15429.29	8.57	7200.00	8873
pc50-5-2bBIS-s	-4383*	-6124.23	39.73	-4383.00	0.00	4542.00	3487
pc50-5-2bBIS-l	-34606*	-36335.83	5.00	-35105.18	1.44	7200.00	2236
pc50-5-3-s	-6050	-12597.49	108.22	-7101.23	17.38	7200.00	5732
pc50-5-3-l	-38504	-51091.09	32.69	-40724.13	5.77	7200.00	6334
pc50-5-3b-s	-21107	-28406.03	34.58	-22687.99	7.49	7200.00	644
pc50-5-3b-l	-60192	-69276.28	15.09	-63347.86	5.24	7200.00	453

Cuadro 3.13: Resultados Branch&Cut 20-50 clientes

Instancia	UB	Nodo Raíz		B&C Completo			
		LB	Gap	LB	Gap	Tiempo (s)	Nodos
pc100-5-1-s	-41917	-62212.14	48.42	-49465.10	18.01	7200.00	867
pc100-5-1-l	-230396	-258909.80	12.38	-239326.91	3.88	7200.00	499
pc100-5-1b-s	-28315	-43288.27	52.88	-34985.02	23.56	7200.00	161
pc100-5-1b-l	-145023	-167476.35	15.48	-154010.31	6.20	7200.00	109
pc100-5-2-s	-20861	-40058.56	92.03	-26270.84	25.93	7200.00	873
pc100-5-2-l	-141791	-172781.81	21.86	-147524.59	4.04	7200.00	398
pc100-5-2b-s	-51662	-64529.91	24.91	-56088.44	8.57	7200.00	196
pc100-5-2b-l	-177450	-200947.55	13.24	-181335.90	2.19	7200.00	123
pc100-5-3-s	-11394	-32049.70	181.29	-15877.57	39.35	7200.00	1154
pc100-5-3-l	-119756	-155173.10	29.57	-126910.13	5.97	7200.00	879
pc100-5-3b-s	-41503	-54651.05	31.68	-45855.59	10.49	7200.00	189
pc100-5-3b-l	-161475	-182697.69	13.14	-166977.57	3.41	7200.00	156
pc100-10-1-s	-7804	-13265.39	69.98	-10237.89	31.19	7200.00	1386
pc100-10-1-l	-111218	-125412.74	12.76	-119669.93	7.60	7200.00	434
pc100-10-1b-s	-26664	-30885.46	15.83	-29459.29	10.48	7200.00	369
pc100-10-1b-l	-147440	-157741.03	6.99	-153435.73	4.07	7200.00	140
pc100-10-2-s	-5180	-21495.17	314.96	-9268.04	78.92	7200.00	783
pc100-10-2-l	-130013	-138265.42	6.35	-136434.90	4.94	7200.00	592
pc100-10-2b-s	-15662	-24836.24	58.58	-17824.08	13.80	7200.00	246
pc100-10-2b-l	-157225	-161987.59	3.03	-160865.85	2.32	7200.00	262
pc100-10-3-s	-2306	-15311.03	563.96	-6642.50	188.05	7200.00	897
pc100-10-3-l	-122919	-141119.09	14.81	-135139.91	9.94	7200.00	429
pc100-10-3b-s	-25428	-34854.17	37.07	-31494.97	23.86	7200.00	259
pc100-10-3b-l	-158249	-169430.43	7.07	-165362.42	4.50	7200.00	288

Cuadro 3.14: Resultados Branch&Cut 100 clientes

Instancia	UB	Nodo Raíz		B&C Completo			
		LB	Gap	LB	Gap	Tiempo (s)	Nodos
pc200-10-1-s	-24852	-71148.58	186.29	-51665.37	107.89	7.200.00	22
pc200-10-1-l	-309573	-366201	18.29	-344913.31	11.42	7.200.00	11
pc200-10-1b-s	-65092	-96686.97	48.54	-80983.58	24.41	7.200.00	19
pc200-10-1b-l	-391121	-435226.06	11.28	-430380.77	10.04	7.200.00	7
pc200-10-2-s	-63260	-76021.22	20.17	-72187.21	14.11	7.200.00	39
pc200-10-2-l	-345084	-371084.04	7.53	-358668.08	3.94	7.200.00	38
pc200-10-2b-s	-108406	-123536.24	13.96	-115720.95	6.75	7.200.00	28
pc200-10-2b-l	-412262	-427208.37	3.63	-422663.95	2.52	7.200.00	22
pc200-10-3-s	-37886	-58139.02	53.46	-47464.9	25.28	7.200.00	28
pc200-10-3-l	-300642	-334904.72	11.40	-325562.96	8.29	7.200.00	15
pc200-10-3b-s	-64310	-84182.07	30.90	-74272.17	15.49	7.200.00	18
pc200-10-3b-l	-384762	-406596.88	5.67	-402706.55	4.66	7.200.00	7

Cuadro 3.15: Resultados Branch&Cut 200 clientes

Según puede observarse en los resultados, las instancias generadas con la finalidad de que se active, en su solución óptima, un conjunto de clientes reducido (instancias terminadas en '-s') son mucho más difíciles de resolver mediante el algoritmo *Branch&Cut* propuesto. Como se analizó anteriormente en 3.10 esto se debe a la sobre-estimación de los costos y ganancias por parte del modelo durante el desenvolvimiento del algoritmo. Se analizaron las soluciones obtenidas mediante el algoritmo metaheurístico presentado en el capítulo 2 y pudo corroborarse dicha situación. Las variables de depósito suelen coincidir rápidamente con las obtenidas en las soluciones del algoritmo metaheurístico. Sin embargo, el conjunto de clientes activos generalmente es más grande en el algoritmo exacto que en la mayor parte de las instancias analizadas terminadas en '-s'. En las instancias diseñadas para contener muchos clientes activos en su solución óptima (terminadas en '-l') la situación es muy diferente. Los gaps obtenidos por el algoritmo *Branch&Cut* son, por lo general, bastante buenos aún en las instancias de mayor tamaño y no superan en ninguno de los casos gaps del 10 %. Se logra demostrar la optimalidad de todas las intancias de 20 clientes y de las instancias de 50 clientes *pc50-5-2BIS-s* y *pc50-5-2bBIS-s*. Para la instancia *pc50-5-2BIS-s* se logra además mejorar la cota obtenida por el algoritmo metaheurístico. Asimismo, se ha logrado probar la optimalidad de la instancia *pc50-5-2bBIS-l* aumentando el tiempo de corrida del algoritmo a 22062 segundos.

3.11. Conclusiones

En este capítulo se ha presentado un algoritmo *Branch&Cut* para el problema *PC-CLRP*. De acuerdo a los análisis de resultados efectuados puede observarse que el algoritmo propuesto es efectivo para resolver por optimalidad instancias pequeñas (20 clientes) y algunas instancias medianas (50 clientes). Por otro lado, se observa que se pueden obtener resultados relativamente buenos en instancias mucho mayores y en tiempos razonables. En la mayor parte de las instancias terminadas en '-l' se obtienen resultados que están por debajo del 10 % del valor óptimo.

En base a los resultados obtenidos con el algoritmo exacto, y a los reportados en el capítulo anterior con el algoritmo metaheurístico, se puede verificar que el problema *PC-CLRP* es un problema difícil de resolver con métodos exactos. Además, el uso del algoritmo implementado ha servido para validar las soluciones del algoritmo heurístico presentado. Este algoritmo metaheurístico ha mostrado ser muy eficiente en la resolución de instancias de diferentes tamaños. En la mayor parte de las instancias evaluadas, el algoritmo de colonia de hormigas ha encontrado la solución óptima al problema o ha encontrado soluciones muy cercanas a la misma.

