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

Problema de ruteo de vehículos con ventanas de tiempo, opciones de entrega y capacidades dinámicas

Tesis de Licenciatura en Ciencias de la Computación

Brian Bohe

Director: Dr. Juan José Miranda-Bront
Codirector: Lic. Gonzalo Lera-Romero
Buenos Aires, mayo de 2021

PROBLEMA DE RUTEO DE VEHÍCULOS CON VENTANAS DE TIEMPO, OPCIONES DE ENTREGA Y CAPACIDADES DINÁMICAS

El reciente crecimiento del e-commerce trae consigo un abrupto incremento en la cantidad y complejidad de los desafíos con los que tienen que lidiar las empresas que participan en la logística de última milla. Entre ellos, los inconvenientes de acceso al lugar de destino de la entrega y una limitada disponibilidad del receptor, aumentan la cantidad de intentos de entrega fallidos e impactan directamente en la eficiencia. En este contexto, muchas empresas optan por soluciones híbridas que consideran por un lado la opción de entregar directamente al domicilio del cliente, así como también utilizar puntos de entrega compartidos, llamados locker-boxes, a los que el cliente luego se acerca para retirar sus pedidos. Esta nueva modalidad que en algún grado ayuda a resolver las problemáticas mencionadas, presenta nuevos desafíos desde la gestión relacionados a la capacidad máxima en la recepción de productos, resultante de las características físicas de espacio disponible y de las entregas y retiros que se realicen durante el horizonte de planificación.

Esta variante del conocido problema de ruteo de vehículos, ha sido introducida recientemente y todavía es muy incipiente lo que hay. Si bien se han analizado distintos aspectos, como la utilización de ventanas de tiempo y la heterogeneidad de los puntos de entrega compartidos, hasta ahora se consideró que la capacidad de recepción es constante durante el horizonte de planificación; mientras que en la realidad, es variable. Aunque la naturaleza de esta información es estocástica, se la puede considerar conocida y determinista para afrontar la resolución del problema que la engloba, pudiendo en la práctica ser reemplazada por el resultado de un pronóstico. En este trabajo presentamos una solución Branch & Price para el problema de ruteo de vehículos con ventanas de tiempo, opciones de entrega y capacidades dinámicas, utilizando heurísticas para acelerar los tiempos de cómputo incurridos en el problema de pricing. También, mostramos que considerar las capacidades de recepción dinámicas en los modelos, permite reducir el costo logístico hasta un 37% en el mejor caso, en detrimento de mayores tiempo de cómputo.

Keywords: ruteo de vehículos, generación de columnas, branch and price, opciones de entrega, puntos de entrega múltiples, locker boxes, última-milla, capacidades dinámicas, retiro de paquetes.

VEHICLE ROUTING PROBLEM WITH TIME WINDOWS, DELIVERY OPTIONS AND DYNAMIC CAPACITY

The recent growth of e-commerce brings with it an abrupt increase in the number and complexity of the challenges that companies involved in last mile logistics have to deal with. Among them, the access issues to the place and the restricted receiver availability, increase the number of failed delivery attempts and have a direct impact on efficiency. In this context, many companies opt for hybrid solutions that consider, both the option of delivering directly to the customer's home, and the option of using shared delivery locations, called locker-boxes, to which the customer then approaches to collect their orders. This new modality, which to some degree helps to solve the problems mentioned, presents a new limitation, which is the maximum capacity of orders reception, resulting from the physical characteristics of the available space and from the deliveries and withdrawals that happen during the planning horizon.

This variant of the well-known vehicle routing problem has been introduced and extensively studied in recent years. Although different aspects have been analyzed, such as the use of time windows and the heterogeneity of shared delivery points, until now it has been considered that the reception capacity is constant during the planning horizon; while in reality, it is variable. Although the nature of this information is stochastic, it can be considered known and deterministic to face the resolution of the problem that encompasses it, and in practice it can be replaced by the result of a forecast. In this paper we present a Branch & Price solution for the vehicle routing problem with time windows, delivery options and dynamic capacities, using heuristics to accelerate the computation times incurred in the pricing problem. We also show that considering the dynamic reception capacities in the models, we can reduce the logistic cost by up to 37% in the best case, to the detriment of longer computation times.

Keywords: vehicle routing problem, column generation, branch and price, delivery options, shared delivery locations, locker-boxes, last-mile, pick-ups, dynamic capacity.

AGRADECIMIENTOS

Descubrí esta carrera que tanto me gusta gracias a que mi familia me mandó a una secundaria técnica, me incentivó a competir y a buscar un poco más. Gracias a ellos pude estudiar sin la obligación de trabajar, y sin preocuparme por nada. Me ayudaron a despertarme, me aguantaron todo y me acompañaron cuando no supe aceptar otra opción.

