UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA CIVIL

PROBLEMA DE LOCALIZACIÓN Y RUTEO CON PICKUP AND DELIVERY

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MATEMÁTICO

TESIS PARA OPTAR AL TÍTULO DE MAGISTER EN CIENCIAS DE LA INGENIERÍA MENCIÓN TRANSPORTE

THOMAS CAPELLE NUÑO

PROFESORES GUÍA:
CRISTIÁN CORTÉS CARRILLO
PABLO REY

MIEMBROS DE LA COMISIÓN:
MARTÍN MATAMALA
LOUIS-MARTIN ROUSSEAU

SANTIAGO DE CHILE
AGOSTO 2012

Resumen

El objetivo de esta tesis es formular un modelo que integre el *Pickup and Delivery Problem* (PDP) y la localización óptima de los *depots* de distribución, el cual llamaremos *Problema de Localización y Ruteo con Pickup and Delivery* (PLRPDP). Este problema se plantea y tiene una gran variedad de aplicaciones en la industria, como por ejemplo en el courier, en el transporte de pasajeros, o en el transporte de alimentos perecibles, y por lo tanto es de suma importancia cuando existe un sistema de alta demanda con diferentes opciones en relación a la ubicación de los *depots*. Para modelar este problema se propone un esquema de generación de columnas, donde el problema maestro resultante es similar al del modelo propuesto por Berger et al. (2007), pero en este caso, el subproblema es un PDP. Para resolver este subproblema y poder generar columnas factibles para el problema maestro, se propone un algoritmo de *label-setting*, que resuelve el problema de camino más corto, con restricciones de *Pickup and Delivery* y ventanas de tiempo. Además se propone un conjunto de heurísticas para acelerar este proceso.

Para validar el modelo, se hace una implementación del esquema de generación de columnas, el cual se prueba en diferentes instancias, algunas ya existentes en la literatura actual, como también desarrolladas en este trabajo. Destacando dentro de estas últimas, las instancias clusterizadas y las de tipo corredor, las cuales por su geometría especial hacen que la localización de los *depots* sea de suma importancia. También se hace un análisis detallado de como los costos de apertura de los *depots* inciden en la solución óptima. Finalmente se presentan resultados del rendimiento de nuestra implementación para cada una de las instancias.

Glosario de definiciones y símbolos

I Conjunto de clientes

J Conjunto de *depots*

P_j Conjunto de rutas del *depot* j

P Conjunto de *pickups*

D Conjunto de *deliverys*

X_j Variable binaria asociada a la apertura de un *Depot*

Y_{jk} Variable binaria asociada a la ruta j del *Depot* k

y_{ij} Variable binaria del arco (i, j)

O_{ij} Variable del *branching* de los *pickups* i y j

a_i Cota inferior de la ventana de tiempo del nodo i

b_i Cota superior de la ventana de tiempo del nodo i

f_j Costos fijos de operación del *depot* j

c_k Costo de la ruta k

a_{ijk} Coeficiente binario, asociado al cliente i, *depot* j y ruta k

$conv(X)$ Envoltura convexa del conjunto X

LP Relajación lineal del problema (manera genérica de referirse a un problema lineal)

IP Problema entero o *integer program*, problema lineal cuya solución buscada es entera

Z_{IP} Valor de la función objetivo en el óptimo entero

Z_{LP} Valor de la función objetivo en el óptimo de la relajación lineal

π Variable dual asociada a la restricción de clientes

μ Variable dual asociada a la restricción de clientes-*depots*

Índice general

Capítulo 1

Introducción

1.1. Introducción

En un mundo globalizado en que las distancias y fronteras han sido de alguna forma abolidas, se ha generado un nuevo escenario en los procesos de transporte e intercambio de productos. Hoy, cuando las empresas requieren tomar decisiones para el logro más eficiente de sus objetivos, el diseño de un sistema logístico óptimo se vuelve cada vez más relevante.

Las decisiones relativas a la ubicación de los lugares de almacenamiento y las políticas de diseño de esos espacios, el manejo de la flota de vehículos y la entrega final de los productos, ya sean concretos o virtuales, requiere un manejo apropiado de todas las variables para obtener la máxima eficiencia posible y por lo tanto un ahorro significativo de recursos.

El rol de la *Investigación de Operaciones* en este campo es proveer a los planificadores con herramientas cuantitativas que les permitan tomar las mejores decisiones. En este contexto, esta tesis propone estudiar algunos de los problemas que aparecen en los diseños de sistemas logísticos, específicamente los relacionados con la localización de los puntos de almacenamiento (*depots*) y la estructura de las rutas a seguir por la flota. Metodológicamente se estudiará el problema desde el punto de vista de la programación matemática.

Una de las decisiones estratégicas más importante en el diseño de un sistema logístico es la ubicación de los *depots*. Encontrar las potenciales ubicaciones no es siempre una tarea fácil, y por lo general constituye el objetivo de un estudio previo. Una vez que el conjunto de ubicaciones posibles está determinado, los planificadores deben elegir un subconjunto de estas, que cumpla con las especificaciones del sistema y que a la vez sea óptimo para alguna función objetivo escogida. La naturaleza de esta decisión puede ser múltiple: minimizar costos de instalación, maximizar cobertura, optimizar alguna medida de bienestar o una combinación de las anteriores.

Si bien la ubicación es determinante, la eficiencia del sistema depende también de otros factores, entre ellos el ruteo de los vehículos está entre los más importantes. Por lo tanto los planificadores, además de determinar la ubicación óptima de los *depots*, deben determinar la cantidad de vehículos asignados para el manejo más adecuado del producto y el diseño de rutas más eficientes. La familia de problemas que estudian este tipo de decisiones es llamada Problemas de Ruteo de Vehículos, (Toth and Vigo, 2002).

Los investigadores has estudiado los problemas de localización y problemas de ruteo de vehículos durante varias decadas. De hecho, estas dos áreas son aún muy atractivas, ya que los modelos de programación matemática que generan son muy desafiantes. Ambos, el problema de localización y el de ruteo, caen en la mayoría de los casos, en el campo de la *Optimización Combinatorial* y bajo este contexto se estudiarán los problemas explorados en esta tesis.

Generalmente, si ambos problemas -localización y ruteo- aparecen juntos en un escenario dado, son estudiados y resueltos por separado. Esta tendencia en general se justifica por la gran dificultad del problema combinado. Dado que ambos problemas pertenecen a la clase $\mathcal{NP} - hard$, (en la mayoría de los casos) su combinación resulta en a un problema más complejo. Sin embargo, estudiar el problema de ruteo de vehículos fijando la localización de los *depots* con anterioridad, generalmente nos lleva a soluciones sub-óptimas.

A pesar de esta dificultad, los avances en recursos computacionales y algorítmicos actuales, además de resultados exactos obtenidos en resolver localización y ruteo simultáneamente Berger et al. (2007), proveen suficientes herramientas para plantear el problema a tratarse en esta tesis: el *Problema de Localización y Ruteo con Pickup and Delivery*. Este problema es una variante del problema clásico de Localización y Ruteo, pero considera que la estructura de las rutas de los vehículos respeta un esquema de *Pickup and Delivery*. Así, el primer problema que se debe estudiar es el *Pickup and Delivery Problem* (PDP), en que una empresa de transporte de pasajeros o carga, consta con una flota fija de m vehículos con capacidades fijas Q cada uno, (Savelsbergh and Sol, 1995). En este problema, cada cliente i tiene una ubicación de orígen i^+ y una ubicación de destino i^-. Analíticamente, el PDP resuelve la manera óptima de rutear los vehículos con el fin de satisfacer la demanda. Esto consiste en recoger la carga del cliente i en i^+ y transportala hasta su destino i^-.

Un ejemplo en la vida cotidiana son los sistemas de *transfers* al aeropuerto , donde cada cliente es recogido en su residencia, y llevado al aeropuerto o viceversa. Otro servicio del tipo PDP son los buses especiales para adultos mayores y discapacitados, los cuales recogen a sus pasajeros y los asisten en el viaje. A estos servicios en la literatura se les conoce como *Paratransit* o *Dial-a-Ride*, (Cordeau and Laporte, 2003).

En la literatura se encuentran algunas variantes a este problema, incluyendo restricciones de ventanas de tiempo en los orígenes o destinos. Por ejemplo Desrosiers et al. (1986), resuelve el problema para el caso de un solo vehículo, por medio de un algoritmo de programación dinámica. Por otro lado, Dumas et al. (1991) resuelve el problema de múltiples vehículos proponiendo un algoritmo exacto por medio de un esquema de generación de columnas. Más recientemente, Ropke and Cordeau (2009) proponen un algoritmo de *Branch and Cut and Price* que considera dos subproblemas para el algoritmo de generación de columnas: si las rutas son elementales o no lo son. Cotas inferiores son agregadas dinámicamente, lo que mejora la ejecución del esquema general de generación de columnas.

En los problemas de ruteo de vehículos, la función objetivo a optimizar depende del problema específico que se quiere resolver. En general es una combinación del costo de los usuarios y el costo de operación de los vehículos. Por lo general, el costo de los usuarios viene dado por los tiempos de viaje y espera. Cuando el problema es modelado con ventanas de tiempo de servicio

como restricción, el costo a los usuarios no es relevante en la función objetivo. Esto es así, ya que las ventanas de tiempo imponen en la solución óptima una calidad de servicio exigida por el usuario. Por lo tanto, la función objetivo suele estar asociada solamente al costo de ruteo. Cuando la demanda es conocida con anterioridad, la localización inicial de los vehículos que se usarán para servir la demanda de manera óptima, son variables relevantes en un contexto de planificación, en especial si la demanda no es homogénea a lo largo de la semana. Agregar la ubicación de los *depots* como variables, generaliza el modelo tradicional de ruteo de vehículos, en el cual los *depots* son fijos. Berger et al. (2007) plantea un modelo que integra tanto la localización de los *depots* como el ruteo de vehículos simultaneamente -Problema de Localización y Ruteo (PLR)-, agregando a la función objetivo un costo por abrir un depot además del costo de ruteo. Se propone un esquema de generación de columnas que es capaz de resolver instancias de hasta 10 *depots* candidatos y 100 clientes.

Con lo anterior, el objetivo central de esta tesis es formular un modelo que integre el *Pickup and Delivery Problem* y la localización óptima de los *depots*, el cual llamaremos *Problema de Localización y Ruteo con Pickup and Delivery* (PLRPD), además de proponer un esquema de solución eficiente.

La herramienta principal que se utilizará para resolver este problema es modelar el problema como un esquema de generación de columnas. Esto divide nuestro problema en dos partes, un problema maestro (similar al propuesto por Berger et al. (2007)), y un subproblema, el cual tiene como propósito alimentar al problema maestro con soluciones factibles. Para nuestro caso, el subproblema se ve modificado por las restricciones propias del PDP.

La primera parte de esta tesis consiste en una revisión bibliográfica, centrando el análisis en elementos de interés como son los problemas de ruteo de vehículos, prestando particular detalle a los problemas de PDP y sus respectivos métodos de solución. Además, se revisa el Problema de Localización y Ruteo, detallando la formulación expuesta en Berger et al. (2007), esto es particular interés, pues nuestro modelo es una extensión de este último.

Luego de la revisión bibliográfica, se hace un detallado análisis de la teoría necesaria para comprender los esquemas de generación de columnas, comenzando por los métodos de *branch and bound*, planos cortantes y *branch and price*. Esto permite dar las herramientas necesarias para poder generar un método de solución eficiente para el problemas propuesto en este trabajo.

El capítulo central de esta tésis presenta el Problema de Localización y Ruteo con *Pickup and Delivery*, presentando su formulación, algoritmo de solución y su validación por medio de resultados computacionales. La formulación presentada extiende la estudiada por Berger et al. (2007), agregando las restricciones propias del PDP. El algoritmo de solución utilizado es un esquema de generación de columnas en conjunto con un esquema de *branching* especificamente desarrollado para este problema. Además, se presentan tanto un algoritmo de programación dinámica y un con-

junto de heurísticas para resolver el problema de camino más corto, con restricciones de *Pickup and Delivery* y ventanas de tiempo. Finalmente, se muestran resultados computacionales para distintos tipos de instancias, se presta particular atención a los costos de apertura de *depots* y la geometría de estas. La tésis cierra con un capítulo de conlusiones y comentarios, donde además se proponen líneas de investigación futura.

Capítulo 2

Revision Bibliografica

La naturaleza combinada del Problema de Localización y Ruteo con *Pickup and Delivery* (PLRPD) lo relaciona con una variedad de otros problemas de la optimización combinatorial. En particular, estar familiarizado con los problemas de Ruteo de Vehículos, *Pickup and Delivery* y Localización es crucial para poder entender el PLRPD.

Este capítulo está estrucuturado en cinco partes, las cuales cubren el estado del arte en los temas abordados por esta tesis. La primera sección entrega una breve introducción de los problemas matemáticos asociados al ruteo de vehículos en general. En la segunda sección se discute el Problema de Ruteo de Vehículos, revisando algunas variantes importantes. En la tercera y cuarta parte, se presta particular atención a dos problemas: el Problema de *Pickup and Delivery* con Ventanas de Tiempo y el Problema de Localización y Ruteo. Se muestran sus formulaciones clásicas y algunos métodos de solución. Además se explica el algoritmo de *Branch and Price* usado para resolver el problema de Localizacíon y Ruteo expuesto en Berger et al. (2007).En la última parte se revisa la teoría sobre la descomposición de Dantzig and Wolfe (Dantzig and Wolfe, 1960) y el Método de Generación de columnas para resolver problemas enteros mixtos.

2.1. Optimización Combinatorial

La Optimización Combinatorial ha sido un área de estudio activa por varias décadas, desarrollando metodologías y algoritmos especializados para atacar un gran variedad de problemas. La mayoría de estos problemas, respecto de su complejidad computacional pertenecen a la clase $\mathcal{NP}$*-hard*, (Sipser, 2005). En este contexto de problemas, las soluciones aproximadas o *heurísticas* son bien recibidas, pues son capaces de obtener soluciones de muy buena calidad en un tiempo razonable.

Una de las técnicas más importantes en optimización combinatorial es la aplicación de la Programación Entera y la teoría de poliedros para describir los problemas. Las metodologías de optimización para resolver los problemas se pueden clasificar en exactas y aproximadas (o heurísticas). Dentro de las metodologías heurísticas destacamos las técnicas de relajación Lagrangeana y generación de columnas, pues son las más relacionadas con el problema a tratarse en esta tesis.

La relajación Lagrageana ha sido ampliamente usada en el contexto de programación entera

para obtener cotas ajustadas. Desde un punto de vista general, consiste en eliminar las restricciones difícles del problema e incorporarlas a la función objetivo, definida con los multiplicadores adecuados. Una buena revisión de relajación Lagrangeana para programación entera es Fischer (1981)

Una de las técnicas con mejores resultados para resolver problemas con un gran número de variables es la Generación de Columnas Dantzig and Wolfe (1960); Gilmore and Gomory (1961). La idea general es considerar un conjunto reducido de variables y sucesivamente ir generando nuevas columnas (variables) que puedan mejorar la solución. En ciertos casos, si se genera un cantidad suficiente de columnas, puede llegarse a la solución óptima del problema.

Lo métodos exactos más exitosos se basan en la enumeración implícita, donde las soluciones se van eliminando en bloques usando técnicas de acotamiento Johnson et al. (2000). Estos métodos varian entre ellos generalmente en la relajación usada (Lineal, Lagrangeana, etc...) para calcular las cotas. Además el modelo puede mejorarse considerando la inclusión de restricciones, preferentemente facetas de la envoltura convexa del conjunto de soluciones factibles. Este último es el principio detrás de los llamados métodos de *Branch and Cut*, introducido por Padberg and Rinaldi (1989). Otra interesante enumeración implícita es la que se obtiene al combinar la generación de columnas con los algoritmos para obtener soluciones enteras de *branch and bound* como es propuesto en Barnhart et al. (1998)

2.2. Problema de Ruteo de Vehículos

Uno de los problemas más estudiados en la optimización combinatorial es el Problema de Ruteo de Vehículos (VRP). Estos consisten en encontrar un conjunto de rutas óptimas que debe usar una flota de vehículos para servir a sus clientes. Han pasado más de 50 años desde que Dantzig y Ramser introdujeron el problema en 1959. Ellos describieron el problema a partir de una aplicación de un problema real que consistía en el abastecimiento de gasolineras, y propusieron la primera formulación matemática y el primer algoritmo de solución.

Formalmente el Problema de ruteo de Vehículos se enuncia de la siguiente manera:

Se dispone con una flota de m vehículos cada uno con capacidad Q, y se necesita despachar carga a n puntos de la ciudad partiendo cada vehículo desde algún depot. Cada uno de los puntos i tiene asociada una demanda d_i que debe ser satisfecha por un único vehículo.

Generalmente la red usada para el transporte se describe con un grafo, donde los arcos representan las secciones de ruta y los vértices corresponden a la ubicación de los clientes y *depots*. El grafo puede ser dirigido para representar la orientación de la ruta, además a cada arco se le asocia un costo que puede representar el tiempo de viaje, o costo del viaje a través de él, probablente dependiente del tipo de vehículo usado o el periódo del día en que es usado.
De acuerdo a los diferentes tipos de clientes, se desprenden variantes al problema original. Algunas de las variantes más importantes:

- Ruteo de Vehículos con Ventanas de Tiempo (VRPTW[1]): El cliente no está disponible nada

[1]Del inglés, VRP=*Vehicle Routing Problem with Time Windows*, VRPB=*Vehicle Routing Problem with Backhauls*,

más que un periodo determinado del día, lo que fuerza a que se vaya a buscar en este horario. Esto se logra agregando a cada nodo de demanda i un par de valores l_i, u_i que definen un intervalo en el cual debe ser atendido el nodo i.

- Ruteo de Vehículos con Carga / Descarga (VRPB[1]): El conjunto de clientes está separado en dos subconjuntos, V y L. El primer subconjuto, L, corresponde a los clientes que requieren una cantidad del producto a ser repartida. El segundo subconjunto, B, contiene los clientes donde una cantidad determinada de producto debe ser recogida.

- Problema de Recoger y Dejar pasajeros (PDP[1]): En este caso, cada cliente i tiene asociada un orígen i^+ y un destino i^-, donde debe ser recogido y dejado respectivamente. Si uno quiere, puede ver el VRP como un caso particular del PDP, pues es cuando todos los orígenes o destinos son reducidos a un solo punto.

Para más detalles, se sugiere la monografía Toth and Vigo (2002).

2.3. Pickup and Delivery Problem (PDP)

Para nuestro problema particular, la especificación del VRP que nos interesa trabajar es el *Pickup and Delivery Problem*. En el problema general, un conjunto de rutas se debe construir para satisfacer las demandas de transporte. Una flota de vehículos está disponible para operar las rutas, donde cada vehículo tiene una capacidad determinada, un punto de inicio y punto de término. Cada requerimiento especifica la carga a transportar, el punto de retiro y el destino. Cada carga debe ser transportada por un solo vehículo desde su orígen a su destino, sin transbordos. Generalmente si la carga a ser transportada son personas, suele llamarse al PDP como *Dial-a-Ride Problem* (DARP), una buena referencia y *review* del PVP y formulaciones es Cordeau and Laporte (2003).

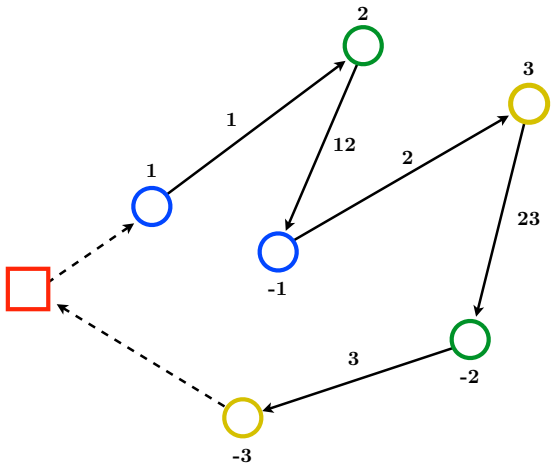


Figura 2.1: Instancia de PDP para 3 clientes y un solo vehículo

En Savelsbergh and Sol (1995) se formula el PDP como un problema de programación entera

PDP=*Pickup and Delivery Problem*

mixta (MIP)[2], el cual se presenta a continuación. Sea N el conjunto de clientes, cada cliente $i \in N$ tiene asociados un par de nodos (i^+, i^-) de origen y destino respectivamente. Denotaremos por N^+ al conjunto de todos los origenes y N^- al conjunto de todos los destinos.
Consideraremos que se cuenta con una flota de M vehículos. Cada vehículo $k \in M$ tiene capacidad $Q_k \in \mathbb{N}$, un lugar de partida k^+ y un lugar de guardado k^-. Definamos $M^+ := \{k^+ \mid k \in M\}$ como el conjunto de los lugares de partida y $M^- := \{k^- \mid k \in M\}$ como el de los lugares de llegada. Además, $V := N^+ \cup N^-$ y $W := M^+ \cup M^-$. Cada cliente debe ser recogido en i^+ por algún vehículo y dejado en i^- por este mismo vehículo (no se considera potenciales transbordos). El problema se plantea como un problema de optimización, que minimiza cierta función objetivo (distancia total recorrida, consumo de combustible, u otro costo general), sujeto a satisfacer la demanda de los clientes.
Para formular el PDP introducimos cuatro variables:

- $z_i^k (i \in N, k \in M)$ variable binaria, se le asigna el valor 1 si al cliente i se le asignó el vehículo k, 0 si no.
- $x_{ij}^k ((i,j) \in (V \cup W) \times (V \times W),\ k \in M)$ variable binaria, se le asigna el valor 1 si el vehículo k, viaja directamente desde i a j.
- $D_l (l \in V \cup W)$ variable real, representa el tiempo de partida en el nodo l.
- $y_l (l \in V \cup W)$ variable entera, representa la carga del vehículo que llega al nodo l.