Para poder encarar la solución óptima de instancias de mayor tamaño, se cree que es necesario utilizar otro tipo de algoritmos o seguir avanzando con el estudio de modelos alternativos que permitan expresar en familias de desigualdades ideas similares a las usadas en los cortes de optimalidad de clientes en equilibrio o depósitos en equilibrio. Se presenta en el apéndice A.3 una formulación lineal entera alternativa basada en un nuevo modelo de 3 índices que puede utilizarse como base de futuros estudios. Esta formulación (PC3) permite expresar mejor las ideas de balance entre las ganancias y los costos. Sin embargo, al igual que otros modelos de 3 índices presentados en la literatura para problemas de ruteo, tiene como principal desventaja la necesidad de una cantidad de variables mucho más elevada y, por consiguiente, esto provoca que la resolución de cada uno de los problemas lineales sea extremadamente costosa. Sin embargo, se cree que este tipo de modelos podría llegar a ser de utilidad para el problema *PC-CLRP*.

En el estudio de algoritmos exactos alternativos, se cree que la utilización de algoritmos *Branch&Cut&Price* podría ser una muy buena opción para abordar el problema. Los trabajos recientes de Contardo y otros [22] para el problema general *CLRP* han sido muy satisfactorios utilizando este tipo de algoritmos. Los problemas *PC-CLRP* tienen características que podrían verse beneficiadas con un enfoque de esta naturaleza. Una primera idea en tal dirección podría ser la de generar columnas (variables) de la etapa de pricing utilizando los clientes activos de la instancia. Luego, podría completarse el esquema con una etapa de planos de corte que utilice algunas de las ideas presentadas en este trabajo. Las desigualdades presentadas que hacen uso de las relación existente entre las ganancias y los costos, como así también las nuevas versiones de las restricciones de capacidades, desigualdades GLM y hypotour son un ejemplo concreto de desigualdades utilizadas en el algoritmo presentado que podrían reutilizarse en un algoritmo *Branch&Cut&Price*. El algoritmo aquí presentado, utiliza una formulación basada en un modelo de 2 índices híbrido que identifica, en los nodos cliente, el depósito que los atiende. Esta versión ofrece la ventaja de no necesitar los cortes path-elimination y depot capacity constraints (también llamadas facility capacity

inequalities). Por lo tanto, las ideas del modelo presentado también podrían ser utilizadas como base de un futuro algoritmo *Branch&Cut&Price*.

Capítulo 4

CONCLUSIONES GENERALES

En este trabajo se presenta un nuevo problema denominado Localización y Ruteo con Capacidades y Premios (*PC-CLRP*). Este problema generaliza los problemas de Localización y Ruteo con Capacidades (*CLRP*) presentados previamente en la literatura. La diferencia fundamental con *CLRP* es la de permitir que los clientes puedan ser visitados o no. Se analizaron distintas variantes de problemas de ruteo estudiados en la literatura. Se relevó especialmente los trabajos efectuados para el problema *CLRP*. Además, se recorrieron los distintos estudios realizados para problemas de ruteo con premios (Prize Collecting). Se presentó un nuevo modelo exacto basado en flujo que extiende las ideas previamente utilizadas para el problema *CLRP*. Se mostró también que el problema *PC-CLRP* pertenece a la clase de problemas $\mathcal{NP} - Hard$.

Se analizaron algoritmos metaheurísticos y exactos para estudiar el problema *PC-CLRP*. En el capítulo 2 se presentó una metaheurística basada en el paradigma de colonia de hormigas. El método utilizado es un algoritmo de optimización colaborativo de múltiples colonias de hormigas basado en el sistema de colonia de hormigas (*ACS*) introducido en Dorigo y Gambardella [29]. El algoritmo presentado utiliza 3 *ACS* especializados para resolver cada uno de los problemas involucrados: localización, clusterizado y ruteo.

Para testear el algoritmo se utilizó el conjunto de instancias de Prodhon [86], que contiene instancias de pequeño, mediano y gran tamaño. Como el problema *PC-CLRP* es un caso general del problema *CLRP*, se compararon los resultados obtenidos con otras metaheurísticas publicadas en la literatura para el problema general *CLRP*. Las soluciones obtenidas mostraron que el algoritmo encuentra muy buenas soluciones y es comparable con los mejores algoritmos dedicados al problema *CLRP*. Para poder testear el algoritmo sobre la clase de problemas *PC-CLRP* se generaron nuevas instancias basadas en las instancias de Prodhon. Se presentó un algoritmo generador de instancias aleatorio que construye instancias *PC-CLRP* en base a las instancias Prodhon [86] del problema *CLRP*. Sobre las instancias generadas se aplicó el algoritmo metaheurístico presentado y se reportaron los resultados. Los mismos, al ser verificados luego mediante el algoritmo exacto presentado en el capítulo 3, muestran soluciones de muy buena calidad, alcanzando en

la mayor parte de las instancias pequeñas y en algunas medianas la solución óptima al problema o reportando soluciones muy cercanas a las mejores soluciones posibles. En las instancias medianas y grandes se obtuvieron también muy buenos resultados. Sin embargo, en muchas de ellas, el algoritmo exacto no pudo probar la optimalidad de las instancias. Las cotas inferiores obtenidas por el algoritmo exacto muestran, en la mayor parte de los casos, que las soluciones obtenidas por la metaheurística se encuentran cercanas a la solución óptima del problema.

En el capítulo 3 se presentó un algoritmo exacto tipo *Branch&Cut*. En primer lugar, se presentó una nueva formulación para el problema *PC-CLRP* basada en los modelos clásicos de flujo de 2 índices. La formulación presentada puede pensarse como la adaptación para un problema con premios de las formulaciones de 2 índices presentadas para el problema *CLRP* en Belenguer y otros [16] y Contardo y otros [21]. A diferencia de las utilizadas para el problema *CLRP*, la formulación presentada se basó en un modelo expresado en un grafo dirigido. Sin embargo, según pudo verificarse en las primeras experiencias computacionales, el modelo tal cual había sido formulado no obtenía buenas soluciones en tiempos razonables. Según se analizó en el capitulo 3, se cree que las deficiencias principales de esta formulación se encuentra en la cantidad de variables utilizadas y en los problemas de simetría derivados de trabajar sobre modelos dirigidos.

Como parte del proceso de búsqueda de alternativas para intentar mejorar los resultados obtenidos, se presentó una nueva formulación basada en un modelo de flujo de 2 índices híbrido que utiliza un grafo con ejes no dirigidos. Esta nueva formulación presenta varias mejoras con respecto a la primera formulación y tiene, gracias a la utilización de un conjunto especializado de variables binarias, la particularidad de no requerir las familias de cortes de eliminación de caminos (path elimination constraints) y las familias de cortes de capacidad de depósitos (facility capacity inequalities). Con la nueva formulación presentada se pudo obtener resultados óptimos sobre todas las instancias pequeñas (20 clientes) y algunas de las medianas (50 clientes). En los casos en que no se llegó al óptimo de la solución se obtuvieron valores cercanos a los valores óptimos o con garantía de encontrarse a una determinada distancia de los mismos.

Como parte del análisis efectuado para desarrollar el algoritmo *Branch&Cut*, se presentaron varias familias de desigualdades válidas especialmente propuestas para este modelo. Se logró adaptar también varias de las familias utilizadas para el problema *CVRP* y luego para el problema *CLRP*. Se presentaron nuevos cortes de optimalidad que lograron reducir considerablemente el árbol de enumeración del algoritmo *Branch&Cut*. Las técnicas de preprocesamiento utilizadas también mostraron muy buenos resultados. Tanto la estrategia de separación como la estrategia de branching y el algoritmo primal de mejoramiento mostraron

también mejorar los resultados reduciendo significativamente los tiempos computaciones totales en la mayor parte de las instancias. En la sección 3.10 se mostraron parte de los experimentos realizados para evaluar el impacto de cada una de las técnicas implementadas en el algoritmo *Branch&Cut* propuesto y se reportaron los resultados generales.