Gracias Fran, Nacho, JC, Juani, Francisco, Veri y el resto del grupo de *Walter y los 40 ladrones* por haber hecho que estos años, en especial en la facultad y el trabajo, me hayan resultado mucho más divertidos.

Gracias Ceci por involucrarte en mi vida. Tu influencia, paciencia y amor fueron clave estos años, incluso para terminar esta tesis y mi carrera.

Gracias a Juan y Gonzalo por ser mis directores, por sus guías en todo el proceso, sus explicaciones y el tiempo invertido. También a Francisco, quien participó de la misma forma durante todo un año. A lo largo de la tesis hablo de nosotros, porque no sería lo que es sin ustedes.

Por último quiero agradecerles a Isabel y Daniel, por aceptar ser jurados de esta tesis y dedicarle tanto tiempo a descubrir nuestro trabajo. Más allá de que tengan una clara trayectoria en el área y lo que significan para mí sus opiniones, son unas de las pocas personas que van a leer este documento completo, y lo valoro mucho.

CONTENTS

1. INTRODUCCIÓN

El objetivo de este capítulo es introducirse en el área, con sus conceptos y nomenclaturas, llegando al estado del arte del problema al final del capítulo. Primero, hacemos una breve introducción a los *problemas de ruteo de vehículos*. Luego introducimos y detallamos los famosos *envíos indirectos*, comentando sus características, objetivos e implementaciones en la realidad. Por último, presentamos el aporte de la academia con un breve recorrido sobre trabajos recientes en los aspectos del problema de ruteo de vehículos con opciones de entrega, culminando en la presentación del enfoque de nuestro trabajo.

1.1 Problemas de ruteo de vehículos

Los Problemas de Ruteo de Vehículos (VRP, por su sigla en inglés) aparecen en la organización de las tareas de distribución de mercadería o personal, planificación de recorridos en robótica móvil o prestación de servicios a un conjunto de clientes mediante una flota de vehículos. Los vehículos realizan sus movimientos a través de una red partiendo de puntos fijos, llamados depósitos. Cada tramo entre dos clientes de esta red tiene asociado un costo y/o tiempo de viaje que puede depender de muchos factores, como por ejemplo del tipo de vehículo o del período durante el cual el tramo es recorrido. Este tipo de problemas es de gran relevancia en empresas de tamaño pequeño a grande, tanto en el sector público como privado. Desde el punto de vista del modelado y resolución, las características de cada aplicación conllevan un desafío particular. Una excelente presentación sobre diferentes variantes de estos problemas puede verse en [12].

La *distribución de última milla* considera el tramo final en el traslado de la mercadería, desde un centro o depósito hasta su destino. Esta etapa ha tomado particular relevancia recientemente, en especial con el crecimiento de las ventas vía canales digitales y el e-commerce. Un escenario frecuente en empresas logísticas de diversa escala es la presencia de depósitos y centros de almacenamiento estratégicamente ubicados en distintos puntos de la ciudad, desde donde los vehículos distribuyen la mercadería hasta su destino final, el domicilio del comprador. En este contexto, el VRP clásico aparece como un modelo natural de esta problemática, donde un centro de almacenamiento representa el depósito, los clientes son los destinos finales de la mercadería (eventualmente con alguna restricción adicional, como ventanas de tiempo), buscando una solución que minimice las distancias (o costos) de distribución sujeta a las restricciones operativas.

Diversos estudios agregados estiman que en los países desarrollados los costos logísticos representan entre un 6–8 % del valor final del bien, y que los mismos tienden a ser un 20–30 % más altos en economías emergentes (en particular, en Latinoamérica y el Caribe). A su vez, se estima que la participación de la última milla representa en muchos casos hasta el 50% de los costos logísticos totales de la mercadería [10], lo que lo convierte en un proceso clave. Una de las razones es que consiste en un proceso complejo, caracterizado por tener destinos en lugares dispersos del mapa, con baja demanda cada uno y una disponibilidad horaria acotada para recibir los envíos. También está influenciado por el tráfico y otras dificultades operativas, como encontrar estacionamiento, devoluciones y errores en los envíos, entre otros. También, la polución del aire generada por el tráfico de la última milla, es hoy en día una preocupación en la sociedad, por lo que al interés

económico de las empresas, se agrega la expectativa de una solución más sustentable con el medio ambiente.

Una de las estrategias para reducir el impacto de estos inconvenientes es la utilización del *envío indirecto*, que consiste en dejar los paquetes en espacios de almacenamiento convenientemente distribuidos en la ciudad, en lugar de ir hasta el domicilio del cliente. Luego, el paquete será recolectado por el cliente por sus propios medios. Desde el punto de vista de la empresa, la logística de distribución de envíos indirectos se combina con la tradicional de última milla, el envío directo al domicilio del comprador. Sin embargo, agrega una nueva capa de consolidación permitiendo agrupar múltiples envíos en un único destino, potencialmente reduciendo el tiempo de distribución de los paquetes respecto a realizar entregas a diferentes domicilios.