[2]Del inglés, MIP=*Mixed Integer Programming*

La formulación es la siguiente:

$$
\begin{aligned}
\text{mín} \quad & f(x) && && (2.1)\\
s.a: \quad & \sum_{k\in M} z_i^k = 1 && \forall i \in N && (2.2)\\
& \sum_{j\in V\cup W} x_{lj}^k = \sum_{j\in V\cup W} x_{jl}^k = z_i^k && \forall i \in N, l \in \{i^+, i^-\}, k \in M && (2.3)\\
& \sum_{j\in N^+\cup N^-\cup\{k^-\}} x_{k^+j}^k = 1 && \forall k \in M && (2.4)\\
& \sum_{i\in N^+\cup N^-\cup\{k^-\}} x_{ik^-}^k = 1 && \forall k \in M && (2.5)\\
& D_{k^+} = 0 && \forall k \in M && (2.6)\\
& D_{i^+} \leq D_{i^-} && \forall i \in N && (2.7)\\
& x_{ij}^k = 1 \Rightarrow D_i + t_{ij} \leq D_j && \forall i, j \in V\cup W, k \in M && (2.8)\\
& y_{k^+} = 0 && \forall k \in M && (2.9)\\
& y_l \leq \sum_{k\in M} Q_k z_i^k && \forall i \in N, l \in \{i^+, i^-\} && (2.10)\\
& x_{ij}^k = 1 \Rightarrow y_i + q_i = y_j && \forall i, j \in V\cup W, k \in M && (2.11)\\
& x_{ij}^k \in \{0,1\} && \forall i, j \in V\cup W, k \in M && (2.12)\\
& z_i^k \in \{0,1\} && \forall i \in N, k \in M && (2.13)\\
& D_i \geq 0 && \forall i \in V\cup W && (2.14)\\
& y_i \geq 10 && \forall i \in V\cup W && (2.15)
\end{aligned}
$$

Donde la ecuación (2.2) asegura que cada requerimiento sea asignado solo a un vehículo. La restriccion (2.3) asegura que un vehículo solo abandona la ubicación l si es orígen o destino de algun requerimiento asociado a este vehículo. Las restricciones (2.4) y (2.5) nos aseguran que cada vehículo empiece y termine en el lugar correcto. Las restricciones (2.6), (2.7), (2.8) y (2.14) en conjuntos forman las restricciones de precedencia. Por otro lado (3.19), (2.10), (2.11) y (2.15) forman las restricciones de capacidad. Cabe notar, que las restricciones (2.8) y (2.11) no son lineales, pero se pueden reescribir como restricciones lineales usando la técnica del *big M*.

Esta formulación es bastante general; por ejemplo es posible agregar ventanas de tiempo en los cuales pueden ser recogidos y dejados los requerimientos, generando intervalos para cada nodo $l \in V \cup W$, de la forma $[a_l, b_l]$, como en Dumas et al. (1991).

Cabe notar que el problema del VRP es un caso particular del PDP, considerando $|N^-| = 1$, e igual al depot.

2.3.1. Métodos de Solución

Los métodos para resolver este problema se pueden clasificar en métodos exactos y heurísticos.

Métodos Exactos

Psaraftis (1980) y Desrosiers et al. (1986) plantean algoritmos de programación dinámica para resolver el PDP. Desrosiers et al. (1986) resuelven el problema minimizando la distancia total recorrida por los vehículos, considerando ventanas de tiempo duras, mientras que Psaraftis (1980) considera en la función objetivo la insatisfacción de los usuarios y no considera ventanas de tiempo. Estos algoritmos definen estados del sistema, los cuales van actualizando a medida que recorren el conjunto de rutas factibles. Las rutas se comparan y se establece un criterio para identificar cuales sub-rutas son mejores que otras. Con esto, se puede establecer que estados son mejores que otros y por lo tanto las rutas dominadas son eliminadas, finalmente el algoritmo termina con la solución óptima.

El algoritmo usado por Desrosiers et al. (1986) resuelve el Problema de Recoger y Dejar Pasajeros con un solo vehículo, minimizando la distancia total recorrida. Se definen los estados (S, i) los cuales son estados factibles, si existe una ruta que pasa por todos los nodos de S y termina en el nodo i. El algoritmo comienza en la iteración $k = 0$ con el vehículo en el depot y el estado $(\{0\}, 0)$, y en cada iteración trata de alcanzar a los vecinos de cada estado, los cuales son posibles respetando las restricciones de precedencia (recoger a un pasajero antes de ir a dejarlo), ventanas de tiempo y capacidad del vehículo.

Dumas et al. (1991) resuelven el PDP con una flota de múltiples vehículos. Para resolver el problema proponen un esquema de generación de columnas en el cual las columnas corresponden a las rutas de cada vehículo. Como sub-problema obtienen una versión relajada del PDP en la cual no es necesario recoger a todos los clientes. Para resolver esto, desarrollan un método de programación dinámica similar al usado en Desrosiers et al. (1986).

Ruland and Rodin (1997) formulan el PDP como un MIP y estudian la estructura poliedral del polítopo que se genera. Entregan como resultado 4 tipos de desigualdades válidas para este problema basándose en el *Traveling Salesman Problem* (TSP) con restricciones de precedencia. Mediante el método de *branch and bound* que proponen son capaces de resolver instancias de hasta 15 clientes.

Más recientemente, Ropke and Cordeau (2009) proponen un nuevo esquema de *brand and cut and price* donde las cotas inferiores son calculadas resolviendo la relajación lineal del problema maestro por medio de un esquema de generación de columnas. Dos subproblemas son considerados en la generación de columnas: en el primero las rutas son elementales (no contienen ciclos) y otro donde las rutas no son elementales. Resuelven a optimalidad instancias de hasta 75 clientes. Además logran resolver instancias de hasta 500 clientes heurísticamente.

Métodos Heurísticos

Los métodos heurísticos para el PDP se remontan a los años 80. De interés particular para esta tesis son los métodos presentados por Sexton and Choi (1986); Xu et al. (2003). La excelente revisión Parragh et al. (2008), (sobre todo la Parte II),recorre la gran mayoría de las heurísticas para el problema estático.

En Sexton and Choi (1986) usan la descomposición de Benders para resolver el problema de *Pickup and Delivery* para el caso estático aproximadamente. Las soluciones iniciales son construídas usando una heurística espacio-tiempo. El mejoramiento incremental de las rutas se basa en la descomposición de Benders. Como las ventanas de tiempo consideradas son suaves, la función objetivo carga con el total de tiempo de operación al igual que con la violación de las ventanas de tiempo.

También usando la descomposición de Benders para resolver el problema, en Cortés et al. (2009) se propone el problema clásico de PDP, pero incorporando la posibilidad de hacer transbordos. En esta variante los pasajeros pueden cambiar de un vehículo a otro en ciertos nodos de transferencia.

En el trabajo de Xu et al. (2003), se propone una heurística basada en la generación de columnas para el caso multi-vehículo. Consideran varias restricciones adicionales como, ventanas de tiempo en el orígen y destino, restricciones de carga, cargas heterogéneas, flota heterogénea como también el horario de los conductores. El problema maestro de la generación de columnas puede ser resuelto por un paquete comercial de problemas lineales. Los subproblemas resultantes son resueltos por dos heurísticas, llamadas *merge* y *two-phase*, las cuales juntan los viajes y usan eliminación glotona con inserción de las demandas.

2.4. Problema de Localización y Ruteo (PLR)

En esta sección se introduce el Problema de Localización y Ruteo (PLR). Para este problema, se asume que un conjunto de clientes con sus respectivas demandas es conocido. Además se cuenta con un conjunto de potenciales ubicación para emplazar los *depots*, cada uno con un costo fijo de operación. También se conoce el costo de viajar entre cualquier par de puntos de la red. El objetivo de este problema es determinar la cantidad y la ubicación de los *depots* a abrirse, además de diseñar las rutas desde cada *depot* seleccionado hacia los clientes, respetando las restricciones y minimizando el costo total.

Donde se localizan los *depots* para guardar la mercadería o los vehículos es casi tan importante como las rutas que se usan para servir a los clientes. En la práctica, el costos de habilitar un *depot* en general es alto, y debe ubicarse estratégicamente para minimizar los costos de ruteo (Webb, 1968). En las modelaciones generales, los vehículos salen de un *depot* ya fijado y visitan a sus clientes en algún orden, volviendo de nuevo al *depot* inicial (*in-out model*). Para capturar el efecto de optimizar en las rutas tanto como en la ubicación espacial de los *depots*, es necesario resolver el ruteo y la localización simultaneamente. A este problema se le conoce como Problema de Localización y Ruteo. La Figura 2.2 representa una instancia de PLR con 15 clientes (blanco) y 7 (rojo). En este caso, todos los *depots* están en uso. La Figura 2.3 es otra solución de la misma instancia, con un elevado costo de los *depots*, resultando en menos *depots* utilizados. A continuación se expone la

formulación expuesta en Berger et al. (2007), desde donde se construyen los modelos desarrollados con el contexto de esta tesis. En este trabajo se establece una formulación que busca minimizar el costo total, simultáneamente eligiendo un subconjunto de *depots* disponibles y construyendo un conjunto de rutas factibles que satisfaga:

- Las demandas de los clientes son satisfechas sin exceder las capacidades de los vehículos y los *depots*.
- El número de vehículos, largo de rutas y duración de las rutas no exceden los máximos permitidos.
- Cada ruta empieza y termina en el mismo *depot*.

Las condiciones anteriores son clásicas del VRP.

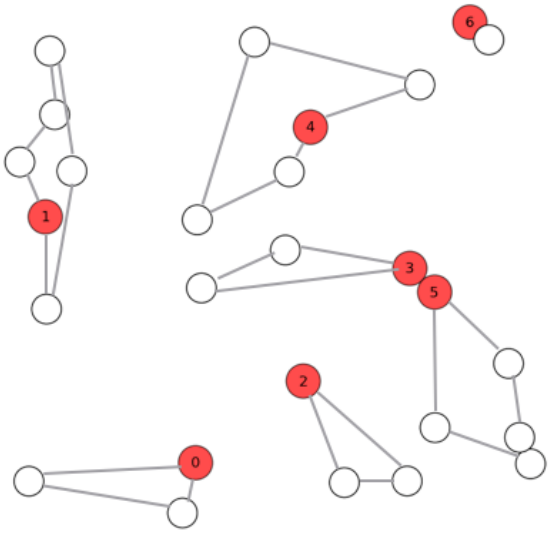


Figura 2.2: Solución de una instancia del PLR con todos los *depots* en uso

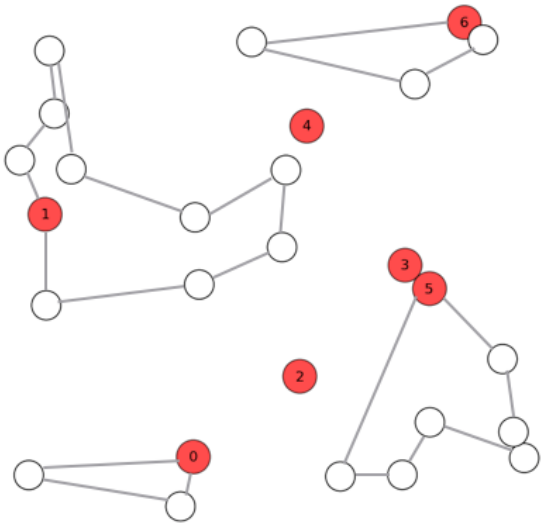


Figura 2.3: Solución de una instancia del PLR con los *depots* 0,1,5 y 6 en uso

2.4.1. Formulación

En esta formulación las variables de decisión corresponden a las rutas. Denotaremos por I al conjunto de clientes y J el conjunto de *depots*. El grafo dirigido en el que se trabaja es: $G = (N, A)$, con $N = I \cup J$ y $A = N \times N$. Se denota por d_{ij} la distancia desde el nodo i al nodo j. Las rutas asociadas al depot j, que visitan a uno o más clientes las denotaremos por el índice k. El conjunto de rutas factibles asociadas a un *depot* j se denota P_j, adicionalmente el costo de una ruta $k \in P_j$ es la suma de los costos de todos los arcos que usa. Además se supone que el costo de atravesar un arco (i, j) proporcional a la distancia d_{ij}. Los parámetros y variables que se definen para formular el modelo son:
Parámetros:

- a_{ijk} vale 1 si la ruta k asociada al *depot* j visita al cliente i, 0 si no.
- c_{jk} es el costo de la ruta k asociada al *depot* j.
- f_j son los costos fijos de operación del *depot* j.
- α factor de peso relativo entre costo de instalación y costo de ruteo.

Variables:[3]

- X_j vale 1 si el *depot* j es usada, 0 si no.
- Y_{jk} vale 1 si la ruta k asociada al *depot* j es usada, 0 si no.

[3]Se utiliza letras mayúsculas para denotar las variables en esta sección, respetando la notación del trabajo original de Berger et al. (2007)

Formulación PLR Berger et al. (2007)

$$\text{mín} \quad \alpha \sum_{j \in J} f_j X_j + \sum_{j \in J} \sum_{k \in P_j} c_{jk} Y_{jk} \tag{2.16}$$

$$s.a: \quad \sum_{j \in J} \sum_{k \in P_j} a_{ijk} Y_{jk} = 1 \qquad \forall i \in I \tag{2.17}$$

$$X_j - Y_{jk} \geq 0 \qquad \forall j \in J, \forall k \in P_j \tag{2.18}$$

$$X_j \in \{0, 1\} \qquad \forall j \in J \tag{2.19}$$

$$Y_{jk} \in \{0, 1\} \qquad \forall j \in J, \forall k \in P_j \tag{2.20}$$

Aquí la función objetivo (2.16) tiene dos partes, una que pretende minimizar los costos de los *depots* usados, y un segundo término que minimiza el largo de las rutas elegidas. La ecuación (2.17) verifica que cada cliente sea servido por exactamente una de las rutas elegidas. Por su parte (2.18) garantiza que el *depot* j sea usado si la ruta k asociada a esa *depot* es usada.
Para resolver el problema efectivamente por *branch-and-price*, es necesario tener una cota inferior eficiente. Desafortunadamente la relajación lineal del problema entrega una cota muy débil. Típicamente observamos que la solución de la relajación lineal asigna valores fraccionarios suficientes como para satisfacer que $X_j - Y_{jk} \geq 0$. Como resultado, el valor objetivo de la relajación lineal es solo un pequeño porcentaje del óptimo en la solución entera. Para corregir esto, se considera el siguiente conjunto de desigualdades:

$$X_j - \sum_{k \in P_j} a_{ijk} Y_{jk} \geq 0 \qquad \forall i \in I, \ \forall j \in J \tag{2.21}$$

En una solución entera factible, cada cliente $i \in I$ está en exactamente una ruta, por lo tanto $\sum k \in P_j a_{ijk} Y_{jk}$ es igual a 1 para algun *depot* j y 0 para todas las otras. Para cualquier *depot* j tal que $\sum k \in P_j a_{ijk} Y_{jk} = 1$, X_j será 1, lo que corresponde a que la apertura de los *depots* donde se seleccionan rutas. Cabe notar que estas implican a las de (2.18), por lo tanto, no consideraremos la formulación con la restricción (2.21) en vez de (2.18).

Proposición 2.4.1. *Las restricciones* (2.21) *implican a las restricciones* (2.18)*, independientemente de las restricciones de integralidad.*

Dem: Consideremos un *depot* $j \in J$ y una ruta $\hat{k} \in P_j$, sea i un cliente en la ruta $\hat{k}$. Entonces, como $Y_{jk} \geq 0$ y $a_{ijk} \geq 0$, se tiene que:

$$X_j \geq \sum_{k \in P_j} a_{ijk} Y_{jk} \geq a_{ijk} Y_{j\hat{k}}$$

Así la formulación resultante tiene un número polinomial de restricciones, dado que $\sum_{j \in J} |P_j|$ restricciones fueron eliminadas y $|J| \cdot |I|$ fueron agregadas. Además, la cota inferior otorgada por la relajación lineal del modelo con la restricción (2.21) es mayor o igual que la cota otorgada por el modelo original, pues la región factible del LP con la restricción (2.21) está contenida en la región

factible original, y los coeficientes de la función objetivo no cambian □

2.5. Métodos de Optimización Entera

Resolver problemas de optimización $\mathcal{NP}$-*hard* a optimalidad es un tópico que ha desafiado a los investigadores desde el inicio de la computación. Grandes avances se han logrado en las últimas décadas, pero para muchos problemas solo instancias pequeñas pueden resolverse. Los problemas de ruteo de vehículos han probado ser díficiles de resolver, siendo los problemas de gran tamaño todavía un desafío.

En este sección revisamos algunos de los métodos usados para resolver problemas de optimización modelados por medio de la programación entera. Revisaremos la parte teórica necesaria para explicar los métodos de *branch and cut* y *branch and price*.

En lo que sigue, asumimos que un problema entero lineal de minimización (ILP) se resolverá, el cual puede escribirse:

$$z_{IP} = \min_{x \in \mathbb{Z}^n} \{c^\top x \mid Ax \geq b\}$$

donde $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ y $b \in \mathbb{R}^m$.

2.6. Branch and Bound

Los procedimientos de *branch and bound* tienen como objetivo encontrar el valor mínimo de una función f en un conjunto P, mediante dos herramientas. La primera consiste en partir el conjunto P en pedazos $\{P_1, P_2, ...P_N\}$ cuya unión sea P y usar que $\min_P f = \min\{v_1, v_2, ..., v_N\}$, donde $v_i = \min_{P_i} f$ (proceso de ramificación o *branching*). Hay que notar que aplicar este proceso recursivamente define una estructura de árbol, donde cada hoja es un subconjunto de P.

La segunda herramienta consiste en calcular cotas inferiores y superiores de f sobre cada subconjunto P_i. (proceso de acotamiento o *bounding*).

2.6.1. Ejemplo

$$\begin{array}{ll} \min & c^\top x \\ sa & x \in \Omega \end{array}$$

donde $c = (-2, -3)$ y $P = \{x \in \mathbb{Z}_+^2 : \frac{2}{9}x_1 + \frac{1}{4}x_2 \leq 1, \frac{1}{7}x_1 + \frac{1}{3}x_2 \leq 1\}$.

Consideremos la relajación lineal del problema, es decir, olvidando las restricciones de variables enteras:

$$\begin{array}{ll} \min & c^\top x \\ sa & x \in P_0 \end{array}$$

con $\Omega_0 = \{x \in \mathbb{R}_+^2 : \frac{2}{9}x_1 + \frac{1}{4}x_2 \leq 1, \frac{1}{7}x_1 + \frac{1}{3}x_2 \leq 1\}$. La solución de este problema es $x^* = (2.17, 2.07)$, con el valor objetivo $p^* = -10.56$. Tomemos los dos subproblemas tales que en

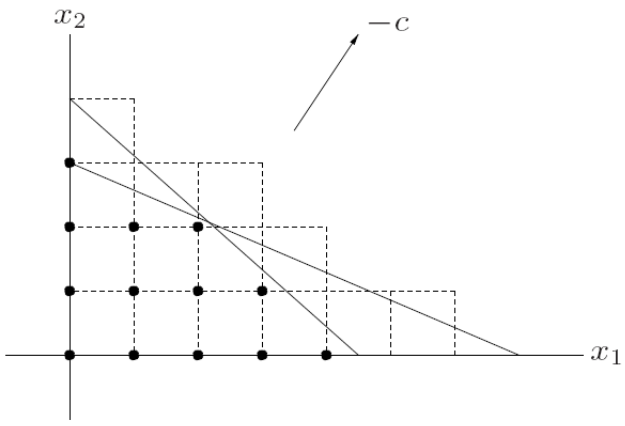


Figura 2.4: Representación gráfica del problema Ω

el primero $x_1 \leq \lfloor 2.17 \rfloor = 2$ y en el segundo $x_1 \geq \lceil 2.17 \rceil = 3$.