4.1. Trabajo futuro

El estudio realizado constituye una primera aproximación computacional al problema *PC-CLRP* presentado. Se abre un amplio camino de enfoques viables para intentar mejorar los resultados computacionales obtenidos. Entre los caminos más importantes a transitar se encuentra el estudio poliedral del modelo propuesto. Esta tarea se sabe que es muy difícil de realizar en problemas complejos como los presentados en este trabajo. No se han encontrado en la literatura estudios poliedrales para el problema *CLRP*. Un primer paso en esta dirección podría ser intentar determinar la dimensión del poliedro asociado al modelo propuesto. Analizar el poliedro podría servir como base para permitir identificar nuevas familias de desigualdades que obtengan especial provecho de la estructura que lo caracteriza. Además, esto habilitaría estudiar resultados teóricos muy importantes como ser la demostración de que determinadas familias de desigualdades son facetas del poliedro del *PC-CLRP*.

Al analizar el avance de las ejecuciones sobre las diversas instancias se pudo vislumbrar una cierta dificultad para obtener soluciones con variables w enteras. En general, se obtienen valores fraccionales que muestran un intento de repartir la demanda del cliente entre varios vehículos. A medida que las desigualdades de capacidad y el resto de las desigualdades implementadas se agregan al sistema, los valores de las variables w van tendiendo a hacerse enteros (binarios 0-1). Se considera una tarea pendiente poder vislumbrar desigualdades válidas que mejoren la integralidad de dichas variables, reforzando la relación existente entre los clientes y las rutas. Además, como se analizó en el capítulo 3, es también una tarea pendiente intentar mejorar la sobre-estimación de costos y ganancias detectada en las instancias de tipo '-s'. Para ello sería deseable conseguir desigualdades válidas o realizar modificaciones al modelo que permitan obtener desde el principio relajaciones con una cantidad de clientes activos más parecida a las obtenidas en las soluciones finales.

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 tipo de algoritmos incorporan las variables necesarias utilizando una

etapa de identificación que seleccionan 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 obtenerse importantes mejoras de rendimiento ya que, todos los clientes desactivados y muchas de las variables involucradas en el modelo, permanecerán inactivas durante la mayor parte de la optimización.

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 *PC-CLRP* 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, un primer enfoque computacional para intentar resolver el problema *PC-CLRP* utilizando herramientas exactas y heurísticas.

Capítulo A

ANEXO I

A.1. Conjuntos de instancias utilizados

Debido a que no se ha podido encontrar en la literatura trabajos para el problema *PC-CLRP*, se enfrentó la necesidad de generar instancias para poder testear los algoritmos propuestos. Para ello, se ha decidido utilizar el conjunto de 30 instancias de Prodhon [86] ideadas para el problema *CLRP*. Las mismas pueden descargarse del sitio http://prodhonc.free.fr/Instances/instances_us.htm. La estructura de cada una de las instancias originales puede observarse en el cuadro A.1. Allí se detalla la cantidad de clientes, la cantidad de localizaciones, la capacidad Q de los vehículos, las capacidades W de las localizaciones y la mejor solución *BKS* para el problema *CLRP* conocida hasta el momento para dicha instancia.

	$\|J\|$	$\|I\|$	Q	W	BKS
coord20-5-1	20	5	70	140	54793
coord20-5-1b	20	5	150	300	39104
coord20-5-2	20	5	70	{70, 140}	48908
coord20-5-2b	20	5	150	{150, 300}	37542
coord50-5-1	50	5	70	{350, 420}	90111
coord50-5-1b	50	5	150	{350, 420}	63242
coord50-5-2	50	5	70	350	88298
coord50-5-2b	50	5	150	350	67308
coord50-5-2BIS	50	5	70	350	84055
coord50-5-2bBIS	50	5	150	300	51822
coord50-5-3	50	5	70	{350, 420}	86203
coord50-5-3b	50	5	150	{350, 420}	61830
coord100-5-1	100	5	70	{700, 770}	274814
coord100-5-1b	100	5	150	{700, 770}	213615
coord100-5-2	100	5	70	{700, 770, 840}	193671
coord100-5-2b	100	5	150	{700, 770, 840}	157095
coord100-5-3	100	5	70	{770, 840}	200079
coord100-5-3b	100	5	150	{770, 840}	152441
coord100-10-1	100	10	70	{420, 490, 560}	287983
coord100-10-1b	100	10	150	{420, 490, 560}	231763
coord100-10-2	100	10	70	{420, 490, 560}	243590
coord100-10-2b	100	10	150	{420, 490, 560}	203988
coord100-10-3	100	10	70	{420, 490, 560}	250882
coord100-10-3b	100	10	150	{420, 490, 560}	204317
coord200-10-1	200	10	70	{910, 980, 1050, 1120, 1190}	477248
coord200-10-1b	200	10	150	{910, 980, 1050, 1120, 1190}	378351
coord200-10-2	200	10	70	{910, 980, 1260}	449571
coord200-10-2b	200	10	150	{910, 980, 1260}	374330
coord200-10-3	200	10	70	{910, 980, 1050, 1120, 1190}	469433
coord200-10-3b	200	10	150	{910, 980, 1050, 1120, 1190}	362817

Cuadro A.1: Instancias de Prodhon

A.2. Generación de nuevas instancias

Para modificar las instancias anteriores y que sean compatibles con el problema *PC-CLRP*, es necesario agregarle a cada cliente el beneficio que el mismo otorgaría en caso de ser atendido. Para que las instancias sean difíciles de resolver y pongan en juego la eficiencia de los algoritmos presentados es importante seleccionar ganancias que permitan a los algoritmos "dudar" de su conveniencia en las soluciones construidas. Es decir, si se eligen ganancias extremadamente pequeñas, los clientes quedaran siempre fuera de cualquier

solución prometedora. Por el contrario, si se eligen ganancias muy altas, los clientes son siempre atendidos en cualquier solución eficiente. Por lo tanto, para poder generar instancias competitivas es necesario establecer en cada cliente un rango de ganancias razonable para luego poder seleccionar algún valor en dicho rango aleatoriamente. El algoritmo propuesto implementa estas ideas.

Para calcular los extremos de cada rango, será necesario estudiar los costos asociados a cada cliente. Se conocen las distancias a todos los otros clientes o depósitos, los costos por vehículo utilizado y el costo de apertura de cada depósito. Para que un cliente sea atendido, la ganancia que otorga debe superar a los costos que acarrea. La función utilizada para calcular la ganancia mínima aceptable para cada cliente i es la siguiente

$$b_i^{min} = \overset{1}{\min_{j \in V}} c_{ij} + \overset{2}{\min_{j \in V}} c_{ij} \tag{A.1}$$

donde $\min^1$ y $\min^2$ son la primera y segunda distancia mínima desde i a otros nodos. La ecuación que calcula la ganancia máxima aceptable es

$$b_i^{max} = \overset{1}{\max_{j \in V}} c_{ij} + \overset{2}{\max_{j \in V}} c_{ij} + (F + \max_{j \in I} O_j)/|J| \tag{A.2}$$

donde $\max^1$ y $\max^2$ son la primer y segunda distancia máxima respectivamente, F es el costo de utilizar cada vehículo y O_j es el costo de apertura del depósito j. La idea de esta ecuación es contemplar la posibilidad de utilizar los ejes más costosos y, al mismo tiempo, contemplar la parte del costo de utilizar los vehículos y la apertura de depósitos. El beneficio calculado por cliente b_i^* se determina utilizando el rango anterior