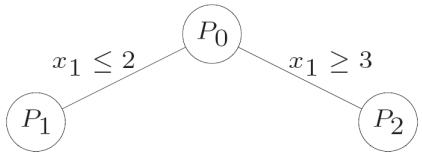
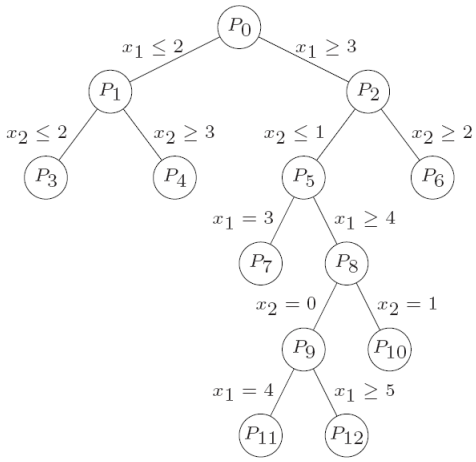


Figura 2.5: Representación gráfica del problema P_0 correspondiente a la primera ramificación

Podemos repetir esto en cada rama y hacer un árbol. Cada vez que la solución es entera o infactible terminamos la rama del árbol. El árbol del ejemplo, con la respectiva tabla de soluciones relajadas se ve como en la figura 2.6.



	$x^\star$	$p^\star$
P_0	$(2.17, 2.07)$	-10.56
P_1	$(2.00, 2.14)$	-10.43
P_2	$(3.00, 1.33)$	-10.00
P_3	$(2.00, 2.00)$	-10.00
P_4	$(0.00, 3.00)$	-9.00
P_5	$(3.38, 1.00)$	-9.75
P_6		$+\infty$
P_7	$(3.00, 1.00)$	-9.00
P_8	$(4.00, 0.44)$	-9.33
P_9	$(4.50, 0.00)$	-9.00
P_{10}		$+\infty$
P_{11}	$(4.00, 0.00)$	-8.00
P_{12}		$+\infty$

Figura 2.6: Árbol de P

Considerando que los problemas relajados permiten soluciones con valores menores a las soluciones permitidas cuando las variables son enteras, podemos concluir lo siguiente:

- P_2: El valor optimal de

$$\begin{array}{ll} \text{mín} & c^t x \\ sa & x \in P,\ x_1 \geq 3 \end{array}$$

  es mayor o igual que -10.

- P_3: La solución de

$$\begin{array}{ll} \text{mín} & c^t x \\ sa & x \in P,\ x_1 \leq 2,\ x_2 \leq 2 \end{array}$$

  es entera y es (2,2).

- P_6: El problema

$$\begin{array}{ll} \text{mín} & c^t x \\ sa & x \in P,\ x_1 \leq 3,\ x_2 \geq 2 \end{array}$$

  es infactible.

De la enumeración, podemos concluir que

- P_0: El valor óptimo de

$$\begin{array}{ll} \text{mín} & c^t x \\ sa & x \in P \end{array}$$

  es mayor o igual que -10.56.

- P_1: El valor óptimo de

$$\begin{array}{ll} \text{mín} & c^t x \\ sa & x \in P,\ x_1 \leq 2 \end{array}$$

es mayor o igual que -10.43.

- P_2: El valor óptimo de

$$\begin{array}{ll} \text{mín} & c^t x \\ sa & x \in P,\ x_1 \geq 3 \end{array}$$

es mayor o igual que -10.

- P_3: El valor óptimo de

$$\begin{array}{ll} \text{mín} & c^t x \\ sa & x \in P,\ x_1 \leq 2,\ x_2 \leq 2 \end{array}$$

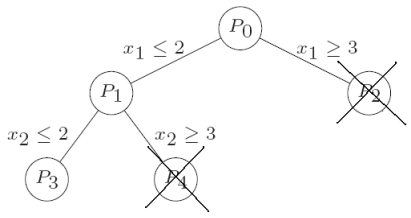
ES -10 (pues la solución es (2,2) entera).

- P_4: El valor óptimo de

$$\begin{array}{ll} \text{mín} & c^t x \\ sa & x \in P,\ x_1 \leq 2,\ x_2 \geq 3 \end{array}$$

es mayor que -9

Sólo observando estos 5 subproblemas se concluye que el óptimo es (2,2). Esto es porque P_3 muestra que es posible encontrar una solución entera con valor -10, y en la rama P_2 las soluciones son mayores o iguales que -10, es decir, es posible eliminar esta rama (antes de construirla, por supuesto). En P_4 las soluciones son $\geq$ -9, es decir, también se pueden eliminar las ramas que comienzan en P_4. El algoritmo seguido es:



	$x^\star$	$p^\star$
P_0	$(2.17, 2.07)$	-10.56
P_1	$(2.00, 2.14)$	-10.43
P_2	$(3.00, 1.33)$	-10.00
P_3	$(2.00, 2.00)$	-10.00
P_4	$(0.00, 3.00)$	-9.00

Este método es llamado Algoritmo Branch and Bound:

2.6.2. Detalles del Algoritmo

0) Denotemos como L la lista de problemas relajados a resolver. Sea $L = \{(P_0)\}$, $\bar{z} = +\infty$.

1)
   - Si $L = \phi$ (vacío) TERMINAR con las siguientes conclusiones:
     Si $\bar{z} = +\infty$ el problema original (de programación entera) es infactible, en caso contrario, el óptimo es $(\bar{x}, \bar{z})$.
   - Si la lista no est· vacía, escoger un problema (nodo del árbol) de la lista. Sea (P) el problema seleccionado.

2) Si (P) es infactible, eliminarlo de la lista (equivale a cortar la rama del árbol de soluciones que parten o hubiesen partido desde (P)). Ir a 1.

3) Si (P) es factible, sea z' el valor óptimo y x' su solución óptima.

   - Si $z' \geq \bar{z}$ ELIMINAR el problema (P) de la lista (equivale a cortar la rama del árbol de soluciones que parten o hubiesen partido desde (P)). Ir a 1 (Poda por Cota).
   - Si $z' < \bar{z}$ y x' cumple integralidad, entonces $\bar{z} \leftarrow z'$ y $\bar{x} \leftarrow x'$, eliminar (P) de la lista e ir a 1 (Poda por solución entera).
   - En caso contrario, sea x_k una componente de x' que no es entera, ramificar el problema (P) creando dos problemas (P^+) y (P^-) que se agregan a la lista L (el problema (P) se saca de la lista). El problema (P^-) es el mismo que (P), pero con la restricción $x_k \leq \lfloor x'_k \rfloor$, mientras que el problema (P^+) es el mismo que (P), pero con la restricción $x_k \geq \lceil x'_k \rceil$. Ir al paso 1).

2.6.3. Planos Cortantes

Como se muestra la sección anterior, para poder construir un esquema de *branch and bound* eficiente para resolver z_{IP}, es necesario una cota inferior de z_{IP}, idealmente una que se puede calcular eficientemente. Una manera es resolver la relajación lineal de z_{IP}

$$z_{LP} = \min_{x \in \mathbb{R}^n} \{c^T x \mid Ax \geq b\} \tag{2.22}$$

como z_{LP} minimiza sobre un conjunto que contiene a $\mathbb{Z}^n$, es claramente una cota inferior para z_{IP}. Definamos $\mathcal{Q} = \{x \in \mathbb{R}^n \mid Ax \geq b\}$ el poliedro de soluciones factibles de z_{LP}. Se puede expresar el conjunto de soluciones factibles de z_{IP} como $\mathcal{F} = \mathcal{Q} \cap \mathbb{Z}^n$. Llamaremos $conv()$ a la envoltura covexa. Como $conv(\mathcal{F})$ es un poliedro y que los puntos extremos de $conv(X)$ siempre están en X:

$$z_{IP} = \min_{x \in \mathbb{Z}^n} \{c^T x \mid Ax \geq b\} = \min_{x \in \mathcal{F}} \{c^T x\} = \min_{x \in conv(\mathcal{F})} \{c^T x\}$$

Cabe notar que el hecho que $P = conv(\mathcal{F})$ sea un poliedro hace que en teoría z_{IP} se puede resolver como un LP, pero en la práctica, esto no da una manera de resolver z_{IP} pues por ejemplo no se

conocen las desigualdades que definen P, y aunque se conocieran podrían ser una cantidad muy grande.

Sin embargo, es posible tratar de aproximar P, que es lo que se hace en una esquema de *branch and bound* para poder obtener mejores cotas inferiores. Para ilustrar esto, se considera el siguiente problema entero lineal como ejemplo:

$$z_{IP} = \text{mín} \quad -2x_1 + 3x_2 \tag{2.23}$$

$$s.a: \quad 2x_1 + x_2 \geq 8 \tag{2.24}$$

$$-3x_1 + x_2 \geq -12 \tag{2.25}$$

$$x_1 - 4x_2 \geq -20 \tag{2.26}$$

$$-x_1 + 2x_2 \geq -1 \tag{2.27}$$

$$x_1, x_2 \in \mathbb{Z} \tag{2.28}$$

El poliedro $\mathcal{Q}$ en este caso está definido por las ecuaciones 2.24-2.27 y

$$x_1, x_2 \in \mathbb{R}^n \tag{2.29}$$

El poliedro $\mathcal{Q}$ es ilustrado en la Figura 2.7(izquierda). Se obtiene que $z_{LP} = -3{,}8$ y que el punto optimal en $\mathcal{Q}$ es $(x_1^*, x_2^*) = (4.6, 1.8)$ (ver Figura 2.8, derecha). El conjunto $\mathcal{F} = \mathcal{Q} \cap \mathbb{Z}^2$ y el poliedro $conv(\mathcal{F})$ son fáciles de obtener e ilustrados en Figura 2.8 izquierda

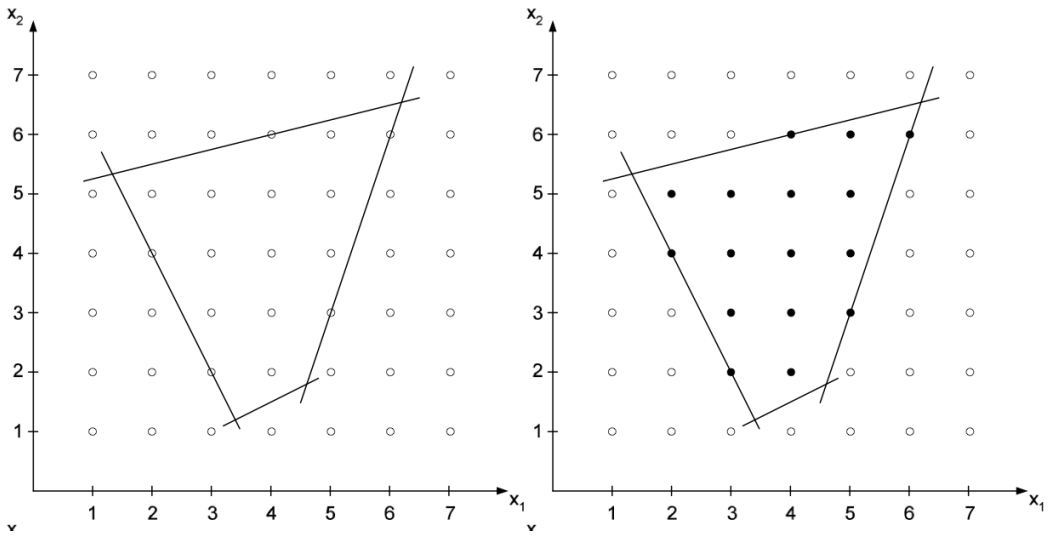


Figura 2.7: La figura de la izquierda ilustra el poliedro $\mathcal{Q}$ definido por las ecuaciones 2.24-2.27. La figura de la derecha muestra $\mathcal{F} = \mathcal{Q} \cap \mathbb{Z}^2$, los elementos de $\mathcal{F}$ son los puntos negros.

En la Figura 2.9 agregamos la desigualdad válida:

$$-2x_1 + x_2 \geq -7 \tag{2.30}$$

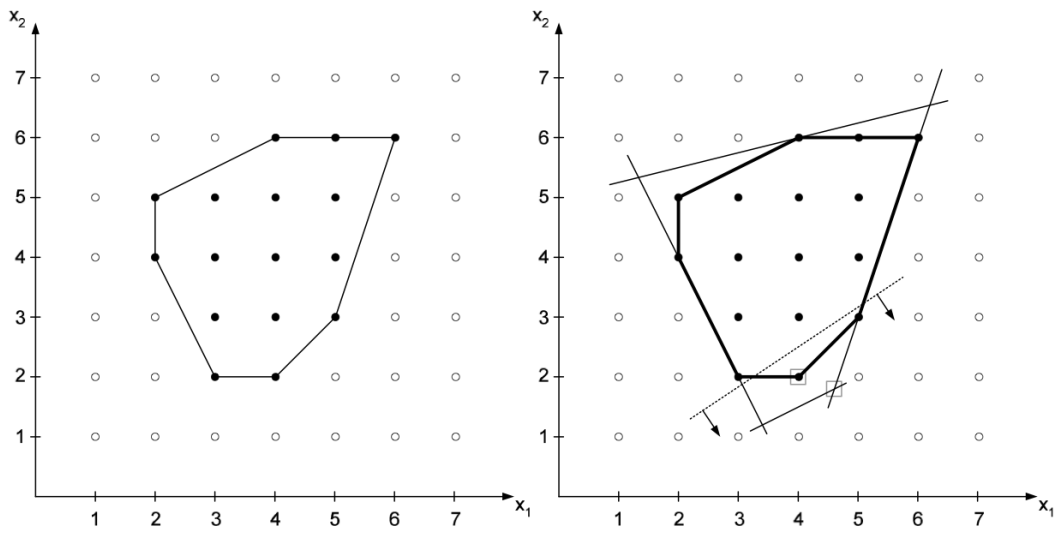


Figura 2.8: La figura en el izquierda ilustra el poliedro $conv(\mathcal{F})$. En la derecha se ilustran $\mathcal{Q}$, $\mathcal{F}$ y $conv(\mathcal{F})$. La figura también muestra la función objetivo (línea punteada) y las soluciones óptimas del LP e IP (rectángulos grises)

Al agregar esta nueva desigualdad se separa el viejo óptimo del LP $x^* = (4.6, 1.8)$ del conjunto $\mathcal{F}$. El nuevo óptimo del LP es $x^* = (4\frac{1}{3}, 1\frac{2}{3})$ con valor objetivo $-3\frac{2}{3}$, que es un poco mejor cota inferior.

Agregando las desigualdades:

$$-x_1 + x_2 \geq -2 \tag{2.31}$$

$$x_2 \geq 2 \tag{2.32}$$

Separando nuevamente x^* de $\mathcal{F}$, la relajación del LP entrega la solución al problema entero $x^* = (4, 2)$, $z_{IP} = z_{LP} = -2$, lo cual se ilustra en la Figura 2.9 (derecha). Este ejemplo muestra que no es necesario tener una descripción completa de la envoltura convexa de $\mathcal{F}$ para obtener la solución del IP usando la relajación LP. Una descripción de la envoltura convexa cerca del punto optimal de IP es suficiente (en este caso las desigualdades 2.31 y 2.32). Además se observa que la desigualdad 2.30 es redundante debido a 2.25-2.32. Las desigualdades 2.31 y 2.32 son desigualdades de facetas de $conv(\mathcal{F})$ mientras que 2.30 define una cara de $conv(\mathcal{F})$. Las desigualdades que definen facetas son las desigualdades válidas más fuertes. La definición de caras y facetas (*faces and facets*) se puede encontrar en Wolsey (1998)

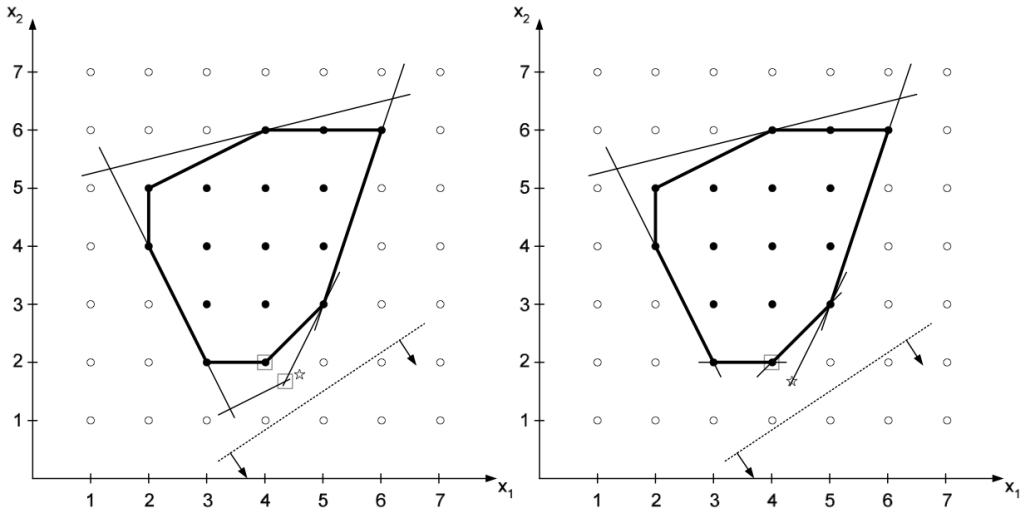


Figura 2.9: La figura en el izquierda la desigualdad 2.30 ha sido añadida a 2.24-2.27 llevandonos a una mejor cota inferior La solución optima del LP está marcada con un cuadrado, mientras que la antigua solución óoptima del LP está marcada con una estrella. En la figura de la derecha

2.7. Algortimo de Planos Cortantes

La desigualdades válidas nos dan un punto de partida para mejorar las cotas inferiores y resolver los problemas enteros. Nuestro punto de partida es la relajación lineal 2.22. Consideremos la definición de $\mathcal{Q}$ de la sección anterior, y escribamos:

$$z_{LP} = \text{mín}\{c^T x \mid x \in \mathcal{Q}\} \tag{2.33}$$

El algoritmo de los planos cortantes, se basa en ir agregando desigualdades válidas a $\mathcal{Q}$ para que se aproxime a $\mathcal{F}$ lo más posible. Los pasos a seguir son básicamente:

1) Definir $t = 0$, $\mathcal{Q}_t = \mathcal{Q}$

2) Resolver el LP

$$z_t = \text{mín}\{c^T x \mid x \in \mathcal{Q}_t\}$$

3) Llamar x^* a la solución óptima

4) Si x^* es entera, encontramos el óptimo del IP, si no, pasar al paso siguiente

5) Buscar una desigualdad $\pi_t x \geq \pi_t^0$ que separe x^* de $\mathcal{F}$

6) Poner $\mathcal{Q}_{t+1} = \mathcal{Q}_t \cap \{x \in \mathbb{R}^n \mid \pi_t x \geq \pi_t^0\}$, $t = t + 1$

7) Ir al paso 2

El paso 5 es el más difícil del algoritmo, sin embargo, existe el algoritmo de *Gomory* para generar planos separadores, según se aprecia en Nemhauser and Wolsey (1988). Este algoritmo provee una manera de generar una desigualdad válida violada, y se puede probar que con este método se

encuentra la solución óptima del IP en una cantidad finita de pasos. La convergencia es muy lenta, y en general no es usado en la práctica.

Alternativamente se puede pueden obtener desigualdades, usando técnicas particulares para el problema específico que se desea resolver. (Por ejemplo para el TSP se pueden usar las desigualdades de tipo *comb*). Cuando no es posible conseguir mas desigualdades violadas el algoritmo termina, y debe ser resuelto por un *branch and bound* usando como cota inferior $z_{LP} = \text{mín}\{c^T x \mid x \in \mathcal{Q}_t\}$

2.7.1. Branch and Cut

El algoritmo de *branch and cut* extiende la idea de planos cortantes de la sección anterior. La idea consiste en generar desigualdades válidas violadas a lo largo de todo el árbol de *branch and bound* y no sólo en el nodo raíz. Las desigualdades son elegidas típicamente de un conjunto preseleccionado y en cada nodo se debe elegir entre mejorar la cota inferior o procesar el nodo lo más rápido posible. En la mayoría de los casos uno dejará de generar desigualdades válidas en un nodo si la mejora de la cota inferior ha sido pequeña después de un número de iteraciones. En tal caso, puede ser mejor hacer la ramificación y luego tratar de generar más desigualdades en los nodos hijos.

Los algoritmos de *branch-and-cut* han sido usados satisfactoriamente en la optimización entera, el caso más notable es la solución del TSP en Applegate et al. (2003) y más recientemente Karaoglan et al. (2012)

2.8. Introduccion al Branch and Price

En las secciones anteriores se discuten técnicas para obtener cotas inferiores mejores y como usar estas cotas en un esquema de *branch-and-bound*. En esta parte discutiremos la descomposición de Dantzig-Wolfe para problemas enteros e investigaremos como se puede usar dentro de un esquema de *branch and bound*. La descomposición de Dantzig-Wolfe fue originalmente introducida para problemas lineales en Dantzig and Wolfe (1960).

El esquema de *branch and price* se basa en dos conceptos. El primero, consiste en la descomposición del problema transformando el problema original desde su formulación compacta, en una formulación que contiene muchas columnas, pero típicamente menos filas que la formulación original. En la literatura a esta transformación se le llama formulación extensa (*extensive formulation*).

El segundo concepto, es la *generación de columnas*. Para poder resolver esta nueva formulación extensa, lo que uno hace, no es generar el modelo completo, pues este tiene típicamente una cantidad muy grande de variables -la cantidad de variables generalmente es exponencial en el tamaño del problema-. Lo que se propone es generar las columnas dinámicamente. Cuando la cota inferior es calculada dinámicamente usando generación de columnas dentro de un esquema de *branch and bound*, el algoritmo es llamado *branch and price*.. Agregar una busqueda de *branch and bound* sobre un problema lineal tratado con generación de columnas puede parecer bastante sencillo, pero tiene sus dificultades. Un ejemplo de estas dificultades mantener la compatibilidad de las reglas del *branch* con el subproblema.

En esta sección solo se hace una breve introducción a las técnicas de generaición de columnas y *branch and price*. Más información se puede encontrar en Wolsey (1998); Barnhart et al. (1998); Feillet (2010). Esta sección está basada en el trabajo de Ropke (2005) y el libro de Wosley (Wolsey, 1998).

2.8.1. Descomposición

Llamaremos problema maestro (PM) a:

$$z_{PM} = \text{mín} \quad c^\top x \tag{2.34}$$
$$s.a: \quad Ax = b \in \mathbb{R}^m \tag{2.35}$$
$$Dx \leq d \tag{2.36}$$
$$x \in \mathbb{N}^n \tag{2.37}$$

Donde A es una matriz de $m_A \times n$ y D es una matriz de $m_D \times n$ a elementos reales, $b \in \mathbb{R}^{m_A}$ y $d \in \mathbb{R}^{m_D}$ son vectores. Además supondremos que $\{x \in \mathbb{R}_+{}^n \quad | \quad Dx \leq d\}$ es un poliedro acotado. (ver Wolsey (1998); Nemhauser and Wolsey (1988), para el caso no acotado). Entonces el conjunto $Q = \{x \in \mathbb{N}^n \quad | \quad Dx \leq d\}$ contiene una cantidad finita de elementos$\{x_i\}_{i\in\Omega}$ y se puede reescribir el conjunto Q como:

$$Q = \left\{ x = \sum_{i\in\Omega} \lambda_i x_i \quad | \quad x_i \in Q, \lambda_i \in \{0,1\}, \sum_{i\in\Omega} \lambda_i = 1 \right\} \tag{2.38}$$

Reemplazando x en la formulación de 2.34-2.37 se obtiene la llamada *formulación extendida*:

$$z_{PM} = \text{mín} \quad c^\top \left(\sum_{i\in\Omega} \lambda_i q_i \right) \tag{2.39}$$
$$s.a: \quad A \left(\sum_{i\in\Omega} \lambda_i q_i \right) = b \tag{2.40}$$
$$\sum_{i\in\Omega} \lambda_i = 1 \tag{2.41}$$
$$\lambda_i \in \{0,1\} \tag{2.42}$$

$$z_{PM} = \text{mín} \quad \sum_{i \in \Omega} \lambda_i (c^\top q_i) \tag{2.43}$$

$$s.a: \quad \sum_{i \in \Omega} \lambda_i (A q_i) = b \tag{2.44}$$

$$\sum_{i \in \Omega} \lambda_i = 1 \tag{2.45}$$

$$\lambda_i \in \{0, 1\} \tag{2.46}$$

Definamos $c_i = c^\top q_i$ y $a_i = A q_i$ para cada $i \in \Omega$, este par $\begin{pmatrix} c^t q_i \\ A q_i \end{pmatrix}$ es lo que se llama una columna para cada $q_i \in Q$. Reemplazando:

$$z_{PM} = \text{mín} \quad \sum_{i=1}^{M} c_i \lambda_i \tag{2.47}$$

$$s.a: \quad \sum_{i \in \Omega} a_i \lambda_i = b \tag{2.48}$$

$$\sum_{i \in \Omega} \lambda_i = 1 \tag{2.49}$$

$$\lambda_i \in \{0, 1\} \tag{2.50}$$

Esta formulación contiene menos filas (restricciones) pero típicamente contiene muchas más columnas (variables). El real beneficio de la formulación por extensión es que la relajación lineal que se obtiene en general es mejor que la relajación lineal de la formulación compacta del problema. Para poder ejemplificar lo anterior, consideremos los poliedros $Q_c = \{x \in \mathbb{R}^n_+ \mid Ax = b, Dx \leq d)$ y $Q_e = \{x \in \mathbb{R}^n_+ \mid Ax = b, x \in conv(Q)\}$. Q_c es el poliedro donde optimiza la relajación lineal de la formulación compacta, mientras que Q_e es el poliedro donde optimiza la relajación lineal de la formulación por extensión. Claramente $Q_e \subset Q_c$ pues,

$$conv(Q) = conv(\{x \in \mathbb{N}^n \mid Dx \leq d\}) \subset \{x \in \mathbb{R}^n_+ \mid Dx \leq d\}$$

Si $conv(Q) = \{x \in \mathbb{R}^n_+ \mid Dx \leq d\}$ entonces ambas formulaciones entregan la misma cota inferior. Este es el caso cuando todos los puntos extremales de $\{x \in \mathbb{R}^n_+ \mid Dx \leq d\}$ son enteros. Toda la construcción anterior tiene una gran desventaja, es necesario conocer el conjunto Q. En la sección siguiente se mostrará que no es necesario tener una definición explícita de Q.

2.8.2. Generación de Columnas

En esta sección se revisa la teoría de los esquemas de generación de columnas para problemas enteros. Esta técnica es ampliamente usada para resolver problemas lineales de gran tamaño Dantzig and Wolfe (1960); Desrosiers et al. (1984); Desrochers and Soumis (1989); Dumas et al. (1991); Savelsbergh and Sol (1998); Barnhart et al. (1998); Ropke and Cordeau (2005); Berger et al. (2007). En términos coloquiales, la generación de columnas es una técnica que permite comenzar con una parte pequeña y manejable del problema (específicamente solo algunas de las variables), resolver esta parte, analizar la solución parcial y determinar como agrandar el problema (ver que variables agregar) para luego resolver el problema aumentado. El proceso se repite hasta que alcanza una solución satisfactoria para el problema total.

Formalmente, la generación de columnas es una forma de resolver problemas lineales, añadiendo columnas (correspondiente a las variables) las cuales se obtienen resolviendo el *pricing problem* del método Simplex. Esto se hace pues, generar una columna en la formulación primal de simplex de un problema lineal corresponde a añadir una restricción en su formulación dual. En la formulación dual de un problema lineal dado, se puede pensar que generar una columna es como aplicar un método de planos cortantes. Es en este contexto, que muchos investigadores han observado que la generación de columnas es una técnica poderosa para resolver una gran variedad de problemas industriales a optimalidad o bien obtener solución muy cercanas al óptimo.

La generación de columnas descansa en el hecho que el método simplex no requiere de tener acceso a todas las variables del problema simultáneamente. De hecho, el método puede empezar a trabajar contando solo con la base (un conjunto particular de las variables) y luego usar el costo reducido para determinar cuales de las otras variables se deben acceder de ser necesario.

Consideremos la relajación lineal del problema maestro, la cual se denotará por LPM:

$$z_{LPM} = \text{mín} \quad \sum_{i \in \Omega} c_i \lambda_i \tag{2.51}$$

$$s.a: \quad \sum_{i \in \Omega} a_i \lambda_i = b \tag{2.52}$$

$$\sum_{i \in \Omega} \lambda_i = 1 \tag{2.53}$$

$$\lambda_i \geq 0 \tag{2.54}$$

Así la relajación lineal considera combinaciones lineales convexas de elementos de Q. Se considera mirar un subconjunto más pequeño de índices$\Omega' \subset \Omega$, donde $|\Omega'| << |\Omega|$. El subconjunto de índices

Ω' debe ser escogido de tal manera que el problema:

$$z_{RPM} = \text{mín} \quad \sum_{i \in \Omega'} c_i \lambda_i \tag{2.55}$$

$$s.a: \quad \sum_{i \in \Omega'} a_i \lambda_i = b \tag{2.56}$$

$$\sum_{i \in \Omega'} \lambda_i = 1 \tag{2.57}$$

$$\lambda_i \geq 0 \tag{2.58}$$

tenga solución factible. Llamaremos RPM a la restricción a Ω' del problema maestro relajado. Para poder proseguir, es necesario recordar como el método simplex resuelve un problema lineal como:

$$\text{mín}\, c^\top x \tag{2.59}$$

sujeto a

$$Mx = b \tag{2.60}$$

$$x \in \mathbb{R}^n_+ \tag{2.61}$$

El método simplex mantiene una *solución básica factible* que como el nombre lo implica es factible para el LP, pero no necesariamente óptima. En cada iteración una columna es elegida para entrar a la base. Para que esta columna mejore la solución básica actual, tiene que tener costo reducido negativo c_i^π

$$c_i^\pi = c_i - \pi M_i$$

donde π es la variable dual actual asociada a 2.60 y M_i es la i-ésima columna de la matriz M. La manera standard de elegir la columna que entra a la base es elegir la columna que resuelve:

$$\min_{i \in \{1,\ldots,n\}} \{c_i - \pi M_i\}$$

cuando no se encuentran columnas con costo reducido negativo, el método de simplex ha encontrado la solución óptima de 2.59-2.61.

Volviendo a la generación de columnas del problema relajado y restringido (RPM) se puede usar el vector dual π para ver si más columnas pueden ser agregadas a Ω'. Para esto, basta calcular el costo reducido de todas las columnas de $\Omega \setminus \Omega'$. Si alguna de estas columnas tiene costo reducido negativo, basta agregarla a Ω' y volver a resolver el nuevo RPM 2.55-2.58. Si no existen columnas con costo reducido negativo en $\Omega \setminus \Omega'$, entonces la solución de RPM 2.55-2.58 es también óptima para LPM 2.51-2.54.

Hay que notar que la descripción anterior requiere que se conozcan todas las columnas a_i para todos los $i \in \Omega$, por lo menos es necesario contar con una función que sea capaz de generarlas. Esto no es práctico para muchos problemas. Lo que se hace generalmente, es contar con una función llamada *oráculo* que dado el vector dual π, es capaz de entregar una columna con costo reducido

negativo o establecer si ésta no existe. El oráculo resuelve el problema:

$$\min_{q \in Q} \{c^T q - \pi A q\} \tag{2.62}$$

o alternativamente

$$\min \{c^T x - \pi A x\} \tag{2.63}$$

sujeto a

$$x \in \mathbb{N}^n, \quad Dx \leq d \tag{2.64}$$

El problema 2.62 es lo que se llama el subproblema o *pricing problem*. Este problema por lo general es díficil por si mismo. Por ejemplo, el problema de ruteo de vehículos con ventanas de tiempo (VRPTW) se descompone en un problema maestro que tiene como *pricing problem* un problema de camino más corto elemental con restricciones de capacidad y ventanas de tiempo, el cual es *NP-hard*, ver Desrosiers et al. (1984).

2.8.3. Branch and Price

En la sección anterior revisamos como obtener cotas inferiores para la formulación de un LP. A veces la solución a la relajación LP resulta ser entera, y por lo tanto el problema entero IP original es resuelto, pero en general, la solución del LP será racional. El *branch and price* resuelve este problema, obteniendo una solución entera cuando las solución del LP es fracional.

La manera intuitiva sería usar *branch and bound*, donde la ramificación (*branching*) se hace sobre las variables λ_i, para cada variables λ_i que no sea entera. Por lo tanto, para cada $i \in \Omega$ donde λ_i es fraccional, se crean dos ramas, una donde $\lambda_i = 1$ y otra donde $\lambda_i = 0$. El problema de hacer esto, es que genera arboles de *branch and bound* muy desbalanceados, pues la rama donde $\lambda_i = 0$ generalmente no cambia mucho el problema (estamos excluyendo una columna de un millón de columnas). El otro problema, es que crea dificultades en el subproblema. Cuando la variable $\lambda_i = 0$, el subproblema no puede generar la columna i. Para evitar estos problemas, se usan esquemas de ramificación que sean compatibles con el subproblema. Esto por lo general se logra, ramificando en las variables de la formulación compacta. Para el PDPTW por ejemplo, en Dumas et al. (1991) proponen una esquema de ramificación en los arcos, lo cual es fácil de transferir al subproblema.

Capítulo 3

Problema de Localización y Ruteo con Pickup and Delivery

El Problema de Localización y Ruteo con *Pickup and Delivery* combina en un solo modelo, el problema de localización de los *depots* y el problema de rutear los vehículos de manera óptima respetando las restricciones propias de un problema de *Pickup and Delivery*.

El modelo expuesto en la sección anterior es flexible, puesto que su formulación basada en rutas, no explicita las restricciones propias y permite ser extendido al problema que se enfrenta en esta tesis.

3.1. Formulación del PLR-PDP

El Problema de Localización y Ruteo con *Pickup and Delivery* es una extensión del Problema de Localización y Ruteo, y nuestra formulación está basada en la formulación introducida en 2.4.1. El objetivo del problema consiste en elegir un conjunto de *depots* y construir las rutas asociadas, de tal manera de minimizar el costo de instalación además del costo de ruteo, respetando que los clientes deben ser servidos por una única ruta. Hay que notar, que cuando nos referimos al cliente i, esto consiste en recogerlo en su *pickup* i^+ y depositarlo en su *delivery* i^-. Se considera entonces la formulación expuesta en 2.16:

Formulación PLR-PDP

$$\text{mín} \quad \alpha \sum_{j \in J} f_j X_j + \sum_{j \in J} \sum_{k \in P_j} c_{jk} Y_{jk} \tag{3.1}$$

$$s.a: \quad \sum_{j \in J} \sum_{k \in P_j} a_{ijk} Y_{jk} = 1 \qquad \forall i \in I \tag{3.2}$$

$$X_j - \sum_{k \in P_j} a_{ijk} Y_{jk} \geq 0 \qquad \forall i \in I,\ \forall j \in J \tag{3.3}$$

$$X_j \in \{0, 1\} \qquad \forall j \in J \tag{3.4}$$

$$Y_{jk} \in \{0, 1\} \qquad \forall j \in J, \forall k \in P_j \tag{3.5}$$

La diferencia con respecto a la formulación de 2.4.1, es que en este caso, las rutas asociadas a los *depots* satisfacen las restricciones del PDPTW. Esto no se refleja directamente en la formulación del problema maestro del PLR-PDP, si no que en la construcción de las rutas propiamente tal. El conjunto P_j consiste en las rutas factibles asociadas al *depot* j. Para cada ruta $k \in P_j$ se satisfacen las siguientes condiciones:

- La ruta k comienza y termina en el *depot* j
- Si el cliente i es servido por la ruta k, entonces la precedencia es respetada, esto es; i^+ (el *pickup*) aparece antes en la ruta que i^- (el *delivery*)
- Si el cliente i es servido por la ruta k, entonces sus ventanas de tiempo en i^+ e i^- son respetadas
- La carga del vehículo (q) es siempre menor a su capacidad (Q)

Consideramos $P = \{i^+ \ : \ i \in I\}$ al conjunto de los *pickups* y $D = \{i^- \ : \ i \in I\}$ el conjunto de los *deliverys*. Para cada nodo $l \in P \cup D$, denotaremos su ventana de tiempo $[a_l, b_l]$ y q_l la demanda por ser transportada en cada nodo. (notar que $q_{i^+} = -q_{i^-}$).

La formulación del PLR-PDP de esta manera, contiene un número exponencial de variables (Y_{jk}). Por lo tanto, para instancias de tamaño práctico, enumerar las rutas no es algo posible. En vez de esto, usaremos un esquema de *branch and price* para resolver instancias de nuestro problema.

3.2. Branch and Price para el PLR-PDP

El algoritmo de *branch and price* que se propone está basado en el trabajo de Berger et al. (2007), el cual es usado para resolver el PLR. Este método ha sido utilizado satisfactoriamente para resolver problema de transporte del tipo *Scheduling* (asignación de tareas con restricciones de horarios) (Desrochers and Soumis, 1989), ruteo de vehículos (Desrosiers et al., 1984) y Pickup and Delivery con ventanas de tiempo (Dumas et al., 1991; Savelsbergh and Sol, 1998). Se sugiere revisar el trabajo de (Barnhart et al., 1998) en este se presenta una discusión de la metodología general del *branch and price*, además en el trabajo de Feillet (2010) se expone como aplicar está metodología para resolver problema de ruteo de vehículos.

Para generar un esquema de *branch and price* para el PLR-PDP usaremos la misma notación introducida en el capítulo 2.4.1. Así 3.1-3.5 definen el problema maestro PM. Llamaremos LPM a la relajación lineal del problema maestro:

LPM

$$\text{mín} \quad \alpha \sum_{j \in J} f_j X_j + \sum_{j \in J} \sum_{k \in P_j} c_{jk} Y_{jk} \tag{3.6}$$

$$s.a: \quad \sum_{j \in J} \sum_{k \in P_j} a_{ijk} Y_{jk} = 1 \qquad \forall i \in I \tag{3.7}$$

$$X_j - \sum_{k \in P_j} a_{ijk} Y_{jk} \geq 0 \qquad \forall i \in I,\ \forall j \in J \tag{3.8}$$

$$X_j \geq 0 \qquad \forall j \in J \tag{3.9}$$

$$Y_{jk} \geq 0 \qquad \forall j \in J, \forall k \in P_j \tag{3.10}$$

en el cual las variables de localización X_j y las variables de ruteo Y_{jk} ya no son binarias, sino que son reales positivas.

El conjunto rutas P_j para cada *depot* no es conocido; por lo tanto, para el esquema de *branch and price* se resuelve una restricción de LPM con un conjunto $P'_j \subset P_j$ para cada *depot* $j \in J$. Cada uno de los conjuntos P'_j es generado independientemente, asegurando la factibilidad del problema total en todo momento. Se denotará RPM a la restricción del LPM, donde sólo un subconjunto de las variables de rutas son consideradas (todas las de localización son siempre consideradas)

RPM

$$\text{mín} \quad \alpha \sum_{j \in J} f_j X_j + \sum_{j \in J} \sum_{k \in P'_j} c_{jk} Y_{jk} \tag{3.11}$$

$$s.a: \quad \sum_{j \in J} \sum_{k \in P'_j} a_{ijk} Y_{jk} = 1 \qquad \forall i \in I \tag{3.12}$$

$$X_j - \sum_{k \in P'_j} a_{ijk} Y_{jk} \geq 0 \qquad \forall i \in I,\ \forall j \in J \tag{3.13}$$

$$X_j \geq 0 \qquad \forall j \in J \tag{3.14}$$

$$Y_{jk} \geq 0 \qquad \forall j \in J, \forall k \in P'_j \tag{3.15}$$

La generación de P'_j es realizada en el subproblema o *pricing problem*.

3.2.1. Formulación del Subproblema

Para el esquema de *branch and price* es necesario identificar los subproblemas asociados al problema maestro PLR-PDP expuesto en 3.1-3.5. En este caso, para cada *depot* $j \in J$ se debe resolver un subproblema independiente, el cual consiste en identificar rutas del conjunto P'_j. Cada uno de estos subproblemas es un problema de *pickup and delivery* con restricciones de capacidad y ventanas de tiempo muy similar al expuesto en Dumas et al. (1991)

De la teoría de programación lineal se sabe que en una solución óptima el costo reducido de cada variable debe ser no-negativo, ver 2.8.2. Por lo tanto, si el *pricing problem* identifica rutas factibles con costo reducido negativo, se añaden al RPM y se reoptimiza, (Figura 3.1). Este proceso se detiene cuando no es posible identificar rutas con costo reducido negativo. Esto nos asegura que

la solución óptima del RPM es óptima para la relajación lineal original del problema.