$$b_i^{min} \leq b_i^* \leq b_i^{max} \tag{A.3}$$

Por cada instancia de Prodhon se ha decidido generar 2 nuevas instancias *PC-CLRP*. Las 60 instancias generadas se encuentran detalladas en los cuadros A.2 y A.3 y pueden descargarse del sitio `http://www.dc.uba.ar/People/negrotto`. Las instancias que finalizan su nombre con '-s' son instancias que se espera tengan pocos clientes atendidos en su solución óptima. Mientras que las que finalizan con '-l' son instancias que se supone que tendrán bastantes clientes atendidos. Para poder generar estas 2 instancias por cada instancia original, se ha decidido multiplicar el beneficio por cliente b_i^* obtenido por un factor de cubrimiento p. Este factor de cubrimiento es un parámetro $0 \leq p \leq 1$ del algoritmo generador que se multiplica al beneficio calculado. De esta manera, si el parámetro es cercano a 0 se estará forzando la generación de instancias con un número reducido de clientes atendidos y si es cercano a 1 lo contrario. La elección del parámetro p se ha efectuado mediante experimentación y se ha modificado para cada instancia. Las instancias obtenidas fueron testeadas con la metaheurística de colonia de hormigas presentada en el

capítulo 2. En caso de no obtenerse soluciones con pocos o muchos clientes atendidos se va modificando el parámetro p hasta obtener instancias acordes a lo buscado.

En los cuadros A.2 y A.3 se muestra el total de los beneficios por instancia ($\sum b$) y la cantidad de depósitos, vehículos y clientes atendidos. Los resultados que dan lugar a los últimos 3 valores fueron obtenidos mediante el algoritmo de colonia de hormigas presentado en el capítulo 2.

	$\sum b$	**Depósitos**	**Vehículos**	**Clientes Atendidos**
pc20-5-1-s	52300	1	2	8
pc20-5-1-l	81600	2	4	17
pc20-5-1b-s	30900	1	1	10
pc20-5-1b-l	87300	1	2	19
pc20-5-2-s	42600	1	3	9
pc20-5-2-l	86600	2	4	16
pc20-5-2b-s	30000	1	2	19
pc20-5-2b-l	90700	1	2	19
pc50-5-1-s	82500	2	7	32
pc50-5-1-l	124100	2	9	40
pc50-5-1b-s	65900	2	5	47
pc50-5-1b-l	124100	2	5	47
pc50-5-2-s	76500	1	5	22
pc50-5-2-l	120200	2	10	42
pc50-5-2b-s	58800	2	4	39
pc50-5-2b-l	120200	2	6	44
pc50-5-2BIS-s	52300	1	4	19
pc50-5-2BIS-l	81400	1	5	24
pc50-5-2bBIS-s	41800	1	2	20
pc50-5-2bBIS-l	74500	2	4	39
pc50-5-3-s	79000	1	5	23
pc50-5-3-l	120400	2	10	44
pc50-5-3b-s	79000	1	3	28
pc50-5-3b-l	120400	2	5	47

Cuadro A.2: Instancias generadas 1 - PC-CLRP

	$\sum b$	**Depósitos**	**Vehículos**	**Clientes Atendidos**
pc100-5-1-s	254000	1	12	51
pc100-5-1-l	468600	2	21	90
pc100-5-1b-s	186700	1	6	50
pc100-5-1b-l	321200	2	10	91
pc100-5-2-s	189600	1	13	55
pc100-5-2-l	334600	2	22	93
pc100-5-2b-s	189600	1	6	55
pc100-5-2b-l	334600	2	11	99
pc100-5-3-s	179600	1	13	55
pc100-5-3-l	314400	2	21	91
pc100-5-3b-s	179600	1	6	54
pc100-5-3b-l	314400	2	11	98
pc100-10-1-s	204100	1	8	36
pc100-10-1-l	352100	2	15	64
pc100-10-1b-s	204100	1	4	37
pc100-10-1b-l	352100	2	8	70
pc100-10-2-s	185200	1	9	37
pc100-10-2-l	328400	2	17	70
pc100-10-2b-s	170800	1	4	38
pc100-10-2b-l	328400	2	8	70
pc100-10-3-s	190600	1	7	32
pc100-10-3-l	337900	2	17	68
pc100-10-3b-s	190600	1	4	36
pc100-10-3b-l	337900	2	8	68
pc200-10-1-s	405900	1	18	81
pc200-10-1-l	741600	2	36	155
pc200-10-1b-s	373000	1	9	81
pc200-10-1b-l	741600	2	15	138
pc200-10-2-s	384400	1	19	84
pc200-10-2-l	724000	2	34	146
pc200-10-2b-s	384400	1	9	84
pc200-10-2b-l	724000	2	16	146
pc200-10-3-s	382400	1	17	74
pc200-10-3-l	704900	2	32	138
pc200-10-3b-s	350100	1	8	75
pc200-10-3b-l	704900	2	16	147

Cuadro A.3: Instancias generadas 2 - PC-CLRP

A.3. Modelo 3 índices

Como alternativa a los modelos para el *PC-CLRP* presentados en las secciones 1.1 y 3.6.2, se consideró también analizar un modelo de 3 índices. La desventaja principal de este modelo es el gran incremento en el número de variables que representan a los ejes del grafo. Los modelos de 3 índices utilizan un eje por localización, siendo el número total de variables x del orden de $n^2 * m$, tomando $n = |J|$ y $m = |I|$. La ventaja de este modelo para el problema *PC-CLRP* es que se podrían expresar nuevas familias de restricciones que relacionen los costos con los beneficios que no son aplicables al modelo de 2 índices presentado. Esto supone una mejora en la relajación del problema inicial que puede llegar a disminuir el costo global del algoritmo *Branch&Cut*.

La formulación presentada se basa como antes en un grafo no dirigido $G = (V, E)$ con un conjunto de nodos V, donde $V = I \cup J$ representa la unión de las posibles localizaciones de depósitos y el conjunto de clientes, y E representa el conjunto de ejes no dirigidos que conecta a los clientes entre sí y a los clientes con las posibles localizaciones. En este caso se representará con $|I|$ multiejes a cada uno de los ejes del grafo, representando de esta manera el depósito responsable de la ruta que pasa por cada eje.

El nuevo modelo utiliza $|I|$ ejes para representar la conexión entre dos nodos. Estos ejes seran representados en la formulación de programación lineal entera por $|I|$ variables binarias. Se utilizará aquí nuevamente variables binarias específicas y para representar las rutas de un sólo cliente. Se cumple que $x_{ij}^k = 1$ si el eje $(i, j) \in E$ se usa en alguna ruta desde el depósito k y 0 en caso contrario; $y_{ij} = 1$ si el cliente j es usado en una ruta de un solo cliente desde la localización i o 0 en caso contrario; $z_i = 1$ si la localización i es utilizada y 0 en caso contrario; $w_{ij} = 1$ si el cliente j es atendido desde la localización i y 0 en caso contrario.

Se mantiene la notación utilizada en (PC2) y (PC2E) y se redefine la notación requerida para expresar el modelo sobre un conjunto de multiejes. Para cada subconjunto $U \subseteq V$ se define $E(U) = \{(u, w) \in E : u, w \in U\}$ y $\delta(U) = \{(u, w) \in E : u \in U, w \notin U\}$. Para cada par de subconjuntos disjuntos U y W se define $(U : W) = \{(u, w) \in E : u \in U, w \in W\}$. Para un conjunto dado $M \subseteq E$ se define $x^i(M) = \sum_{e \in M} x_e^i$ y para un conjunto $M' \subseteq \delta(I)$ se define $y(M') = \sum_{e \in M'} y_e$. Para cada cliente j se define $w(j) = \sum_{i \in I} w_{ij}$ y para un conjunto de clientes S se establece que $w(S) = \sum_{j \in S, k \in I} w_{kj}$ y $w^k(S) = \sum_{j \in S} w_{kj}$. Para un conjunto de clientes $S \subseteq J$, se define $d(S) = \sum_{j \in S} d_j$ y $b(S) = \sum_{j \in S} b_j$.

La formulación (PC3) para el modelo de 3 índices es la siguiente:

$$min \sum_{e\in E,k\in I} c_e x_e^k + 2\sum_{e\in\delta(I)} c_e y_e + \sum_{i\in I} O_i z_i + \sum_{e\in\delta(I),k\in I} \frac{F}{2} x_e^k + \sum_{e\in\delta(I),k\in I} F y_e - \sum_{j\in J} b_j w(j) \tag{A.4}$$

sujeto a

$$\sum_{k\in I} x^k(\delta(j)) + 2y(I:j) \leq 2 \qquad j \in J \tag{A.5}$$

$$x^k(E(S)) \leq w^k(S) - \sum_{j\in S} \frac{d_j w_{kj}}{Q} \qquad S \subseteq J \quad k \in I \tag{A.6}$$

$$x_{ij}^i + y_{ij} \leq z_i \qquad i \in I \quad j \in J \tag{A.7}$$

$$x_{ij}^i + y_{ij} \leq w_{ij} \qquad i \in I \quad j \in J \tag{A.8}$$

$$\sum_{k\in I} x^k(I:j) + \sum_{k\in I} y(I:j) \leq 1 \qquad j \in J \tag{A.9}$$

$$\sum_{j\in J} w_{ij} \leq |J| z_i \qquad i \in I \tag{A.10}$$

$$\sum_{j\in J} d_j w_{ij} \leq W_i z_i \qquad i \in I \tag{A.11}$$

$$x^i(\delta(j)) + 2y_{ij} = 2w_{ij} \qquad j \in J \quad i \in I \tag{A.12}$$

$$x_e^k \in \{0,1\} \qquad e \in E \quad k \in I \tag{A.13}$$

$$y_e \in \{0,1\} \qquad e \in \delta(I) \tag{A.14}$$

$$z_i \in \{0,1\} \qquad i \in I \tag{A.15}$$

$$w_{ij} \in \{0,1\} \qquad i \in I \quad j \in J \tag{A.16}$$

La función objetivo (A.4) representa la minimización del costo total y la maximización simultánea de los beneficios. El costo se define como la suma de los costos de ruta, los costos fijos de apertura de los depósitos utilizados y los costos de los vehículos usados. Estos últimos son contabilizados mediante el flujo saliente del conjunto de depósitos. Los beneficios se calculan sumando los premios de cada cliente activo (clientes con valor 1 en las variables w).

Las desigualdades (A.5) son las restricciones de grado para los nodos clientes. Junto a las restricciones (A.12) determinan el grado de cada cliente y al mismo tiempo sincronizan el valor de las variables w con el flujo activo en cada cliente. Las restricciones (A.6) son una nueva versión desagregada por depósito k de las desigualdades de capacidad para los vehículos, especializada en nuestro problema específico *PC-CLRP*.

Las desigualdades (A.7) establecen que los ejes incidentes a un depósito sólo deben ser usados si el depósito está activo. Se puede ver que $x_{ij}^i + y_{ij} \leq 1$ siempre se cumple debido a que las variables x_{ij}^i y y_{ij}

no pueden valer simultáneamente 1. Las desigualdades (A.8), por su parte, establecen que las variables w_{ij} deben activarse para los clientes que inician y terminan sus rutas en el depósito i. La familia de restricciones (A.9) elimina soluciones ilegales donde una ruta comienza en un depósito, atiende un cliente y termina luego en otro depósito.

Por su parte, las desigualdades (A.10) establecen que si hay variables w activadas para un depósito, entonces el mismo debe encontrarse activado. Las restricciones (A.11) imponen las restricciones de capacidad para los depósitos.

Por último, las desigualdades (A.13)-(A.16) son las restricciones de integralidad.

Las familias de restricciones (A.6) contienen un número de restricciones que crece exponencialmente de acuerdo al tamaño de J. Para poder utilizar esta formulación de manera eficiente se deberían usar estas restricciones dentro de un algoritmo de planos de corte. Habría que identificar las restricciones de capacidad violadas y agregarlas cuando las mismas sean necesitadas. Esto es equivalente a lo efectuado para la formulación (PC2E). Los cortes utilizados para la formulación (PC2E) pueden ser adaptados para la formulación de 3 índices presentada.

Ahora bien, con este modelo, desigualdades como las de depósitos en equilibrio (3.32) pueden expresarse incluyendo los costos de los ejes de las rutas. Esto seguramente debería ayudar a mejorar las cotas de la relajación inicial y del resto del algoritmo *Branch&Cut*.

$$\sum_{j \in J} b_j w_{ij} \geq O_i z_i + \frac{F}{2} x^i(\delta(i)) + F y(\delta(i)) + \sum_{e \in E} c_e x_e^i + \sum_{e \in \delta(i)} 2 c_e y_e \qquad i \in I$$

Capítulo B

GLOSARIO DE ACRÓNIMOS

ACO	Ant Colony Optimization
ACS	Ant Colony System
ALNS	Adaptive Large-Neighborhood Search
AS	Ant System
BKS	Best Known Solution
Branch&Bound	Branch&Bound
Branch&Cut	Branch&Cut
Branch&Cut&Price	Branch&Cut&Price
Branch&Price	Branch&Price
BRP	Bank Robber Problem
CFLP	Capacitated Facility Location Problem
CLRP	Capacitated Location Routing Problem
CTOP	Capacitated Team Orienteering Problem
CVRP	Capacitated Vehicle Routing Problem
FLP	Facility Location Problem
GLRPP	Generalized Location Routing Problem with Profits

GRASP	Greedy randomized adaptive search procedure
GTS	Granular Tabu Search
ILP	Integer Linear Programming
IRACE	Iterated Racing for Automatic Algorithm Configuration Software
LB	Lower Bound
LEMON	Library for Efficient Modeling and Optimization in Networks
LOP	Lineal Ordering Problem
LP	Linear Programming
LRP	Location Routing Problem
MACO	Multiple Ant Colony Optimization
MACS	Multiple Ant Colony System
MCLP-P	Partial Maximal Coverage Location Problem
MCP	Maximum Collection Problem
MDVRP	Multiple Depot Vehicle Routing Problem
MELRP	Multi-Echelon Location-Routing Problems
MILP	Mixed Integer Linear Programming
MTMCP	Multiple Tour Maximum Collection Problem
OP	Orienteering Problem
PC-CLRP	Prize-Collecting Capacitated Location Routing Problem
PC-CVRP	Prize-Collecting Capacitated Vehicle Routing Problem
PCTSP	Prize-Collecting Traveling Salesman Problem
PTP	Profitable Tour Problem

SA	Simulated Annealing
SEFLP	Single-Echelon Facility Location Problem
SPP	Set-Packing Problem
SPP	Set-Partitioning Problem
STSP	Selective Traveling Salesperson Problem
TOP	Team Orienteering Problem
TOPTW	Team Orienteering Problem with Time Windows
TSP	Traveling Salesman Problem
UB	Upper Bound
VLNS	Variable Large Neighborhood Search
VND	Variable Neighborhood Descent
VNS	Variable Neighborhood Search
VRP	Vehicle Routing Problem
VRP-P	Vehicle Routing Problem with Profits