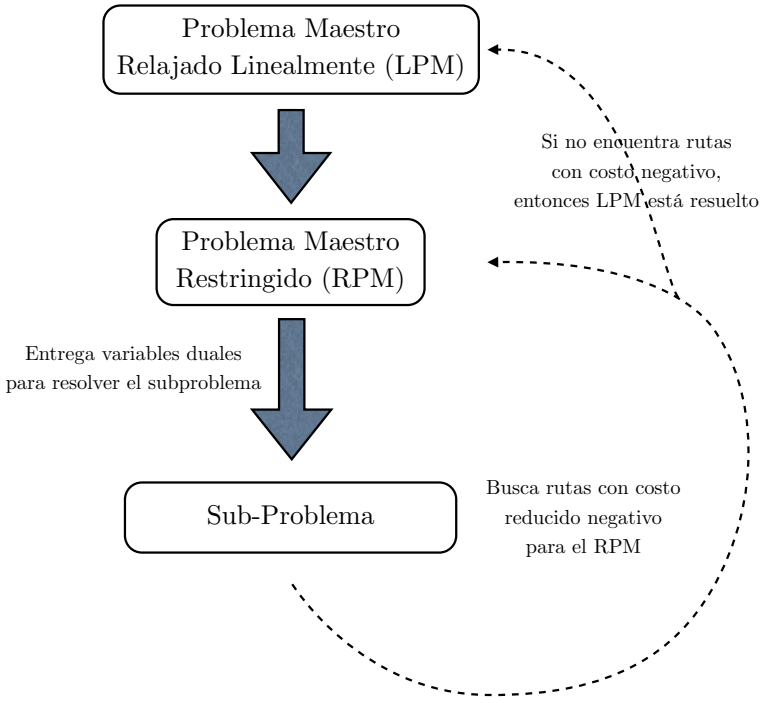


Figura 3.1: Metodología del esquema de generación de columnas.

Para formular el *pricing problem*, se usa la teoría de dualidad para problemas lineales para obtener el costo reducido de una ruta k asociada a un depot j como:

$$\hat{c}_{jk} := c_{jk} - \sum_{i \in I} a_{ijk}(\pi_i - \mu_{ij}) \tag{3.16}$$

donde el vector π son las variables duales asociadas a las restricciones 3.7 y la matriz μ está asociada a las restricciones de 3.8. Para poder obtener las rutas con costo reducido negativo, es necesario transferir la información de las variables duales a los arcos.Como el subproblema es independiente para cada depot, se construye una red auxiliar para cada depot j. Así, para cada *depot* j se construye un grafo $D_j = (V_j, A_j)$, el cual consiste en la red completa borrando los otros *depots*. Los conjuntos P y D son de los *pickups* y *deliverys* respectivamente, ambos con cardinal $|P| = |D| = N$. El conjunto de vértices $V_j = P \cup D \cup \{0, 2N+1\}$. El nodo 0 representa al *depot* j como punto de partida y el nodo $2N+1$ también al *depot* j pero como punto de llegada. El conjunto de arcos $A_j = [\{0\} \times P] \cup [(P \cup D) \times (P \cup D)] \cup [D \times \{2N+1\}]$ como muestra la figura 3.2.

Así, se construye el costo reducido de un arco (l, m) como:

$$\hat{d}_{lm} := \begin{cases} d_{l,m} - \pi_m + \mu_{m,j} & \text{si } l \in V_j, m \in P \\ d_{l,m} & \text{si } l \in V_j, m \in D \cup \{2N+1\} \end{cases} \tag{3.17}$$

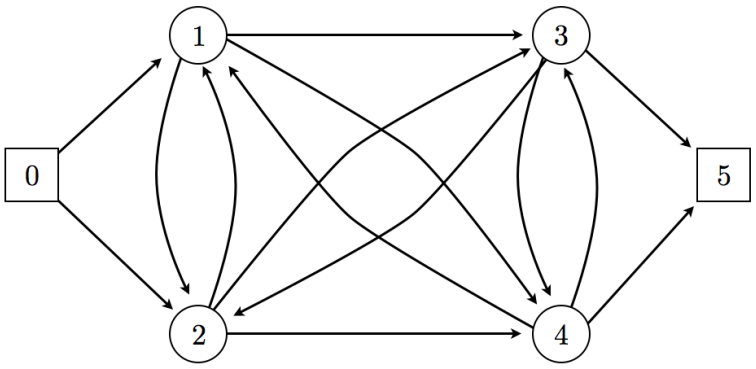


Figura 3.2: Red Auxiliar para un *depot* j, con 2 clientes.

Lo que ocurre en 3.17 es que los arcos que llegán a un *pickup* se llevan todo el valor de las variables duales. Además la definición es consecuente con la expresada en 3.16

Una vez que los costos han sido transferidos a los arcos mediante 3.17, el subproblema para cada *depot* j consiste en resolver un problema de camino más corto sobre la red auxiliar respectiva, con restricciones de precedencia, ventanas de tiempo y capacidad. Lo cual en la literatura se conoce como *Elementary Shortest Path with Pickup and Delivery and Time Windows Constraints* ESPPDTWC, ver Ropke and Cordeau (2009).

3.2.2. Solución del Subproblema

Los problemas de camino más corto son comunes en los esquemas de generación de columnas de problemas de ruteo de vehículos. Para resolver estos problemas, generalmente se usan versiones modificadas de los algortimos típicos como *Dijkstra* o *Bellman*, (Dijkstra, 1959; Bellman, 1958). La idea general es asociar a cada ruta parcial una etiqueta o *label*, y extender estos *labels* comprobando la factibilidad de los recursos (tiempo, carga en el vehículo), hasta que el mejor camino posible haya sido encontrado. Reglas de dominancia se usan para comparar las rutas parciales que llegan a un mismo nodo, descartandose algunas de estas. Para cada nodo del grafo se debe mantener una gran cantidad de *labels*, dado que cada comparación toma en cosideración los recursos consumidos. Lamentablemente este algoritmo ya no es polinomial, -de hecho es *NP*-hard. (Dror, 1994)

Es común evitar calcular el conjunto de rutas óptimas terminando el subproblema prematuramente, cuando un conjunto de columnas con costo reducido negativo es conocida. De hecho, encontrar el camino de costo mínimo es solo necesario en la última iteración del algoritmo, cuando hay que probar que no existen rutas con costo reducido negativo.

Para obtener la solución a cada *pricing problem* se resuelve un problema de camino más corto sobre esta red auxiliar, por medio de un algoritmo de tipo *label-setting*. Se denota por z_j^* al costo reducido de la solución del *pricing problem* para el depot j. Si $z_j^* \geq 0 \quad \forall j \in J$, la relajación lineal del problema (LPM) ha sido resuelto. De lo contrario, para cada $j \in J$ tal que $z_j^* < 0$ se añade la correspondiente columna al RPM y reoptimizamos.

3.3. Algoritmo de Label-Setting para camino más corto con recursos

Se considera un grafo dirigido $G = (V, A)$, donde V es el conjunto de nodos y A es el conjunto de arcos, se denota por s al nodo inicial y t al nodo sumidero. Se asume que no hay arcos entrantes al nodo s y no hay arcos salientes del nodo t. Además, se conocen los recursos $R_1, ..., R_n$ del problema. Algunos ejemplos de estos recursos son: tiempo de viaje, capacidad del vehículo, combustible, trasbordos. Además, al atravesar un arco, consume tales recursos. Llamemos $r_{ij} \in \mathbb{R}^n$ denota el consumo de los recursos para el arco $(i, j) \in A$. Además para cada nodo $i \in V$ existen cotas $a^i \in \mathbb{R}^n$ (inferior) y $b^i \in \mathbb{R}^n$ (superior) para los recursos.

Una Etiqueta o *Label* consiste en tres elementos: un nodo, los recursos consumidos hasta este nodo y un puntero al nodo que lo precede. Se denotará a un *label* por $L = (i, R, p)$, que corresponde a un camino que comienza en el nodo s y termina en el nodo i con un consumo de recursos que viene dado por el vector $R \in \mathbb{R}^n$. El puntero p apunta al *Label* padre, que es necesario para reconstruir recursivamente el camino que une s con i. Un pseudo-código para resolver el problema de camino más corto con recursos viene ejemplificado en el Algoritmo 1.

Algorithm 1 Pseudo-código para algoritmo de *label setting*

1: **procedure** Camino_más_corto(G, A, s, t)
2: $H = \{(s, (a_1^s, ..., a_n^s), none)\}$
3: **while** $H \neq \emptyset$ **do**
4: L = sacar_primero(H)
5: i = nodo(L)
6: **if** L no está dominado por ningun label de $\mathcal{L}_i$ **then**
7: $\mathcal{L}_i$=$\mathcal{L}_i \cup \{L\}$
8: **for** (i, j) saliendo de i **do**
9: **if** la extensión a j es factible **then**
10: agregar la extensión a H
11: **return** mejor camino de $\mathcal{L}_t$

En la línea 2 del algoritmo se inicia la lista de *labels* no procesados H con la etiqueta $(s, (a_1^s, ..., a_n^s), none)$ correspondiente al nodo inicial s. Los recursos consumidos se establecen como la cota inferior de cada uno $(a_1^s, ..., a_n^s)$ para el nodo s. $\mathcal{L}_i$ denota la lista de labels procesados en el nodo i (caminos terminando en i). En la línea 4, la función sacar_primero entrega el primer nodo de la lista H y lo quita de la lista H. En la linea 5 se entrega el nodo terminal del label L y en la 6 verifica si este puede ser eliminado. Si este no puede ser eliminado, se guarda en la lista de nodos procesados para el nodo $\mathcal{L}_i$, además se extiende este label a todos los nodos vecinos de i, así para cada arco (i, j) se entrega un nuevo label (j, R', L), donde R' se calcula a partir de los recursos ya consumidos hasta i dados por R, y las cotas inferiores y superiores en el nodo j. En la linea 9 se revisa si la extensión es factible, si todos los recursos están dentro de las cotas del nodo j. Cada una de estas extensiones es agregada a la lista H en la linea 10. Para terminar, el algoritmo devuelve la mejor ruta desde el conjunto $\mathcal{L}_t$. Para poder garantizar la finitud del algoritmo se deben verificar ciertas condiciones. Primero es necesario definir un orden entre los *labels* que determine que uno es mayor que otro. El

proceso de extender los *labels* debe generar un *label* más grande con respecto al orden y además debe existir un *label* maximal con respecto al orden. Si la función sacar_primero() entrega el más pequeño de los *labels* de H, nunca se procesará un *label* más de una vez, pues extenderlo genera *labels* más grandes. Además como se dispone de una cota superior en el posible *label*, el proceso termina pues eventualmente alcanzará tal cota.

Sin la línea 6 del código, el algoritmo es una enumeración total de los caminos factibles. El test que se hace en esta línea elimina los caminos que no son prometedores basado en algún criterio de *dominancia*. Idealmente se pretenden eliminar prematuramente los *label* que no serán parte de la solución de costo mínimo y así no perder recursos extendiéndolos.

Un *label* L_1 **domina** a L_2, si ambos están asignados al mismo nodo, y no existe una extensión factible desde L_2 hasta el nodo terminal t que tenga un costo menor que la mejor extensión factible de L_1 hasta t. Puede resultar muy difícil identificar la dominancia entre *labels*, pues potencialmente se deben estudiar todas las extensiones de los correspondientes caminos hasta t. Por lo tanto, es necesario analizar si la ganancia obtenida por aplicar la dominancia es mayor que el costo de identificar los *labels* comparables.

3.3.1. Manejo de los Labels

La definición de *label* que usaremos en este trabajo, inicialmente se basó en la usada por Desrosiers et al. (1986), pero finalmente se toma como base la expuesta en Ropke and Cordeau (2009). En esta, se considera que el conjunto de recursos para este problema son 3: el tiempo de llegada al nodo, el costo de llegada y la carga del vehículo. Cada *label* debe contener información suficiente para poder reconstruir el camino desde el origen y poder compararlo con otro *label*. Se asume que el grafo donde se aplica el algoritmo posee N clientes, donde el conjunto de *pickups* vienen dado por $P = \{1, ..., N\}$ y el conjunto de *deliverys* por $D = \{N + 1, ..., 2N\}$. Para cada cliente i su respectivo *delivery* es el nodo $i + N$. Además se considera que el nodo origen es $s = 0$ y el nodo terminal es $t = 2N + 1$.

Se define cada *label* como una tupla $L = (i, t, l, c, V, O, L_p)$, donde:

- i: Nodo de llegada
- t: Tiempo de llegada al nodo i
- l: Carga del vehículo en el nodo i
- c: Costo del *label* al nodo i
- $V \subset \{1, ..., N\}$: Conjunto de nodos clientes visitados por la ruta, y posiblemente terminados
- $O \subset \{1, ..., N\}$: Conjunto de nodos de clientes servidos pero no terminados, o sea, el *pickup* ha sido servido pero no su respectivo *delivery*. Este conjunto se llama el de los clientes abiertos. (Concepto introducido originalmente por Dumas et al. (1991)
- L_p: Puntero al *label* del cual se genero L

Posteriormente para aplicar las estrategia del *branching* se agregó una componente extra a cada *label*, denotada por $p \in \{1, ..., N\}$ la cual guarda el último *pickup* realizado por la ruta. Se utiliza la notación $i(L)$ para referirnos al nodo de llegada asociado al *label* L, $t(L)$ para el tiempo de llegada asociado a L y así respectivamente.

Cuando se extiende un *label* L a lo largo de un arco $(i(L), j)$, la extensión es factible si:

$$t(L) + t_{i(L),j} \leq b_j \tag{3.18}$$

$$l(L) + q_j \leq Q \tag{3.19}$$

La restricción (3.18) asegura que la llegada al nodo j se haga dentro de su ventana de tiempo, y la restricción (3.19) asegura que se respete la capacidad del vehículo.

Además, L y j deben satisfacer alguna de las siguientes condiciones:

1. Si $1 \leq j \leq N$, o sea j es un *pickup*, entonces $j \notin V$. No puede ya haber sido visitado.
2. Si $N + 1 \leq j \leq 2N$, o sea, j es un *delivery*, entonces $j - N \in O$. Su *pickup* ya debe haber sido visitado y no debe haber sido servido.
3. Si $j = 2N + 1$, entonces $O = \emptyset$. No deben haber *pickups* no terminados arriba del vehículo.

Suponiendo que alguna de las 3 condiciones anteriores se cumple, el nuevo *label* L' es creado a partir de la información de L como sigue:

$$i(L') = j \tag{3.20}$$

$$t(L') = \text{máx}\{a_j, t(L) + t_{i(L),j}\} \tag{3.21}$$

$$l(L') = l(L) + q_j \tag{3.22}$$

$$c(L') = c(L) + d_{i(L),j} \tag{3.23}$$

$$V(L') = V \cup (\{j\} \cap P) \tag{3.24}$$

$$O(L') = \begin{cases} O(L') \cup \{j\} & \text{si } j \in P \\ O(L') \setminus \{j - n\} & \text{si } j \in D \end{cases} \tag{3.25}$$

Las ecuaciones (3.20)-(3.23) actualizan los valores del nodo terminal, tiempo, carga del vehículo y costo del nuevo *label*. La ecuación (3.24) actualiza el conjunto de nodos visitados. El nodo j es añadido solamente si es un *pickup*. La ecuación (3.25) actualiza el conjunto O el cual contiene los requerimientos abiertos, por lo tanto, si j es un *pickup*, este se agrega, pero si j es un delivery, es necesario eliminar de O su *pickup* pues ya se completó el requerimiento.

3.3.2. Eliminación de Labels y Dominancia

Los criterios de eliminación de *labels* se pueden separar en dos métodos (Dumas et al., 1991). Estos son, la post-factibilidad y la dominancia. La eliminación de estos *labels* resulta en la eliminación del camino asociado.

Se denotará $\mathcal{P}(L)$ al camino asociado al *label* L. Se dirá **post-infactible** a un *label* asociado a un camino $\mathcal{P}(L)$, factible desde el nodo 0 al nodo $i(L)$, el cual no puede extenderse desde el nodo $i(L)$ al nodo $2N+1$. Este concepto fue introducido por Desrosiers et al. (1986). Se usará la notación $\mathcal{P}_1 \to \mathcal{P}_2$ para el camino obtenido de la concatenación de $\mathcal{P}_1$ con $\mathcal{P}_2$.

Proposición 3.3.1. *Un label L tal que $O \neq \emptyset$ es eliminado si no existe una extensión desde $i(L)$ a $2N+1$ que visite los deliverys respectivos de los requerimientos abiertos.*

Se debe verificar la post-factibilidad con todos los *deliveries* correspondientes a los nodos del conjunto O, esto permite revisar que los clientes que ya se encuentran en el vehículo realmente puedan ser depositados. Para implementar esto se puede hacer un TSP con ventanas de tiempo con los *deliverys* que faltan al nodo terminal $2N+1$. En la práctica solo se realiza esta prueba cuando $|O| \leq 2$.

Proposición 3.3.2. *Si se tienen dos labels L_1,L_2 tal que $i(L_1) = i(L_2), t(L_1) \leq t(L_2), c(L_1) \leq c(L_2), V(L_1) = V(L_2), O(L_1) = O(L_2)$entonces L_2 puede ser eliminado.*

La demostración se basa en el hecho que todo camino que extiende a $\mathcal{P}(L_2)$ también extiende a $\mathcal{P}(L_1)$ con un costo menor. Una relajación de este criterio viene dada por la siguiente proposición:

Proposición 3.3.3. *Si se tienen dos labels L_1,L_2 tal que $i(L_1) = i(L_2), t(L_1) \leq t(L_2), c(L_1) \leq c(L_2), V(L_1) \subset V(L_2), O(L_1) \subset O(L_2)$ entonces L_2 puede ser eliminado.*

La demostración se basa en la Proposición 4 de Dumas et al. (1991). Sea $\mathcal{P}$ el camino óptimo (con respecto al costo) que extiende el camino de $\mathcal{P}(L_2)$ al nodo terminal $2N+1$. Se denotará por $\mathcal{P}'$ al camino obtenido de remover de $\mathcal{P}$ los *deliverys* correspondientes a la diferencia $O(L_2) \setminus O(L_1)$. Como $\mathcal{P}(L_2) \to \mathcal{P}$ es factible, también lo será $\mathcal{P}(L_1) \to \mathcal{P}'$. La ventanas de tiempo no serán violadas, pues el tiempo de viaje satisface la desigualdad triangular. La capacidad del vehículo no será sobrepasada en $\mathcal{P}(L_2) \to \mathcal{P}'$ pues no lo era en $\mathcal{P}(L_1) \to \mathcal{P}$, además $\mathcal{P}(L_2)$ no visita los *pickups* removidos de $\mathcal{P}$. El costo de $\mathcal{P}(L_2) \to \mathcal{P}'$ es menor o igual que el costo de $\mathcal{P}(L_1) \to \mathcal{P}$, pues $c(L_1) \leq c(L_2)$ y el costo de $\mathcal{P}'$ es menor o igual que el costo de $\mathcal{P}$. Esto se debe al hecho de que al remover *deliveries* de un camino no sube su costo, puesto que la construcción de los $\hat{d}_{ij}$ en 3.16 satisfacen la desigualdad triangular. Por lo tanto, la mejor extensión (con respecto al costo) de L_1 a $2N+1$ será siempre mejor que la correspondiente extensión de L_2 a $2N+1$. Así, L_1 domina a L_2. No es necesario considerar las cargas ya que si $O(L_1) \subset O(L_2)$, entonces $l(L_1) \leq l(L_2)$ $\square$

3.3.3. Implementación

Al implementar el algoritmo de *label setting*, los conjuntos V y O pueden guardarse como vectores binarios. En esta representación, el $i-$ésimo bit indica si el $i-$ésimo cliente es parte del conjunto. Consideremos una máquina con 32 bit de precisión y $N = 50$, entonces $\lceil 50/32 = 2 \rceil$ enteros son necesarios para guardar cada conjunto. El algoritmo de *label setting* tiene que hacer una gran cantidad de dominancias, en las cuales una inclusión de conjuntos al menos debe ser revisada. La inclusión se puede implementar eficientemente haciendo operaciones binarias tal que w bits pueden procesarse en paralelo, donde w es el tamaño de una palabra de máquina. Para comparar una palabra x y otra palabra y hacemos la operación $x\&y$, donde & realiza una comparación *and* binaria. Si $x - (x\&y) = 0$, entonces el conjunto correspondiente a x está incluído en el conjunto correspondiente a y. La otra inclusión puede revisarse de la misma manera. Si $x - (x\&y) \neq 0$ y $y - (x\&y) \neq 0$ entonces ninguno es subconjunto del otro.

3.3.4. Preproceso

En la literatura, se han sugerido una gran variedad reglas de preproceso. En esta tesis se han considerado la gran mayoría, principalmente las sugeridas en Dumas et al. (1991), Ropke and Cordeau (2009) y Desrochers et al. (1992).