Capítulo C

GLOSARIO DE NOTACIÓN

A	Conjunto de ejes dirigidos
$E(U)$	Ejes con ambos nodos en U
F	Costo de utilizar un vehículo
$G = (V, A)$	Grafo dirigido
$G = (V, E)$	Grafo no dirigido
I	Conjunto de Posibles Localizaciones
J	Conjunto de clientes
L^{bs}	Valor de la función objetivo de la mejor solución del algoritmo de colonia de hormigas
L^{it}	Valor de la función objetivo de la mejor solución de la iteración del algoritmo de colonia de hormigas
N_j^k	Vecindad de tamaño k del nodo j
O_i	Costo de apertura de la localización i
Q	Capacidad de los vehículos (homogénea)
$U : W$	Ejes con un nodo en W y otro en U
V	Conjunto de Nodos
W_i	Capacidad máxima de la localización i
Z^{bs}	Mejor solución del algoritmo de colonia de hormigas
Z^{it}	Mejor solución de la iteración en el algoritmo de colonia de hormigas
α_C	Proporción de feromona en la regla aleatoria proporcional para clusterizado
α_L	Proporción de feromona en la regla aleatoria proporcional para localización
α_R	Proporción de feromona en la regla aleatoria proporcional para ruteo

β_C	Proporción de nivel de atracción en la regla aleatoria proporcional para clusterizado
β_L	Proporción de nivel de atracción en la regla aleatoria proporcional para localización
β_R	Proporción de nivel de atracción en la regla aleatoria proporcional para ruteo
$\delta(U)$	Ejes no dirigidos con un nodo en U y otro fuera de U
$\delta^+(U)$	Ejes dirigidos con la cola en U y la cabeza fuera de U
$\delta^-(U)$	Ejes dirigidos con la cabezas en U y la cola fuera de U
ϵ	Número de iteración
η_{ij}	Nivel de atracción para el cliente i asignado al depósito j
η_j	Nivel de atracción en la localización j
$\mathbb{R}$	Conjunto de números reales
$\mathcal{NP} - Hard$	Clase de problemas polinomiales no deterministas difíciles
$\mathcal{NP}$	Clase de problemas polinomiales no deterministas
$\mathcal{P}$	Clase de problemas polinomiales
ϕ_{ij}	Nivel de atracción para el eje (i, j)
π_j	Nivel de feromona en la localización j
ψ_{ij}	Nivel de feromona para el eje (i, j)
ρ_C	Proporción de feromona que se evapora o actualiza en la regla de actualización del ACS de clusterizado
ρ_L	Proporción de feromona que se evapora o actualiza en la regla de actualización del ACS de localización
ρ_R	Proporción de feromona que se evapora o actualiza en la regla de actualización del ACS de ruteo
τ_{ij}	Nivel de feromona para el cliente i asignado al depósito j
θ	Parámetro que determina el nivel de ejecución de etapas de clonado en el algoritmo de colonia de hormigas
$b(S)$	Suma de los beneficios de los clientes en el conjunto S
b_j	Beneficio o ganancia otorgada cuando el cliente j es atendido
c_{ij}	Costo de atravezar el eje (i, j)
$conv(S)$	Cápsula convexa del conjunto S
$d(S)$	Suma de las demandas de los clientes en el conjunto S
d_j	Demanda del cliente j
gap	Distancia a la mejor solución conocida BKS para la instancia

k	Tamaño de la vecindad en el algoritmo de colonia de hormigas
$maxiterClonado$	Parámetro del algoritmo de colonia de hormigas que indica la cantidad máxima de iteraciones en la etapa de clonación
$maxiter$	Parámetro del algoritmo de colonia de hormigas que indica la cantidad máxima de iteraciones
$nAnts$	Cantidad de hormigas en el algoritmo de colonia de hormigas
p_j^ϵ	Función que selecciona la próxima localización a abrir en base a la regla clásica ACS aleatoria proporcional
$parPricing$	Parámetro que determina el nivel aceptación de clientes activos
q_{0C}	Parámetro que determina el nivel de aleatoriedad de la regla pseudoaleatoria proporcional para clusterizado
q_{0R}	Parámetro que determina el nivel de aleatoriedad de la regla pseudoaleatoria proporcional para ruteo
$w(S)$	Suma de los valores de las variables w en el conjunto de clientes S
$w(j)$	Suma de los valores de las variables w para el cliente j
$w^k(S)$	Suma de los valores de las variables w en el conjunto de clientes S para el depósito k
$x(M)$	Suma de los valores de los ejes x en el conjunto M
$x^i(M)$	Suma de los valores de los ejes x en el conjunto M para el depósito i
x_{ij}	Variable binaria que indica si el eje (i, j) es utilizado
$y(M')$	Suma de los valores de los ejes y en el conjunto M'
y_{ij}	Variable binaria que indica si el eje (i, j) es utilizado, siendo i un depósito y j un cliente
z_i	Variable binaria que indica si el depósito i es utilizado

BIBLIOGRAFÍA

[1] T. Achterberg, T. Koch, y A. Martin, "Branching rules revisited," *Operations Research Letters*, vol. 33, pp. 42–54, 2004.

[2] J. Ahn, O. de Weck, Y. Geng, y D. Klabjan, "Column generation based heuristics for a generalized location routing problem with profits arising in space exploration," *European Journal of Operational Research*, vol. 223, no. 1, pp. 47 – 59, 2012.

[3] J. Ahn, O. De Weck, y J. Hoffman, "An optimization framework for global planetary surface exploration campaigns," *Journal of the British Interplanetary Society*, vol. 61, no. 12, p. 487, 2008.

[4] Z. Akca, R. Berger, y T. Ralphs, *A Branch-and-Price Algorithm for Combined Location and Routing Problems Under Capacity Restrictions*, in *Operations Research and Cyber-Infrastructure*, ser. Operations Research/Computer Science Interfaces, vol. 47. Springer US, 2009.

[5] D. J. A. Albareda-Sambola, M. y E. Fernndez, "A compact model and tight bounds for a combined location-routing problem," *Computers & Operations Research*, vol. 32, no. 3, pp. 407 – 428, 2005.

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

[7] C. Archetti, D. Feillet, A. Hertz, y M. G. Speranza, "The capacitated team orienteering and profitable tour problems," *Journal of the Operational Research Society*, vol. 60, no. 6, pp. 831–842, 2009.

[8] C. Archetti, M. G. Speranza, y D. Vigo, *Chapter 10: Vehicle Routing Problems with Profits*, in *Vehicle Routing*, D. Vigo y P. Toth, Eds. Society for Industrial and Applied Mathematics, 2014.

[9] P. Augerat, *Approche poly'edrale du probléme de tournées de véhicles*. Polytechnique de Grenoble: PhD thesis, 1995.

[10] B. Awerbuch, Y. Azar, A. Blum, y S. Vempala, "New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen," *SIAM Journal on Computing*, vol. 28, no. 1, pp. 254–262, 1998.

[11] E. Balas, "The prize collecting traveling salesman problem: Ii. polyhedral results," *Networks*, vol. 25, no. 4, pp. 199–216, 1995.

[12] E. Balas, "New classes of efficiently solvable generalized traveling salesman problems," *Annals of Operations Research*, vol. 86, pp. 529–558, 1999.

[13] R. Baldacci, A. Mingozzi, y R. W. Calvo, "An exact method for the capacitated location-routing problem," *Operations Research*, vol. 59, no. 5, pp. 1284–1296, 2011.

[14] F. Barahona y M. Grötschel, "On the cycle polytope of a binary matroid," *Journal of Combinatorial Theory, Series B*, vol. 40, no. 1, pp. 40 – 62, 1986.

[15] F. C. P. a. J. Barreto, S. y B. S. Santos, "Using clustering analysis in a capacitated location-routing problem," *European Journal of Operational Research*, vol. 179, no. 3, pp. 968–977, 2007.

[16] J. M. Belenguer, E. Benavent, C. Prins, C. Prodhon, y R. W. Calvo, "A branch-and-cut method for the capacitated location-routing problem," *Computers & Operations Research*, vol. 38, no. 6, pp. 931 – 941, 2011.

[17] S. E. Butt y T. M. Cavalier, "A heuristic for the multiple tour maximum collection problem," *Computers & Operations Research*, vol. 21, no. 1, pp. 101 – 111, 1994.

[18] C. P. C. Duhamel, P. Lacomme y C. Prodhon, "A grasp x els approach for the capacitated location-routing problem," *Computers & Operations Research*, vol. 37, no. 11, pp. 1912–1923, 2010.

[19] I.-M. Chao, B. L. Golden, y E. A. Wasil, "The team orienteering problem," *European journal of operational research*, vol. 88, no. 3, pp. 464–474, 1996.

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

[21] C. Contardo, J.-F. Cordeau, y B. Gendron, "A computational comparison of flow formulations for the capacitated location-routing problem," *Discrete Optimization*, vol. 10, no. 4, pp. 263 – 295, 2013.

[22] C. Contardo, J.-F. Cordeau, y B. Gendron, "An exact algorithm based on cut-and-column generation for the capacitated location-routing problem," *INFORMS Journal on Computing*, vol. 26, no. 1, pp. 88–102, 2014.

[23] C. Contardo, J.-F. Cordeau, y B. Gendron, "A grasp + ilp-based metaheuristic for the capacitated location-routing problem," *Journal of Heuristics*, vol. 20, no. 1, pp. 1–38, 2014.

[24] H. Crowder, E. L. Johnson, y M. Padberg, "Solving large-scale zero-one linear programming problems," *Operations Research*, vol. 31, no. 5, pp. 803–834, 1983.

[25] G. Dantzig, R. Fulkerson, y S. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the Operations Research Society of America*, vol. 2, no. 4, pp. 393–410, 1954.

[26] M. Dell'Amico, F. Maffioli, y A. Sciomachen, "A lagrangian heuristic for the prize collectingtravelling salesman problem," *Annals of Operations Research*, vol. 81, pp. 289–306, 1998.

[27] M. Dell'Amico, F. Maffioli, y P. Värbrand, "On prize-collecting tours and the asymmetric travelling salesman problem," *International Transactions in Operational Research*, vol. 2, no. 3, pp. 297–308, 1995.

[28] B. Dezsö, A. Jüttner, y P. Kovács, "LEMON - an open source C++ graph template library," *Electronic Notes in Theoretical Computer Science*, vol. 264, no. 5, pp. 23 – 45, 2011, proceedings of the Second Workshop on Generative Technologies (WGT) 2010. [Online] http://lemon.cs.elte.hu/trac/lemon

[29] M. Dorigo y L. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 53–66, Apr 1997.

[30] M. Dorigo, V. Maniezzo, y A. Colorni, "Ant system: optimization by a colony of cooperating agents," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 26, no. 1, pp. 29–41, Feb 1996.

[31] M. Dorigo y T. Stützle, *Ant Colony Optimization*, ser. A Bradford book. Bradford book, 2004.

[32] M. Dorigo, "Optimization, learning and natural algorithms," *Ph. D. Thesis, Politecnico di Milano, Italy*, 1992.

[33] M. Dorigo, V. Maniezzo, y A. Colorni, "Positive feedback as a search strategy," Politecnico di Milano, Italy, Tech. Rep. 91-016, 1991.

[34] J. W. Escobar, "Heuristic algorithms for the capacitated location-routing problem and the multi-depot vehicle routing problem," Ph.D. dissertation, Università di Bologna, 2013.

[35] J. W. Escobar, R. Linfati, y P. Toth, "A two-phase hybrid heuristic algorithm for the capacitated location-routing problem," *Computers & Operations Research*, vol. 40, no. 1, pp. 70 – 79, 2013.

[36] D. Feillet, P. Dejax, y M. Gendreau, "Traveling salesman problems with profits," *Transportation science*, vol. 39, no. 2, pp. 188–205, 2005.

[37] M. Fischetti y P. Toth, "An additive approach for the optimal solution of the prize-collecting travelling salesman problem. vehicle routing: Methods and studies. studies in management science and systems-volume 16," *Publication of: Dalctraf*, 1988.

[38] M. M. Flood, "The traveling-salesman problem," *Operations Research*, vol. 4, no. 1, pp. 61–75, 1956.

[39] L. M. Gambardella, R. Montemanni, y D. Weyland, "Coupling ant colony systems with strong local searches," *European Journal of Operational Research*, vol. 220, no. 3, pp. 831–843, 2012.

[40] L. M. Gambardella, É. Taillard, y G. Agazzi, "Macs-vrptw: A multiple colony system for vehicle routing problems with time windows," in *New Ideas in Optimization*, pp. 63–76. McGraw-Hill, 1999.

[41] B. Gavish, "The delivery problem: New cutting planes procedure." Copenhagen: presented at the TIMS XXVI conference, 1984.

[42] M. Gendreau, G. Laporte, y F. Semet, "A branch-and-cut algorithm for the undirected selective traveling salesman problem," *Networks*, vol. 32, no. 4, pp. 263–273, 1998.

[43] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Comput. Oper. Res.*, vol. 13, no. 5, pp. 533–549, May 1986.

[44] B. L. Golden, L. Levy, y R. Vohra, "The orienteering problem," *Naval Research Logistics (NRL)*, vol. 34, no. 3, pp. 307–318, 1987.

[45] B. L. Golden, S. Raghavan, y E. A. Wasil, *The Vehicle Routing Problem: Latest Advances and New Challenges*, vol. 43. Springer Science & Business Media, 2008.

[46] R. E. Gomory, "Outline of an algorithm for integer solutions to linear programs," *Bull. Amer. Math. Soc.*, vol. 64, no. 5, pp. 275–278, 09 1958.

[47] L. Gouveia, *A result on projection for the vehicle routing problem.* 610-624: European journal of operational research 85, 1995.

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

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

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

[51] M. Grtschel, L. Lovsz, y A. Schrijver, "The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica*, vol. 1, no. 2, pp. 169–197, 1981.

[52] P. H. Hansen, B. Hegedahl, S. Hjortkjaer, y B. Obel, "A heuristic solution to the warehouse location-routing problem," *European Journal of Operational Research*, vol. 76, no. 1, pp. 111–127, 1994.

[53] V. C. Hemmelmayr, J.-F. Cordeau, y T. G. Crainic, "An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics," *Computers & Operations Research*, vol. 39, no. 12, pp. 3215–3228, 2012.

[54] IBM, "IBM ILOG CPLEX optimizer version 12.6.1," 2014. [Online] http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[55] M. Jabal-Ameli, M. Aryanezhad, y N. Ghaffari-Nasab, "A variable neighborhood descent based heuristic to solve the capacitated location-routing problem," *International Journal of Industrial Engineering Computations*, vol. 2, no. 1, pp. 141–154, 2011.

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

[57] A. Jokar y R. Sahraeian, "A heuristic based approach to solve a capacitated location-routing problem," *Journal of Management and Sustainability*, vol. 2, no. 2, pp. 219 – 226, 2012.

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

[59] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica*, vol. 4, no. 4, pp. 373–395, 1984.