El primer paso consiste en ajustar las ventanas de tiempo usando los caminos parciales $i \to N + i \to 2N + 1$ y $N + i \to 2N + 1$ para que sean admisibles para todos los valores $T_i \in [a_i, b_i]$ y $T_{N+i} \in [a_{N+i}, b_{N+i}]$. Se redefinen sucesivamente las ventanas de tiempo como sigue:

$$
\begin{aligned}
b_{N+i} &= \text{mín}\{b_{N+i}, b_{2N+1} - t_{i,2N+1}\} \\
b_i &= \text{mín}\{b_i, b_{N+i} - t_{i,N+i}\} \\
a_i &= \text{máx}\{a_i, a_0 + t_{0,i}\} \\
a_{N+i} &= \text{máx}\{a_{N+i}, a_i + t_{i,N+i}\}
\end{aligned}
$$

Una vez hecho lo anterior, para cada $k \in \{1, ..., 2N\}$ se aplican las reglas 2 y 3 expuestas en el trabajo de Desrochers et al. (1992):

$$
\begin{aligned}
a_k &= \text{máx}\left\{a_k, \text{mín}\left\{b_k, \min_{(k,j)\in E}\{a_j - t_{k,j}\}\right\}\right\} \\
b_k &= \text{mín}\left\{b_k, \text{máx}\left\{a_k, \max_{(i,k)\in E}\{b_i + t_{i,k}\}\right\}\right\}
\end{aligned}
$$

Dadas las ventanas de tiempo y las restricciones de precedencia, varios arcos pueden ser eliminados puesto que no pueden pertenecer a una solución factible del problema. Siguiendo a Dumas et al. (1991), las restricciones del problema son usadas para eliminar los siguientes arcos:

- **[prioridad]** Los siguientes arcos son eliminados, pues no tienen sentido: $(0, N + i)$, $(N + i, i)$, $(2N + 1, 0)$, $(2N + 1, i)$, $(i, 2N + 1)$ y $(2N + 1, N + i)$ para $i = 1, ..., N$ son eliminados

- **[capacidad]** La capacidad del vehículo no puede ser excedida en ningún momento. Si $q_i + q_j > Q, i, j \in \{1, ..., N\},\ i \neq j$ los siguientes arcos son eliminados: $(i, j),\ (j, i),\ (i, N+j),\ (j, N+i),\ (N+i, N+j)$ y $(N+j, N+i)$
- **[ventanas de tiempo]** Debe ser posible llegar a cada nodo dentro de su respectiva ventana de tiempo. Si $a_i + t_{ij} > b_j,\ i, j \in \{1, ..., 2N\}$, entonces el arco (i, j) es eliminado
- **[ventanas de tiempo en conjunto con precedencia]** Como el tiempo de viaje satisface la desigualdad triangular, algunos arcos pueden ser eliminados si no pueden ser parte de ninguna camino que incluya tanto al *pickup* como al *delivery* para algún cliente.
  - El arco $(i, N+j)$ es eliminado si el camino $j \to i \to N+j \to N+i$ es infactible considerando $t_j = a_j$.
  - El arco $(N+i, j)$ es eliminado si el camino $i \to N+i \to j \to N+j$ es infactible considerando $t_i = a_i$.
  - El arco (i, j) es eliminado si el camino $i \to j \to N+i \to N+j$ es infactible con $t_i = a_i$.
  - El arco $(N+i, N+j)$ es eliminado si los caminos $i \to j \to N+i \to N+j$ y $j \to i \to N+i \to N+j$ son ambos infactible considerando $t_i = a_i$ y $t_j = a_j$ respectivamente.

A continuación se presenta un ejemplo de la ejecución del algoritmo de Label-Setting:

3.4. Ejemplo Algoritmo de Label Setting

En esta sección se ilustra con un ejemplo de 4 clientes el algoritmo de programación dinámica empleado para generar nuevas columnas (Dumas et al., 1991). La tabla 3.1, muestra la matriz de costos entre los diferentes nodos de la red, luego la tabla 3.2 muestra la reducción obtenida después de aplicar el preproceso. Para hacer fácil identificar los clientes de *pickup and delivery* se usará la notación $\{1^+, 2^+, 3^+, 4^+\}$ y $\{1^-, 2^-, 3^-, 4^-\}$ para denotar a los nodos de *pickup* y *delivery* respectivamente. El *depot* será representado por los nodos 0 y 5. La capacidad del vehículo será igual a 100, el costo y el tiempo de atravesar un arco se considerarán iguales.

Costo de viaje (tiempo de viaje) entre los arcos												
nodos	0	1^+	2^+	3^+	4^+	1^-	2^-	3^-	4^-	5	Demandas	Ventanas T.
0	-	27	51	42	100	-	-	-	-	-	-	[360,720]
1^+	-	-	29	18	74	63	95	111	152	-	60	[540,600]
2^+	-	29	-	28	46	40	69	80	128	-	40	[540,600]
3^+	-	18	28	-	69	52	86	103	139	-	70	[360,650]
4^+	-	74	46	69	-	26	28	38	88	-	30	[580,650]
1^-	-	63	40	52	26	-	30	51	89	95	-60	[600,720]
2^-	-	95	-	86	28	30	-	18	60	64	-40	[600,720]
3^-	-	111	80	-	38	51	18	-	55	54	-70	[360,720]
4^-	-	152	128	139	-	89	60	55	-	14	-30	[600,720]
5	-	-	-	-	-	-	-	-	-		-	[360,720]

Tabla 3.1: Datos originales

Costo de viaje (tiempo de viaje) entre los arcos												
nodos	0	1^+	2^+	3^+	4^+	1^-	2^-	3^-	4^-	5	Demandas	Ventanas T.
0	-	27	51	42	100	-	-	-	-	-	-	[360,521]
1^+	-	-	-	-	-	63	-	-	-	-	60	[540,562]
2^+	-	-	-	-	46	40	69	-	-	-	40	[540,587]
3^+	-	-	-	-	69	-	-	103	-	-	70	[402,563]
4^+	-	-	-	-	-	-	28	38	88	-	30	[580,618]
1^-	-	-	-	-	-	-	30	-	-	95	-60	[603,625]
2^-	-	-	-	-	-	-	-	-	60	64	-40	[609,656]
3^-	-	-	80	-	38	-	-	-	55	54	-70	[505,666]
4^-	-	-	128	-	-	-	-	-	-	14	-30	[668,706]
5	-	-	-	-	-	-	-	-	-		-	[559,720]

Tabla 3.2: Datos reducidos

La Tabla 3.2 proviene de la Tabla 3.1 después de aplicar las reglas de preproceso de la sección 3.3.4. En esta tabla las ventanas de tiempo han sido reducidas calculando (en cada nodo) el tiempo de llegada más temprano y la salida más tarde que sean posibles. Luego, los arcos $(1^+, 3^+), (2^+, 3^+), (1^+, 3^-), (2^+, 3^-), (1^-, 3^-)$ y $(2^-, 3^-)$ son eliminados por la restricción de capacidad del vehículo. El resto de los arcos eliminados son por ventanas de tiempo. Por ejemplo, el arco $(1^+, 4^+)$ se desecha, pues los caminos $(1^+, 4^+, 1^-, 4^-)$ y $(1^+, 4^+, 4^-, 1^-)$ son infactibles.

En lo que sigue, se presenta detalladamente como opera el algoritmo de programación dinámica en la primera llamada. Para esto, se considera que la formulación del problema maestro se inicializó con las columnas triviales, o sea, un vehículo para cada cliente (matriz identidad). El costo de salida del *depot* es de 1000 unidades. Los costos de las columnas iniciales son $(1185, 1184, 1199, 1202)$, con esto las variables duales son $\pi = C_B B^{-1} = (1185, 1184, 1199, 1202)$ Por ejemplo el costo de la primera columna se obtiene como 1000 más el costo del camino $(0, 1^+, 1^-, 5)$: 1000+27+63+95=1185. Además hacer más fácil la lectura los *labels* son procesados por orden de largo.

Notas de la tabla 3.3:

(1) El costo del *label* 1.1 es 1000 más el costo del arco $(0, 1^+)$: 1000+27.

(2) El costo del *label* 2.1 es el costo del *label* 1.1 más el costo del arco $(1^+, 2^+)$, menos la variable dual asociada al cliente 1: 1027+29-1185=-129.

(3) Este *label* es rechazado pues el cliente 1 no está actualmente en el vehículo.

(4) El tiempo de salida en el nodo 4^+ es igual a la cota inferior de la ventana de tiempo, pues esperar está permitido.

(5) Este *label* es rechazado pues la capacidad del vehículo es excedida.

(6) Este *label* es rechazado pues no es posible visitar a 1^- posteriormente.

(7) Este *label* es rechazado pues el cliente 2 no se encuentra en el vehículo.

(8) Este camino es descartado pues no tiene costo reducido negativo

(9) Este *label* es rechazado pues no es posible visitar a 2^- posteriormente.

(10) El *label* 3.3 no es dominado por el *label* 3.2 pues no terminan en el mismo nodo.

(11) El *label* 3.4 no es rechazado pues aunque el costo marginal sea positivo, es solo una ruta parcial y la ruta final podría tener costo reducido negativo.

(12) Los *labels* 4.2, 4.3 y 4.4 son todos caminos parciales terminando en 4^- sin clientes en el vehículo. Sin embargo, ningún *label* domina a otro, pues los tiempos son decrecientes en el costo.

(13) Este *label* y 4.1 tienen el mismo tiempo de llegada al nodo 2^- y los mismo clientes siendo servidos (conjunto vacío) pero difieren en costo y tiempo. Sin embargo, este *label* tiene tiempo mayor ($645 > 639$) y costo mayor ($-1089 > -1243$). Por lo tanto es dominado por 4.1, así cualquier extensión del este *label* será inferior a la misma extensión en el *label* 4.1.

(14) Cuatro caminos con costo reducido negativo son encontrados, ver tabla 3.3 El mejor de estos viene dado por el *label* 5.2 con la columna $(0, 1, 0, 1)$ la cual debe entrar a la base en la siguiente iteración del algoritmo de simplex. En nuestro algoritmo, agregariamos las primeras 3 columnas, la cuarta sería decartada por ser igual a la tercera y tener mayor costo. Agregando estas tres columnas a las iniciales, es posible encontrar una nueva solución de la relajación lineal de la formulación del problema maestro. Si el subproblema no es capaz de encontrar columnas con costo reducido negativo, la solución de la relajación lineal es óptima. Si esta solución no es entera, es necesario hacer *branch and bound*.

Label predecesor	Último nodo visitado	Nodos potenciales llegada	Clientes Incompletos	Tiempo de llegada al nodo	Carga	Costo acumulado	Nombre Label	Notas
Inicialización								
-	-	0	$\emptyset$	360	0	0	0	0
Iteración 1								
0	0	1^+	$\{1^+\}$	540	60	1027	1.1	(1)
0	0	2^+	$\{2^+\}$	540	40	1051	1.2	
0	0	3^+	$\{3^+\}$	402	70	1042	1.3	
0	0	4^+	$\{4^+\}$	580	30	1100	1.4	
Iteración 2								
1.1	1^+	2^+	$\{1^+, 2^+\}$	569	100	-129	2.1	(2)
		1^-	$\emptyset$	603	0	-95	2.2	
1.2	2^+	4^+	$\{2^+, 4^+\}$	586	70	-87	2.3	
		1^-	rechazado $1^+ \notin \{2^+\}$					(3)
		2^-	$\emptyset$	609	0	-64	2.4	
1.3	3^+	4^+	$\{3^+, 4^+\}$	580	100	-88	2.5	(4)
		3^-	$\emptyset$	505	0	-54	2.6	
1.4	4^+	2^-	rechazado $2^+ \notin \{4^+\}$					
		3^+	rechazado $3^+ \notin \{4^+\}$					
		4^-	$\emptyset$	668	0	-14	2.7	
Iteración 3								
2.1	2^+	4^+	$\{1^+, 2^+, 4^+\}$	615	$130 > 100$ rechazado			(5)
		1^-	$\{2^+\}$	609	40	-1273	3.1	
		2^-	$\{1^+\}$	638 rechazo no es post-factible para 1^-				(6)
2.2	1^-	2^-	rechazado $2^+ \notin \emptyset$					(7)
		5	$\emptyset$	698	0	0	-	(8)
2.3	4^+	2^-	$\{4^+\}$	614	30	-1261	3.2	
		3^-	rechazado $3^+ \notin \{2^+, 4^+\}$					
		4^-	$\{2^+\}$	674 rechazo no es post-factible para 2^-				(9)
2.4	2^-	4^-	rechazado $4^+ \notin \emptyset$					
		5	$\emptyset$	673	0	0	-	(8)
2.5	4^+	2^-	rechazado $2^+ \notin \{3^+, 4^+\}$					
		3^-	$\{4^+\}$	618	30	-1252	3.3	(10)
		4^-	$\{3^+\}$	668 rechazo no es post-factible para 3^-				
2.6	3^-	2^+	$\{2^+\}$	585	40	26	3.4	(11)
		4^+	$\{4^+\}$	580	30	-16	3.5	(4)
		4^-	rechazado $4^+ \notin \emptyset$					
		5	$\emptyset$	559	0	0	-	(8)
2.7		5	$\emptyset$	682	0	0	-	(8)

Label predecesor	Último nodo visitado	Nodos potenciales llegada	Clientes Incompletos	Tiempo de llegada al nodo	Carga	Costo acumulado	Nombre Label	Notas
Iteración 4								
3.1	1^-	2^-	$\emptyset$	639	0	-1243	4.1	
		5	rechazado, 2^+ en el vehículo					
3.2	2^-	4^-	$\emptyset$	674	0	-1201	4.2	(12)
		5	rechazado, 4^+ en el vehículo					
3.3	3^-	2^+	$\{2^+, 4^+\}$	698 ventana.T. excedida				
		4^+	rechazado, 4^+ ya fue servido					
		4^-	$\emptyset$	673	0	-1197	4.3	(12)
3.4	2^+	4^+	$\{2^+, 4^+\}$	631 ventana.T. excedida				
		1^-	rechazado, $1^+ \notin \{2^+\}$					
		2^-	$\emptyset$	654	0	-1089	rechazado	(13)
3.5	4^+	2^-	rechazado, $2^+ \notin \{4^+\}$					
		3^-	rechazado, $3^+ \notin \{4^+\}$					
		4^-	$\emptyset$	668	0	-1130	4.4	(12)
Iteración 5								
4.1	2^-	4^-	rechazado, $4^+ \notin \emptyset$					
		5	$\emptyset$	703	0	-1179	5.1	(14)
4.2	4^-	5	$\emptyset$	688	0	-1187	5.2	(14)
4.3	4^-	5	$\emptyset$	687	0	-1183	5.3	(14)
4.4	4^-	5	$\emptyset$	682	0	-1116	5.4	(14)

Tabla 3.3: Algoritmo de Programación Dinámica detallado por pasos

Label	Camino	Columna	Costo Reducido	Costo Real
5.1	$(0, 1^+, 2^+, 1^-, 2^-, 5)$	$(1, 1, 0, 0)$	-1179	1190
5.2	$(0, 2^+, 4^+, 2^-, 4^-, 5)$	$(0, 1, 0, 1)$	-1187	1199
5.3	$(0, 3^+, 4^+, 3^-, 4^-, 5)$	$(0, 0, 1, 1)$	-1183	1218
5.4	$(0, 3^+, 3^-, 4^+, 4^-, 5)$	$(0, 0, 1, 1)$	-1116	1285

Tabla 3.4: Solución Final

La Tabla 3.4 muestra las solución final del algorimto. Como todas las columnas tienen costo reducido negativo, se agregan al RPM.

3.5. Heurísticas para el PDPTW

El subproblema consiste en encontrar rutas de costo reducido negativo que mejoren la solución actual del problema maestro restringido RPM. Para esto, no es necesario resolver a optimalidad el subproblema, basta con encontrar rutas buenas y agregarlas al RPM.

3.5.1. Enumeración Explícita

Esta no es realmente un metodología a aplicar al problema estudiado, es más bien una manera de ilustrar que un enfoque así no tiene mucho sentido.

Sin considerar las ventanas de tiempo, la primera estrategia de solución que se podría considerar a implementar, consiste en enumerar todas las rutas posibles y buscar la con menor costo. En el caso de rutas totales entre un orígen y un destino, en un grafo de N nodos, es fácil ver que la cantidad de rutas viene dada por la fórmula:

$$R(N) = \sum_{k=0}^{N} \binom{N}{k} k!$$

Así, se tiene que $R(0) = 1, R(1) = 2, R(2) = 5, R(4) = 65, R(6) = 1957$.
Para el caso en que el conjunto de nodos se divide en orígenes y destinos (*Pickups* y *Deliveries*) el cálculo de las rutas es más difícil, pues hay que respetar la precedencia. Consideremos $2N$ nodos, donde los primeros $1, ..., N$ son los orígenes y $N + 1, ..., 2N$ son los destinos. Se denotará por s al el nodo inicial y t al nodo final. Es necesario establecer una recursión para insertar un cliente nuevo a una ruta ya existente, para esto, se considera una ruta que sirve a k clientes:

$$s \to c_1 ... c_2 ... c_k ... c_{1+N} .... c_{k+N} ... \to t$$

Por ejemplo, el orígen del cliente $k + 1$ puede ser inserado en la posición 1 del string:

$$s \to C_{k+1} c_1 ... c_2 ... c_k ... c_{1+N} .... c_{k+N} ... \to t$$

Como el destino del cliente $k + 1$ debe venir despúes del orígen, restan solo $2k + 1$ posiciones para insertarlo, si ahora el orígen es trasladado un lugar más a la derecha, restan $2k$ posiciones para insertar el destino, así sucesivamente, se obtiene que en una ruta de k clientes, es posible agregar un cliente nuevo de $(2k + 1)(k + 1)$ formas distintas. Si r_k es la cantidad de rutas que sirven exactamente a k clientes, la cantidad de rutas totales viene dada por:

$$R(N) = \sum_{k=0}^{N} \binom{N}{k} r_k$$

donde los r_k se calculan mediante la siguiente recurrencia:[1]

$$r_0 = 1$$
$$r_{k+1} = (2k+1)(k+1)r_k$$

Tabla 3.5: Comparación de cantidad de rutas con y sin precedencia

$2N =$	2	4	6	8	10	12
Sin Precedencia	5	65	1.957	109.601	9.864.100	$1{,}30206 * 10^9$
Con Precedencia	2	9	112	2.921	126.966	8.204.497

Efectivamente se aprecua que la cantidad de rutas cuando hay precedencia involucrada es mucho menor al caso sin precedencia. Sin embargo incluso para problemas pequeños con 20 clientes, el número de rutas totales es grande como para realizar una enumeración. Cabe notar que al agregar las ventanas de tiempo el conjunto de rutas se reduce, pero sigue siendo muy grande para enumerarlo. El problema de encontrar rutas con costo reducido negativo para el subproblema debe ser resuelto lo más rápido posible, por lo tanto este método no es viable.

[1]Esta sucesión no se ha relacionado antes con problemas de ruteo, las referencias se pueden ver en www.oeis.org

3.5.2. Heurísticas para el Subproblema

Como es necesario encontrar rutas con costo reducido negativo en cada iteración del problema maestro, no es siempre necesario resolver a optimalidad el subproblema. Por lo general, el algoritmo de *label-setting* encuentra rutas con costo reducido negativo antes de terminar de comparar todas las rutas. Las heurísticas que se presentan a continuación trabajan truncando de distintas maneras el algoritmo de *label-setting* expuesto en la sección 3.3.

H1 Restringir el algoritmo de *label-setting* a una red de tamaño reducido es un práctica muy común. Construir redes con el 30 % o 50 % de los nodos, considerando los mejores arcos con respecto al costo es lo que proponen en Dumas et al. (1991). Una variante a esto consiste en definir G_m una red en la cual cada nodo mantenga los m mejores vecinos. Dependiendo de la red, el valor m de se ajusta para obtener resultados más eficientes. Para nuestro problema, para cada *depot*, construimos la red auxiliar D_m consistente en los m mejores arcos con respecto a d_{ij}. Para cada nodo $i \in \{1, ..., 2N\}$ se limita el conjunto de vecinos $N(i)$ a m. Una vez aplicado esto a cada nodo del grafo, se completa con los arcos $(0, i)$, $(i, i + N)$, $(N + i, 2N+1)$. Siguiendo los resultados expuestos en Ropke and Cordeau (2009), consideraremos $m = 5$ y $m = 10$. Si no es posible encontrar rutas de costo reducido negativo en D_5, se prosigue con D_{10}.

H2 Mantener una cantidad limitada de *labels* en cada momento, para el algortimo 3.5.3, consiste en poner un límite al cardinal del conjunto H. Se limita $|H| \leq \lambda$, tal que los mejores *labels* con respecto al costo son guardados. Al igual que **H1**, esta heurística se aplica por etapas, considerando $\lambda = 500, 100, 1500$. Para instancias simples, este procedimiento puede retornar la solución óptima. Lo valores de λ escogidos han probado funcionar bien para una variedad de instancias (Ropke and Cordeau, 2009; Baldacci et al., 2011).