[60] N. Labadie, J. Melechovskỳ, y R. W. Calvo, "Hybridized evolutionary local search algorithm for the team orienteering problem with time windows," *Journal of Heuristics*, vol. 17, no. 6, pp. 729–753, 2011.

[61] G. Laporte y S. Martello, "The selective travelling salesman problem," *Discrete applied mathematics*, vol. 26, no. 2, pp. 193–207, 1990.

[62] G. Laporte, H. Mercure, y Y. Nobert, "An exact algorithm for the asymmetrical capacitated vehicle routing problem," *Networks*, vol. 16, no. 1, pp. 33–46, 1986.

[63] A. Letchford, G. Reinelt, y D. Theis, "Odd minimum cut-sets and b-matchings revisited," *SIAM Journal on Discrete Mathematics*, vol. 22, no. 4, pp. 1480–1487, 2008.

[64] S.-W. Lin y F. Y. Vincent, "A simulated annealing heuristic for the team orienteering problem with time windows," *European Journal of Operational Research*, vol. 217, no. 1, pp. 94–107, 2012.

[65] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, y M. Birattari, "The irace package, iterated race for automatic algorithm configuration," IRIDIA, Universit Libre de Bruxelles, Belgium, Tech. Rep. TR/IRIDIA/2011-004, 2011.

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

[67] J. Lysgaard, A. N. Letchford, y R. W. Eglese, "A new branch-and-cut algorithm for the capacitated vehicle routing problem," *Mathematical Programming*, vol. 100, p. 2004, 2003.

[68] D. Maniezzo, M. Dorigo, V. Maniezzo, y A. Colorni, "Ant system: An autocatalytic optimizing process," Tech. Rep. 91-016, 1991.

[69] F. E. Maranzana, "On the location of supply points to minimise transport costs," *Operational Research*, vol. Quarterly 15, pp. 261–270, 1964.

[70] S. Martello, D. Pisinger, y P. Toth, "Dynamic programming and strong bounds for the 0-1 knapsack problem," *Manage. Sci.*, vol. 45, no. 3, pp. 414–424, 1999. [Online] http://www.diku.dk/~pisinger/combo.c

[71] P. C. Melechovský, J. y R. Wolfler-Calvo, "A metaheuristic to solve a location-routing problem with non-linear costs," *Journal of Heuristics*, vol. 11, pp. 375 – 391, 2005.

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

[73] H. Min, V. Jayaraman, y R. Srivastava, "Combined location-routing problems: A synthesis and future research directions," *European Journal of Operational Research*, vol. 108, no. 1, pp. 1 – 15, 1998.

[74] R. Montemanni y L. Gambardella, "An ant colony system for team orienteering problems with time windows," *Foundations of computing and Decision Sciences*, vol. 34, pp. 287–306, 2009.

[75] G. Nagy y S. Salhi, "Location-routing: Issues, models and methods," *European Journal of Operational Research*, vol. 177, no. 2, pp. 649 – 672, 2007.

[76] G. L. Nemhauser y L. A. Wolsey, *Integer and Combinatorial Optimization.* New York, NY, USA: Wiley-Interscience, 1988.

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

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

[79] M. Padberg, "On the facial structure of set packing polyhedra," *Mathematical Programming*, vol. 5, no. 1, pp. 199–215, 1973.

[80] J. Perl y M. S. Daskin, "A warehouse location-routing problem," *Transportation Research Part B: Methodological*, vol. 19, no. 5, pp. 381–396, 1985.

[81] S. Pirkwieser y G. R. Raidl, "Variable neighborhood search coupled with ilp-based very large neighborhood searches for the (periodic) location-routing problem," *Hybrid metaheuristics. Lecture notes in computer science*, vol. 6373, pp. 174–189, 2010.

[82] E. Polat, "A location and routing-with-profit problem in glass recycling," Master's thesis, Middle East Technical University, 2008.

[83] C. Prins, C. Prodhon, y R. W. Calvo, "A memetic algorithm with population management (ma—pm) for the capacitated location-routing problem," vol. 3906, pp. 183–194, 2006.

[84] C. Prins, C. Prodhon, y R. Calvo, "Solving the capacitated location-routing problem by a grasp complemented by a learning process and a path relinking," *4OR*, vol. 4, no. 3, pp. 221–238, 2006.

[85] C. Prins, C. Prodhon, A. Ruiz, P. Soriano, y R. W. Calvo, "Solving the capacitated location-routing problem by a cooperative lagrangean relaxation-granular tabu search heuristic," *Transportation Science*, vol. 41, no. 4, pp. 470–483, 2007.

[86] C. Prodhon, "Le problème de localisation-routage (the location-routing problem)," *Ph.D. thesis, Troyes University of Technology, (in French)*, 2006. [Online] http://prodhonc.free.fr/Instances/instances_us.htm

[87] C. Prodhon y C. Prins, "A survey of recent research on location-routing problems," *European Journal of Operational Research*, vol. 238, no. 1, pp. 1 – 17, 2014.

[88] K. S. y M. S., "Single constraint maximum collection problem," *Journal of the Operations Reseach Society of Japan*, vol. 31, pp. 515 – 530, 1988.

[89] S. Salhi y G. K. Rand, "The effect of ignoring routes when locating depots," *European Journal of Operational Research*, vol. 39, no. 2, pp. 150 – 156, 1989.

[90] T. T., "Heuristic methods applied to orienteering," *Journal of the Operational Research Society*, vol. 35, pp. 797–809, 1984.

[91] T. Thomadsen y T. Stidsen, *The Quadratic Selective Travelling Salesman Problem*, 2003.

[92] C.-J. Ting y C.-H. Chen, "A multiple ant colony optimization algorithm for the capacitated location routing problem," *International Journal of Production Economics*, vol. 141, no. 1, pp. 34 – 44, 2013.

[93] P. Toth y D. Vigo, *The Vehicle Routing Problem*, ser. Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2002.

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

[95] P. Toth y D. Vigo, *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, D. Vigo y P. Toth, Eds. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2014.

[96] E. Tresoldi, "Location and routing problems: an unified approach," Ph.D. dissertation, Phd thesis, Università degli Studi di Milano, 2012.

[97] F. Tricoire, M. Romauch, K. F. Doerner, y R. F. Hartl, "Heuristics for the multi-period orienteering problem with multiple time windows," *Computers & Operations Research*, vol. 37, no. 2, pp. 351–367, 2010.

[98] D. Tuzun y L. I. Burke, "A two-phase tabu search approach to the location routing problem," *European Journal of Operational Research*, vol. 116, no. 1, pp. 87 – 99, 1999.

[99] P. Vansteenwegen, W. Souffriau, G. V. Berghe, y D. Van Oudheusden, "Iterated local search for the team orienteering problem with time windows," *Computers & Operations Research*, vol. 36, no. 12, pp. 3281–3290, 2009.

[100] P. Vansteenwegen, W. Souffriau, y D. Van Oudheusden, "The orienteering problem: A survey," *European Journal of Operational Research*, vol. 209, no. 1, pp. 1–10, 2011.

[101] E. Von Boventer, "The relationship between transportation costs and location rent in transportation problems," *Journal of Regional Science*, vol. 3, pp. 27–40, 1961.

[102] C. Watson-Gandy y P. Dohrn, "Depot location with van salesmen - a practical approach," *Omega*, vol. 1, no. 3, pp. 321–329, 1973.

[103] M. Webb, "Cost functions in the location of depots for multi-delivery journeys," *Operational Research*, vol. Quarterly, 19, pp. 311–320, 1968.

[104] V. F. Yu, S.-W. Lin, W. Lee, y C.-J. Ting, "A simulated annealing heuristic for the capacitated location routing problem," *Computers & Operations Research*, vol. 58, no. 2, pp. 288 – 299, 2010.