H3 Esta metodología consiste en mejorar una ruta r encontrando la mejor permutación para servir a sus clientes. Para esto, supongamos que $r = [0, 1, N+1, 7, 14, N+14, N+7, 2N+1]$ es una ruta la cual queremos mejorar. Esta ruta sirve a los clientes $1, 7$ y 14, por lo tanto, se construye un grafo auxiliar G_r que solo contenga los nodos $0, 1, N+1, 7, N+7, 14, N+14, 2N+1$, con su respectivos arcos. Sobre este grafo, resolvemos un camino más corto usando el algoritmo de *label*-setting. El camino resultante es la mejor manera de servir a estos clientes.
Está metodología se puede aplicar tanto a variables de ruta usadas en una solución actual, y a variables de ruta que no tienen costo reducido negativo. Elegimos para esto un conjunto de hasta 4 variables, y aplicamos el procedimiento sobre un grafo auxiliar que contiene a todos los clientes de estas variables. Para esto es necesario aplicar el preproceso sobre este grafo particular pues las ventanas de tiempo son ajustadas y hay arcos infactibles eliminados.

3.5.3. Branching

Una solución óptima del LPM puede contener variables con valores no enteros. Aplicar sobre esta solución un esquema *branchandbound* standard no funciona, pues el conjunto de columnas

no contiene todas las variables. Cuando las variables son fijadas durante la ramificación y el LPM es reoptimizado, las variables duales pueden tener nuevos valores. Como resultado, una columna que no tuvo costo reducido negativo, ahora sí puede tenerlo. Por lo tanto, nuevas columnas deben irse generando a medida que se recorre el árbol.
Para evitar ese problema, se resuelve un *Branch-and-Price* que es el nombre que se le da a la versión generalizada del *Branch and Bound* en donde generación de columnas es usado para resolver un problema lineal apropiado en cada nodo del árbol. Lo importante es crear una estrategia adecuada de ramificación que sea compatible con el *pricing problem*. Para esto, Berger et al. (2007) usan una estrategia donde se mantiene la estructura de camino más corto en el *pricing problem*. La estrategia fundamental consiste en dos reglas de ramificación para los dos tipos de variables.
Para las variables de localización X_j, la convencional dicotomía de ramificación es adecuada. Fijar la variable $X_j = 1$ es fijar la utilización del depot j imponiendo 1 en su valor, y resolver el *pricing problem* para el depot j. Fijar la variable en $X_j = 0$ se logra imponiendo todas las $Y_{jk} = 0$ para todas las correspondientes $k \in P'_j$ en el RPM y después resolver el *pricing problem*.

La idea de la estrategia es poder transferir, desde el problema maestro PM al subproblema, la información de la rama. Hay que notar que una ramificación dicotómica no es posible en las variables de ruta, sin ningún cambio en el subproblema. La primera estrategia que podría ser considerada, consiste en fijar en 0 o 1 las variables de flujo en arcos las cuales se denotarán por y_{ij}, realizando esto para cada arco de la ruta Y_r.
Ejemplo: Se cuenta con una variable de ruta Y_r la cual no es entera (asociada a algun depot, que será omitido en la notación). La ruta correspondiente está dada por: $0 \to 1 \to 2 \to n+2 \to n+1 \to 2n+1$. Con esto, seis ramas son creadas:

$$
\begin{aligned}
&y_{0,1} = y_{1,2} = y_{2,n+2} = y_{n+2,n+1} = y_{n+1,2n+1} = 1 \\
&y_{0,1} = y_{1,2} = y_{2,n+2} = y_{n+2,n+1} = 1,\ y_{n+1,2n+1} = 0 \\
&y_{0,1} = y_{1,2} = y_{2,n+2} = 1,\ y_{n+2,n+1} = 0 \\
&y_{0,1} = y_{1,2} = 1,\ y_{2,n+2} = 0 \\
&y_{0,1} = 1,\ y_{1,2} = 0 \\
&y_{0,1} = 0
\end{aligned}
$$

En la primera rama solo es posible utilizar la ruta $0 \to 1 \to 2 \to n+2 \to n+1 \to 2n+1$. En la cuarta rama, es necesario salir del *depot* e ir directo al cliente 1 e inmediatamente después al cliente 2, pero no es posible ir inmediatamente después a dejar al cliente 2. La última rama solo prohibe las rutas que comienzan con el cliente 1.

Esta información es fácil de transferir a la red auxiliar del subproblema. Sin embargo, el árbol crece muy rápido, pues para cada ruta que satisface n_r clientes, se crean $2(n_r + 1)$ ramas. Por lo tanto, se usará la estrategia propuesta por Dumas et al. (1991). Se definen variables de orden O_{ij}, con $i, j \in P \cup \{0, 2n+1\}$. Para un camino r que satisface n_r clientes, solo $(n_r + 2)$ ramas son creadas. Similar a la estrategia anterior, para el ejemplo 4 ramas son creadas:

$$O_{0,1} = O_{1,2} = O_{2,2n+1} = 1$$
$$O_{0,1} = O_{1,2} = 1,\ O_{2,2n+1} = 0$$
$$O_{0,1} = 1,\ O_{1,2} = 0$$
$$O_{0,1} = 0$$

En la segunda rama, el primer *pickup* de la ruta es el nodo 1, después el nodo 2, y debe haber al menos un *pickup* extra antes del llegar al depot. Esto parece muy restrictivo, pero esto solo ocurre en rutas que contienen al *pickup* 1 y 2, por lo tanto una ruta como $0 \to 3 \to 4 \to n{+}3 \to n{+}4 \to 2n{+}1$ está totalmente permitida en la segunda rama. Precisamente:

$$O_{ij} = 1, \Rightarrow \text{Si } j \text{ está en la ruta después de } i\text{, viene inmediatamente después.} \tag{3.26}$$
$$O_{ij} = 0, \Rightarrow \text{Si } i \text{ está en la ruta, } j \text{ no puede ir inmediatamente después.} \tag{3.27}$$

La información es fácil de transferir a los subproblemas si los *labels* del algortimo de programación dinámica son modificados, agregando una componente p que represente al último *pickup visitado* ($p = 0$ en el inicio de la ruta, $p = 2n + 1$ al final). Nuevas reglas se deben imponer en el algoritmo de camino más corto, para satisfacer estas nuevas restricciones. El subproblema no es más díficil de resolver, dado que esta nueva información está estrechamente relacionada con la que ya se tenía.

A continuación se presentan algunos detalles de implementación del *branching* propuesto anteriormente, supongamos que para un par $i, j \in P \cup \{0, 2n+1\}$, estamos en una rama en la cual hay que imponer $O_{ij} = 0$, en el problema maestro LPM debemos imponer que todas las columnas (rutas) en las cuales el *pickup* del cliente j viene inmediatamente después del *pickup* del cliente i deben ponerse en 0. Analíticamente, consideremos que el conjunto total de columnas en una iteración del algoritmo es Ω' (omitiremos a cual *depot* pertenecen pues el *branching* se hace independiente del *depot*), llamemos $S^0_{ij} = \{k \in \Omega' \ : \ i, j \in k,\ \text{y } j \text{ viene inmediatamente después de } i\}$, así S^0_{ij} son las columnas se deben fijar en 0.

$$Y_k = 0, \quad \forall k \in S^0_{ij}$$

En el caso de la restricción $O_{ij} = 1$ es un poco más sutil, pues se hace poniendo en cero las rutas que no satisfacen esto. Así, $S^1_{ij} = \{k \in \Omega' \ : \ i, j \in k,\ \text{y } j \text{ está después de } i\text{, pero no inmediatamente después}\}$

$$Y_k = 0, \quad \forall k \in S^1_{ij}$$

Haciendo esto, las variables duales originales π y μ se llevan todo el peso del *branching*, no haciendo necesario agregar variables duales nuevas.

Con respecto al subproblema, es necesario hacer un cambio en la línea 6 del Algoritmo . Para poder extender un *label* L a un nodo j, hay que revisar el último *pickup* que hizo el *label* L, llamemos i a este *pickup*. Si $j \in D$, o sea es un *delivery*, la extensión se hace normalmente. Si $j \in P \cup \{0, 2N + 1\}$, entonces debemos revisar todas las reglas de *branching* terminadas en j, de

acuerdo a esto, tenemos 3 casos:

- Si $O_{ij} = 1$ está en las restricciones, extender el *label*.
- Si $O_{mj} = 1$ para algún $m \neq i$, revisar si m ya fue visitado, si no lo ha sido, extender el *label*.
- Si $O_{ij} = 0$ entonces no está permitida la extensión a j.

Este procedimiento se debe hacer para cada *label* que se procesa y se desea extender.

3.6. Resumen del Algoritmo

El siguiente esquema muestra como funciona el Algoritmo de generación de columnas:

Algorithm 2 Pseudo-código para algoritmo de generación de columnas

```
Generar un conjunto de columnas iniciales Ω
while True do
    Resolver LPM(Ω)
    Γ = Columnas obtenidas del subproblema
    Ω = Ω ∪ Γ
    if Γ = ∅ then
        Break
```

Este procedimiento entrega una solución del problema maestro relajado (donde las variables no son necesariamente enteras). Si la solución encontrada por el Algoritmo 2 fuese entera, entonces se ha encontrado el óptimo del problema, de no ser así comienza el esquema de *branching*. Para esto se modificó el subproblema para recibir las reglas de *branching* establecidas en la sección anterior.
El *branching* se hace a dos niveles, dada una solución no entera del LPM obtenida desde el Algoritmo 2 primero se analiza si existen variables de *depot* no enteras. De ser así, por cada *depot* se crean dos problemas. Luego se procede con las rutas, eligiendo la ruta con valor más cercano a 1. Para esta ruta se procede a crear un subproblema modificado para cada una de las ramas del *branching*, como lo explica la sección anterior. Cada uno de estas ramas se resuelve usando el Algoritmo 2, la única diferencia es que las columnas obtenidas del subproblema respetan las restricciones propias de la rama de *branching* correspondiente.
En cada una de estas ramificaciones se va calculando la cota inferior (LB) y superior (UB), para tratar de cerrar el GAP=UB-LB. Para UB se utiliza el valor que entrega CPLEX mediante el *branch and bound* incorporado. Como LB se utiliza el valor del óptimo del LPM, es fácil notar que el valor óptimo z^* satisface $LB \leq z^* \leq UB$.

3.7. Resultados Computacionales

Esta sección describe los experimentos computacionales realizados con el algoritmo de *branch and price* propuesto por este trabajo. Diversas instancias son usadas para obtener los resultados, algunas de estas son variantes de las instancias originales encontradas en la literatura para el PDPTW y otras instancias son generadas para probar partes específicas del modelo. El algoritmo fue codificado en $Python$ versión 2.7 para MAC OSX y todos los experimentos fueron realizados en un computador *Intel i7* (2.2 GHz). Como $LP - solver$ se usó CPLEX 12.0.1 en conjunto con la interfaz de $Python$ para CPLEX. Todos los experimentos tenían una limitación en su ejecución de 1 hora.

Las instancias a considerar son modificaciones de las instancias usadas en Ropke and Cordeau (2009). Estas instancias originales consideran un único *depot* localizado en el centro de un cuadrado. Las posiciones de los clientes son elegidas uniformemente en un cuadrado de $[0, 50] \times [0, 50]$. La carga de cada cliente es elegida aleatoriamente en el intervalo $[5, Q]$, donde Q es la capacidad del vehículo. Se considera un horizonte $T = 600$ y cada ventana de tiempo tiene ancho W. Las ventanas de tiempo son construidas aleatoriamente para cada cliente i. En Ropke and Cordeau (2009), el objetivo principal consiste en minimizar el número de vehículos usados, por lo tanto, se agrega un costo de 10^4 a cada arco que sale del depot. En el problema que resuelve este trabajo, el objetivo es minimizar el costo de localización simultáneamente con el de ruteo, por lo tanto no se considera este valor en todas las instancias. Para poder usar estas instancias para el problema de localización y ruteo con *pickup and delivery* es necesario agregar nuevos *depots*, los cuales son agregados uniformemente, manteniendo el depot central original. Se consideran instancias hasta con 7 depots. Todas las instancias usadas son variaciones de las instancias tipo AA de Ropke and Cordeau (2009). Así las instancias con 10 y 15 clientes, son sólo los primeros 10 o 15 clientes de las respectivas A o B originales. Para obtener instancias con $W = 30$ se modificó para cada $i \in \{1, ..., N\}$ la correspondiente ventana de tiempo superior $\hat{b}_i = b_i - 30$.

Considerando la cantidad de *depots* $|J|$ y la cantidad de clientes $N = |I|$, el cuadro 3.7 resume las características de nuestras instancias. Además las figuras 3.3-3.5 entregan una representación gráfica de las instancias. Las instancias con menos de 7 *depots* sólo consideran los primeros *depots*. Para las gráficas se usará la notación i^+ e i^- para representar los *pickup* y *delivery* respectivamente. Los nodos rojos representan la ubicación de los *depots* y los blancos los clientes.

$\|J\|$	Q	W	N
1	15	30	10
1	15	60	10
3	15	30	10
3	15	60	10
3	15	30	15
3	15	60	15
3	15	30	30
3	15	60	30
7	15	30	30
7	15	60	30
3	20	30	30
3	20	60	30
7	20	30	30
7	20	60	30

Tabla 3.6: Descripción de las Instancias

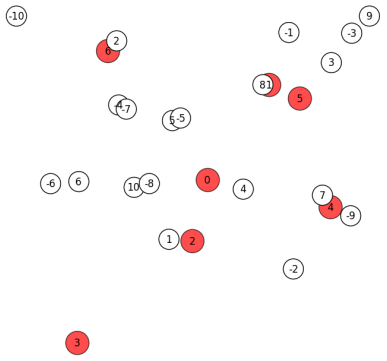


Figura 3.3: AA10: Instancia de 10 clientes

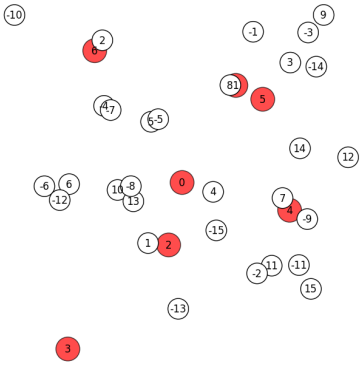


Figura 3.4: AA15: Instancia de 15 clientes

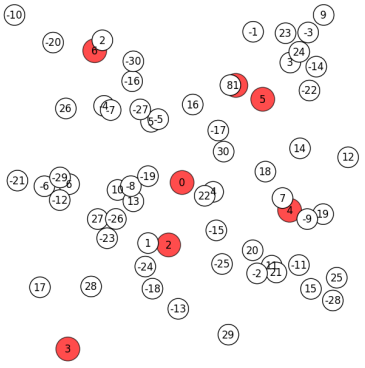


Figura 3.5: AA30: Instancia de 30 clientes

3.7.1. Preproceso

Los resultados del preproceso expuesto en 3.3.4 se presentan en la siguiente tabla, agregados por tipo de instancia. Se considera el promedio total de por instancia, pues el preproceso se hace por cada *depot*.

	$W = 30, Q = 15$	$W = 30, Q = 20$	$W = 60, Q = 15$	$W = 60, Q = 20$
AA10				
Ventana de Tiempo:				
Número de a_l modificados	10.71	10.71	10.71	10.71
Número de b_l modificados	10.00	10.00	10.00	10.00
Modificación promedio de a_l	0.95	0.95	0.95	0.95
Modificación promedio de b_l	0.21	0.21	0.21	0.21
Eliminación de Arcos:				
Arcos originales instancia	440.00	440.00	440.00	440.00
Eliminados por Prioridad	20.00	20.00	20.00	20.00
Eliminados por Capacidad	210.00	90.00	210.00	90.00
Eliminados por Ventana de T.	72.14	132.14	62.00	114.00
Eliminados por Ventana y Precedencia	27.29	87.29	23.00	77.00
Porcentaje de arcos eliminados	74.87	74.87	71.59	68.41
AA15				
Ventana de Tiempo:				
Número de a_l modificados	15.71	15.71	15.71	15.71
Número de b_l modificados	15.00	15.00	15.00	15.00
Modificación promedio de a_l	0.71	0.71	0.71	0.71
Modificación promedio de b_l	0.20	0.20	0.20	0.20
Eliminación de Arcos:				
Arcos originales instancia	960.00	960.00	960.00	960.00
Eliminados por Prioridad	30.00	30.00	30.00	30.00
Eliminados por Capacidad	534.00	270.00	534.00	270.00
Eliminados por Ventana de T.	150.29	279.29	137.00	249.00
Eliminados por Ventana y Precedencia	47.86	178.86	40.00	164.00
Porcentaje de arcos eliminados	79.39	78.97	77.19	74.27
AA30				
Ventana de Tiempo:				
Número de a_l modificados	31.57	31.57	31.57	31.57
Número de b_l modificados	30.00	30.00	30.00	30.00
Modificación promedio de a_l	0.81	0.81	0.81	0.81
Modificación promedio de b_l	0.23	0.23	0.23	0.23
Eliminación de Arcos:				
Arcos originales instancia	3720.00	3720.00	3720.00	3720.00
Eliminados por Prioridad	60.00	60.00	60.00	60.00
Eliminados por Capacidad	2238.00	1158.00	2238.00	1158.00
Eliminados por Ventana de T.	632.29	1150.43	570.71	1043.29
Eliminados por Ventana y Precedencia	182.43	711.86	157.00	649.43
Porcentaje de arcos eliminados	83.68	82.80	81.33	78.24

Tabla 3.7: Resultados pre-proceso de las instancias (promedios para 30 instancias de cada tipo)

Ls restricción de capacidad es la que elimina más arcos. En cada *pickup* la cantidad a recogerse oscila entre 5 y Q, como la capacidad del vehículo es limitada (15 o 20) no pueden haber más de 3 (o 4) cargas en el vehículo en ningún momento. En particular, ocurre que la carga de 2 clientes sobrepasa en muchas ocasiones a Q. Esto elimina una gran cantidad de arcos entre *pickups*.
Para las instancias tipo AA de Ropke and Cordeau (2009), las ventanas de tiempo son bastante estrechas, así una gran cantidad de arcos son eliminados por no respetar las ventanas de tiempo en conjunto con la precedencia. Con respecto a la modificación de a_l y b_l, esta no es muy grande, pues las instancias están construidas de manera tal que la llegada al nodo y salida de el ya son bastante ajustadas.
En general el preproceso es una parte importante del algoritmo pues ayuda a reducir el tamaño de las instancias, lo cual acelera considerablemente el tiempo de ejecución de los subproblemas. La cantidad de arcos eliminados está asociado principalmente a la construcción de las instancias y no se puede generalizar los resultados para instancias de cualquier tipo. La geometría y la estrechez de las ventanas de tiempo, tanto como la capacidad del vehículo son factores cruciales para poder

establecer la eliminación de los arcos.

3.7.2. Análisis de Sensibilidad en el Costo de los Depots

La función objetivo del LPRPDP, tiene un parte asociada a los costos fijos de los *depots*, (ver ecuación 3.1). El valor de este costo fijo, incide directamente en el tipo de soluciones que se obtienen. Lo lógico, es que a mayor sea el costo fijo de abrir un *depot*, la solución óptima utilice menor cantidad de *depots*. La métodogía para realizar este análisis consiste en construir 30 instancias similares, y analizar el promedio de las soluciones. Para construir cada una de estas instancias, se consideran fijas las ubicaciones de los *depots* y las ventanas de tiempo de los clientes. La posición de los clientes son elegidas aleatoriamente respetando que el *pickup* y *delivery* se puedan satisfacer dentro de su ventana de tiempo. Para este análisis solo se consideró instancias de 30 clientes. La figura 3.6 muestra la utilización de *depots* en la solución óptima, a medida que se sube su costo de apertura.

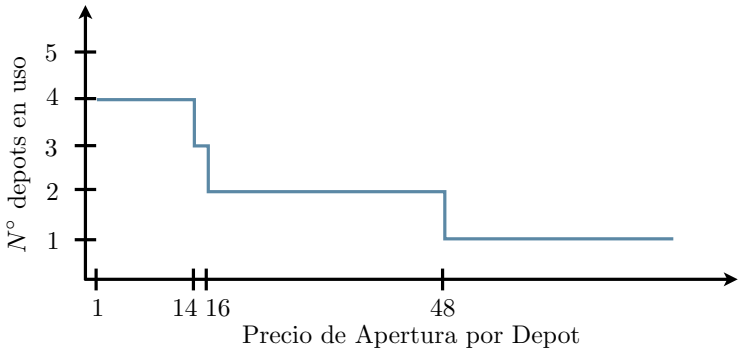


Figura 3.6: Uso de *depots* en solución óptima vs el costo unitario de apertura de cada *depot* para las instancias AA de 30 clientes

Para hacer más variado el análisis se considera también instancias similares a las de Ropke and Cordeau (2009) donde existe un costo de 10.000 unidades por utilizar cada vehículo. La idea de imponer este valor es minimizar la cantidad de vehículos usados en la solución óptima, implícitamente. (sin agregar esto en la función objetivo).

En la figura 3.7 el análisis se torna interesante cuando el valor de abrir un depot supera el precio de una ruta (cada ruta tiene un costo base de 10.000 unidades). Hay que notar que en este caso se ven efectos cruzados, pues abrir un *depot* implica que al menos una ruta sale desde él, por lo tanto, la solución óptima trata de usar la menor cantidad de rutas posible, es decir, la menor cantidad de *depots*.

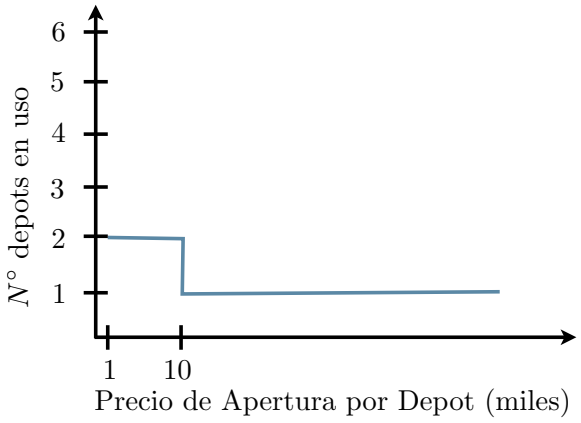


Figura 3.7: Uso de *depots* en solución óptima vs el precio costo fijo de apertura para las instancias AA de 30 clientes original de Ropke and Cordeau (2009)

3.7.3. Instancias Especiales

La topología de las instancias es importante para la planificación tanto de la ubicación de los *depots* como de las rutas a utilizarse. En distintas situaciones tanto ciudadanas como extra urbanas, se encuentran organizaciones espaciales particulares. Por ejemplo en una ciudad como Santiago, donde cada comuna es como un gobierno regional, la necesidad de viajes intra comunales es alta. Por lo tanto, pensar en localizar centros de almacenamiento o transferencia en cada comuna es bastante razonable. Desde un nivel macro, los viajes extra urbanos en Chile regional, pueden estudiarse como un corredor por el cual es necesario transportar los bienes a través de un eje central.

En esta parte se consideran 2 tipos de instancias. El primer tipo, son instancias *clusterizadas* o agrupadas (donde los clientes se encuentra agrupados por sectores), para ello, consideramos los clientes distribuidos en un cuadrado de 100×100, que a su vez dividimos en 4 cuadrantes de 50×50. Dentro de cada uno de estos cuadrantes repartimos los clientes. Una vez localizado el cliente i en un cuadrante, tanto su *pickup* como *delivery* se localizan en él. Se consideran instancias con 30 clientes y 7 *depots*, donde los clientes son distribuidos aleatoriamente en los cuadrantes, y luego la posición de su *pickup* y *delivery* se distribuyen uniformemente en el cuadrante. Las ventanas de tiempo usadas para estas instancias son construidas de la misma manera que en las instancias anteriores. Para aumentar la riqueza del análisis se construyen 30 instancias de esta manera. La figura 3.8 es un ejemplo de una de estas.

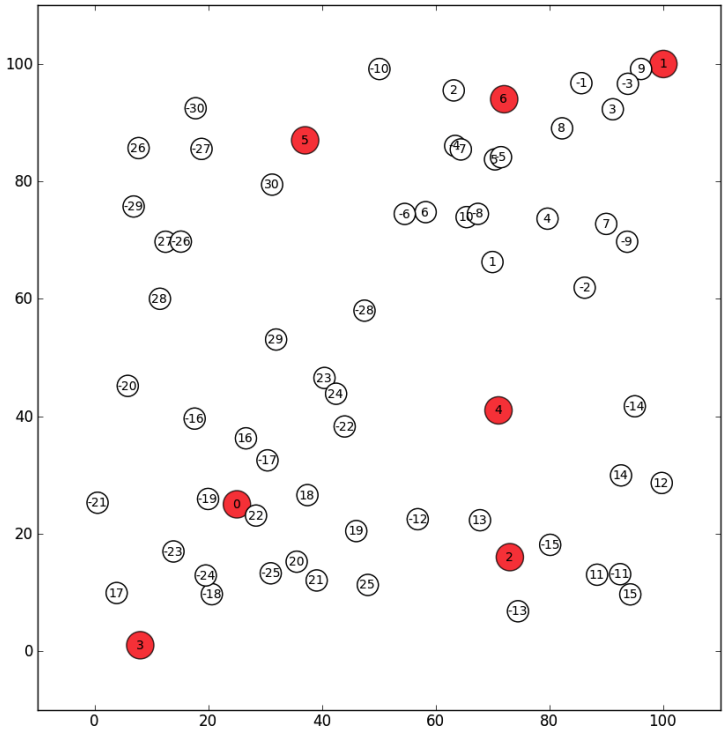


Figura 3.8: Instancia AA30C clusterizada

AA30C	$W = 30, Q = 15$	$W = 60, Q = 15$
Ventana de Tiempo:		
Número de a_l modificados	33	34
Número de b_l modificados	30	30
Modificación promedio de a_l	3.95	7.95
Modificación promedio de b_l	0.21	0.21
Eliminación de Arcos:		
Arcos originales instancia	3720	3720
Eliminados por Prioridad	120.00	120.00
Eliminados por Capacidad	2238.00	2238.00
Eliminados por Ventana de T.	712.14	641.33
Eliminados por Ventana y Precedencia	189.29	198.34
Porcentaje de arcos eliminados	88.45	85.39

Tabla 3.8: Resultados pre-proceso de las instancias Clusterizadas

En este tipo de instancias, se repite el análisis de costos de los *depots*. Dado que los clientes se encuentran agrupados por sectores, esto fuerza de cierta manera la apertura de un *depot* por sector en la mayoría de los casos. la distribución de los clientes, el precio del *depot* incide menos en la solución obtenida. Ver figura 3.9

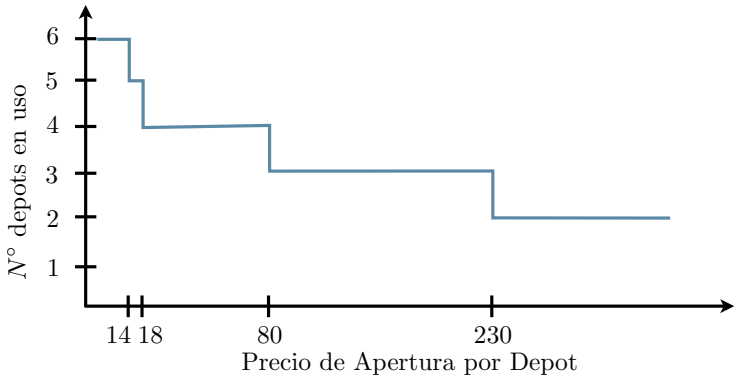


Figura 3.9: Uso de *depots* en solución óptima vs el precio costo fijo de apertura para las instancias clusterizadas AA30C

El otro tipo de instancias que se consideró, son instancias en las cuales los nodos se encuentran distribuidos a lo largo de un corredor. Aplicamos la misma metodología para construir estas instancias, solamente limitando a que los clientes sean repartidos en una faja diagonal de 30 unidades de ancho. La figura 3.10 representa una de estas instancias.

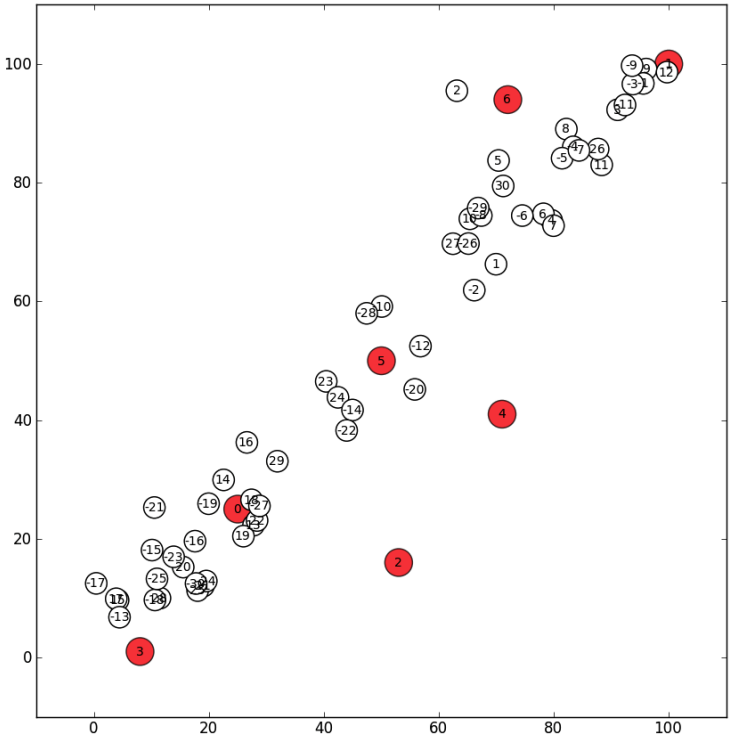


Figura 3.10: Instancia AA30Co tipo corredor

El preproceso de las instancias con corredor viene dado por el cuadro 3.9 y su respectivo análisis de costos por la figura 3.11

AA30Co	$W = 30, Q = 15$	$W = 60, Q = 15$
Ventana de Tiempo:		
Número de a_l modificados	25	25
Número de b_l modificados	21	20
Modificación promedio de a_l	7.99	7.95
Modificación promedio de b_l	6.57	4.09
Eliminación de Arcos:		
Arcos originales instancia	3720	3720
Eliminados por Prioridad	120.00	120.00
Eliminados por Capacidad	2238.00	2238.00
Eliminados por Ventana de T.	751.45	663.41
Eliminados por Ventana y Precedencia	240.65	207.19
Porcentaje de arcos eliminados	89.95	87.27

Tabla 3.9: Resultados pre-proceso de las instancias Clusterizadas

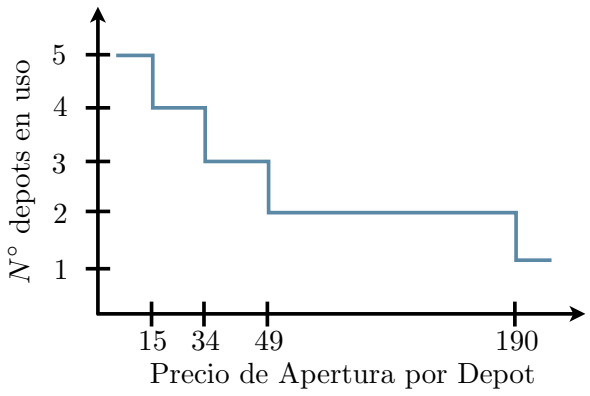


Figura 3.11: Uso de *depots* en solución óptima vs el precio costo fijo de apertura para las instancias de tipo corredor 30Co

El análisis de costos entrega resultados esperables, siendo las instancias con geometría más difícil las más affectadas por el costo de los *depots*. Las instancias *clusterizadas* se ven muy afectadas por el precio del *depot*, puesto que no se puede servir a todos los clientes desde un *depot* central.

Las siguientes tablas muestran los resultados más standard de los modelos de *Branch and Price*, reportando: Z_{IP} mejor solución entera, Z_{LP} solución óptima de la relajación lineal, Gap: diferencia porcentual entre Z_{IP} y Z_{LP}, Cpu Time LP: tiempo empleado en segundos en resolver los diferentes LPs, Cpu Time Pricing: tiempo empleado en segundos es resolver los subproblemas, Depots usados: cantidad de *depots* usados en la solución final, Nodos explorados: cantidad de nodos que exploró el esquema de *branching*, Columnas: cantidad de variables totales agregadas al problema maestro.

El primer cuadro 3.10 es el modelo aplicado a las instancias de 10, 15 y 30 clientes con tolerancia de 2 %, esto quiere decir que si el Gap es menor al 2 %, no se hace el *branching* y se entrega la solucíon en el nodo raíz con un *Branch-and-Bound* de CPLEX. La segunda tabla 3.11, es con Gap de 1 % sobre las instancias que originalmente no lo obtuvieron.

Instancias	Q	W	Z_{IP}	Z_{LP}	Gap %	Cpu Time LP [s]	Cpu Time Pricing [s]	Depots usados	Nodos explorados	Columnas
AA10	15	30	343.37	343.37	0	0.10	8.99	3	1	415
	15	60	343.37	343.37	0	0.10	25.46	3	1	504
	20	30	343.37	343.37	0	0.09	10.50	3	1	415
	20	60	343.37	343.37	0	0.23	36.23	3	1	475
AA15	15	30	483.92	477.99	1.23	0.21	45.65	5	1	958
	15	60	470.20	470.20	0	0.15	25.46	5	1	1114
	20	30	483.92	477.99	1.23	0.16	48.48	5	1	959
	20	60	470.20	470.20	0	0.33	71.39	5	1	1173
AA30	15	30	1068.64	1068.64	0	1.21	132.55	5	1	6298
	15	60	1046.09	1043.31	0.26	3.27	241.84	5	1	17226
	20	30	1070.14	1059.15	1.03	1.10	151.49	5	1	6822
	20	60	1042.12	1031.67	1.01	4.03	291.52	6	1	17686

Tabla 3.10: Resultados para todas las instancias, sin costo de *depot* con GAP<2 %

Instancias	Q	W	Z_{IP}	Z_{LP}	Gap %	Cpu Time LP [s]	Cpu Time Pricing [s]	Depots usados	Nodos explorados	Columnas
AA15	15	30	478.00	477.99	0.00	0.88	47.21	5	26	995
	20	30	478.00	477.99	0.00	0.70	51.55	5	26	995
AA30	20	30	1068.64	1059.15	0.89	3.16	220.01	5	5	6845
	20	60	1033.27	1031.67	0.25	8.33	378.22	5	29	17717

Tabla 3.11: Resultados para todas las instancias, sin costo de *depot* con GAP<1 %

Capítulo 4

Conclusiones

En esta tesis se tratan varios temas relacionados con los problemas de ruteo de vehículos, más precisamente relacionados con el *Pickup and Delivery*. El objetivo principal era formular conjutamente el problema de Localización y Ruteo incluyendo las restricciones propias del problema de *Pickup and Delivery*. Esta tesis ha presentado una nueva formulación para un problema que no existía en la literatura y realizado un estudio del tipo de soluciones.
Los principales aportes de esta tesis son:

- Se presenta una formulación para el problema de Localización y Ruteo con *Pickup and delivery* y Ventanas Tiempo, basada en la formulación expuesta en Berger et al. (2007).
- Se implementó un esquema de generación de columnas con reglas de *branching* adecuadas para resolver instancias de hasta 45 clientes y 7 *depots*.
- Se adaptaron las heurísticas ya existentes para el $PDPTW$ para hacerlas funcionar eficientemente en el PLRPD, además se incluye una heurística nueva (H3) que trabaja como búsqueda local, pero usando el algoritmo de *Label-Setting*.
- Se expone un análisis detallado de la utilización de los *depots* de acuerdo al costo fijo de apertura de estos. Analizando instancias de la literatura (Ropke and Cordeau (2009)) e instancias del tipo clusterizadas y tipo corredor. Adicionalmente se entregan los resultados clásicos de un esquema de *Branch-and-Price*.

Para poder cumplir con todos estos logros, se realizó una revisión detallada de los métodos necesarios para implementar un esquema de generación de columnas en conjunto con un esquema de branching adecuado para el problema de *Pickup and Delivery*. A continuación se formuló un modelo para el PLRPD, diseñando algoritmos de solución específicos para tal formulación. Estas implementaciones fueron validadas con varias instancias recogidas de la literatura, así como a partir de instancias generadas en el contexto de esta tesis para estudiar configuraciones relevantes a este problema.

Como investigación futura sería interesante explorar las siguientes extensiones de nuestro problema:

- Incluir en el problema de *pricing* algoritmos de acotamiento de las rutas parciales. Estos algoritmos han probado ser muy eficientes en el cálculo de rutas con costos reducidos negativos, ver Baldacci et al. (2011).
- Probar estrategias alternativas de *branching* para las rutas.
- Extender esta metología al caso de localización de hubs donde se producen transferencias, y no necesariamente inicios y finales de rutas.

Apéndice

Código en Python

Tanto la implementación del algoritmo de *Branch-and-Price* como las instancias usadas, se pueden descargar de http://www.dim.uchile.cl/ tcapelle/PLRPDP

Bibliografía

D Applegate, R Bixby, V Chvátal, and W Cook. Implementing the dantzig-fulkerson-johnson algorithm for large traveling salesman problems. *Mathematical Programming*, Ser. B(97):91–153, 2003.

Roberto Baldacci, Enrico Bartolini, and Aristide Mingozzi. An exact algorithm for the pickup and delivery problem with time windows. *Operation Research*, 59(2):414–426, 2011.

J.F. Barnhart, E.L. Johnson, G. L. Nemhauser, M.W.P Savelsbergh, and P.H. Vance. Column generation for solving huge integer programs. *Operation Research*, 46:316–329, 1998.

R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(87-90), 1958.

R Berger, C Coullar, and M Daskin. Location-routing problems with distance constraints. *Transportation Science*, 41(1):29–43, 2007.

Jean-Francois Cordeau and Gilbert Laporte. The dial-a-ride problem (darp): Variants, modeling issues and algorithms. *4OR*, pages 89–101, 2003.

Cristián E. Cortés, Martín Matamala, and Claudio Contardo. The pickup and deliery problema with transfer: Formulation and a branch-and-cut solution method. *European Journal of Operational Research*, 2009.

George B. Dantzig and Phil Wolfe. Descomposition principle for linear programs. *Operation Research*, 8:101–111, 1960.

Martin Desrochers and Francois Soumis. A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23(1-13), 1989.

Martin Desrochers, Jaques Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing rpoblem with time windows. *Operation Research*, 40(2):342–354, 1992.

Jaques Desrosiers, Francois Soumis, and Martin Desrochers. Routing with time windows by column generation. *NETWORKS*, 14:545–565, 1984.

Jaques Desrosiers, Yvan Dumas, and Francois Soumis. A dynamic programing solution of the large-scale single-vehicle dial-a-ride problem with time windows. *American Journal of Mathematics And Management Sciences*, 6(3):301–325, 1986.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1 (269-271), 1959.

M. Dror. Note on the complexity of the shortest path models for column generation in vrptw. *Operation Research*, 42(5):977–978, 1994.

Yvan Dumas, Jaques Desrosiers, and Francois Soumis. The pickup and delivery with time windows. *European Journal of Operational Research*, 54:7–22, 1991.

D. Feillet. A tutorial on column generation and bracn-and-price for vehicle routing problems. *Operation Research*, 8:407–242, 2010.

L. Fischer. The lagrangian relaxation method for solving integer programming problems. *Management Science*, 27, 1981.

P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operation Research*, 9(6):849–859, 1961.

E.L. Johnson, G. L. Nemhauser, and M.W.P Savelsbergh. Progress in linear programming-based algorithms for integer programming: An exposition. *INFORMS Journal on Computing*, 12(1), 2000.

Ismail Karaoglan, Fulya Altiparmak, Imdat Kara, and Berna Dengiz. The location-routing problem with simultaneous pickup and delivery: Formulations and a heuristic approach. *Omeg*, 40(465-477), 2012.

G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1988.

M. Padberg and G. Rinaldi. A branch-and-cut approach to a traveling salesman problem with side constraints. *MANAGEMENT SCIENCE*, 35(11):1393–1412, 1989.

Sophie Parragh, Karl Doerner, and Richard Hartl. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58:21–51, 2008. ISSN 0344-9327. URL http://dx.doi.org/10.1007/s11301-008-0033-7. 10.1007/s11301-008-0033-7.

Harilaos N. Psaraftis. A dynamic programing solution to de single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154, 1980.

Stefan Ropke. *Heuristic and exact algorithms for vehicle routing problems*. PhD thesis, 2005.

Stefan Ropke and Jean-Francois Cordeau. Branch-and-cut-and-price for the pickup and delivery problem with time windows. 2005.

Stefan Ropke and Jean-Francois Cordeau. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, 43(3):267–286, August 2009.

K.S. Ruland and E.Y. Rodin. The pickup and delivery problem: Faces and branch-and-cut algorithm. *Computers and Mathematics with Applications*, 33(12):1–13, 1997.

M.W.P Savelsbergh and M Sol. The general pickup and delivery problem. *Transportation Science*, 29(1):17–29, 1995.

M.W.P Savelsbergh and M Sol. Drive: Dynamic routing of independent vehicles. *Operation Research*, 46(474-490), 1998.

T.R. Sexton and Y.M. Choi. Pickup and delivery of partial loads with "soft" time windows. *Am J Math Managagement Science*, 6(369-398), 1986.

Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, 2005.

Paulo Toth and Daniele Vigo, editors. *The Vehicle Routing Problem*. SIAM, 2002.

M. H. J. Webb. Cost functions in the location of depot for multiple delivery journeys. *Operation Research Quaterly*, 19:311–320, 1968.

L.A. Wolsey. *Integer Programming*. Wiley-Interscience Publication, 1998.

H. Xu, Z.L. Chen, S. Rajagopal, and S. Arunapuram. Solving a practical pickup and delivery problem. *Transportation Science*, 37(347-364), 2003.