La experiencia práctica en el desarrollo de métodos exactos para variantes multi-vehículo de VRP tradicionales indica que los enfoques más efectivos se basan en algoritmos de tipo Branch & Price o Branch-Cut & Price (ver, e.g., [3]). Este tipo de algoritmos se aplica a modelos que poseen un número exponencial de variables (por ejemplo, una variable por cada ruta factible) lo que hace que no sea posible resolver explícitamente la relajación lineal del modelo. En estos casos, las variables se van generando bajo demanda mediante la técnica de generación de columnas. En el caso de VRP, el subproblema de generación de columnas corresponde a resolver un problema de camino mínimo elemental con restricciones de recursos.

1.2 Envíos indirectos

En esta sección extendemos el concepto de envío indirecto, profundizando particularmente en dos tipos del mismo. La información presentada es una recopilación del extenso trabajo publicado por Rhomer y Gendron [11].

Como dijimos previamente, los envíos indirectos, realizados en *puntos de entrega compartidos*, son una alternativa a la tradicional entrega en el domicilio del cliente. La idea general es proveer al cliente de lugares alternativos en donde los pedidos pueden ser recibidos, potencialmente sin necesidad de su supervisión. El paquete se encontrará disponible para la posterior recolección del cliente durante un tiempo máximo, acordado previamente entre todos los agentes que participan en el envío. Dentro de estos agentes, y dependiendo del caso, pueden estar incluidos la empresa que vende el producto, la distribuidora encargada de realizar el envío y el cliente.

Los puntos de entrega compartidos presentan una ventaja para el cliente ya que evita conflictos y responsabilidades, como arreglar y reservar previamente una franja horaria en la que este tiene que permanecer en el domicilio informado de manera ininterrumpida. Hoy en día, este tipo de puntos está comúnmente integrado con tecnologías y protocolos capaces de velar por la seguridad del producto, limitar el acceso al mismo, y brindar un método de retiro fácil y rápido a los clientes. En algunos casos realiza también un seguimiento de los paquetes, informando cuándo está disponible para ser retirado, cuánto tiempo se tiene para hacerlo y cuándo fue retirado, mejorando así la interacción con el cliente.

Comparados con las entregas a domicilio, los envíos indirectos ofrecen mayor flexibilidad horaria en términos de cuándo puede hacerse la entrega del pedido. El hecho de poder entregar varios pedidos en un mismo punto, simplifica el recorrido que tiene que hacer el chofer respecto a múltiples entregas a domicilio, potencialmente reduciendo la

distancia total recorrida, los costos, el tráfico al que está expuesto el vehículo y la cantidad de este que genera. También permiten un mayor aprovechamiento de la capacidad de los vehículos, ya que los paquetes que se entregan en un mismo punto pueden almacenarse juntos dentro del espacio de carga del vehículo. Por último, este tipo de envío no presenta casi intentos fallidos de entrega, algo muy común en las entregas a domicilio según el relevamiento de [11]. Por todo esto, esta opción es significativamente menos costosa que las entregas estándar a domicilio, generando en los últimos años una proliferación de inversiones, implementaciones, empresas y servicios relacionados.

Los destinos utilizados para los envíos indirectos se categorizan principalmente en *puntos de entrega compartidos* y *lockers automatizados*. Los primeros consisten en pequeños almacenes atendidos por personas, generalmente presentes en tiendas, y por lo tanto con un horario de funcionamiento restringido por estas. Este tipo de lugares pueden requerir de la construcción o modificación de edificios, por lo que la decisión de cuántos y dónde ubicarlos conlleva una planificación a largo plazo, de generalmente décadas.

Los lockers automatizados consisten de uno o más módulos de casilleros, operados por una misma terminal lógica, similar a la que encontramos en cajeros automáticos. Un tipo de módulo de lockers tiene ciertas características, como la cantidad de casilleros que tiene para almacenar los paquetes, el volumen de estos casilleros, el material del que están compuestos, y si tiene tecnología para controlar la temperatura. Un mismo módulo puede incluso tener más de un tipo de casilleros. Dónde colocar el locker, cuántos módulos incluir y con qué características, depende de las necesidades que este vaya a suplir. Por ejemplo, si el objetivo es que se reciban o almacenen productos comestibles que necesitan mantener una cadena de frío como chocolates, golosinas y lácteos, se necesita al menos un módulo de lockers que pueda trabajar con temperaturas frías. A diferencia de los puntos de entrega compartidos, los lockers pueden estar disponibles las 24 horas del día, y los clientes retiran sus paquetes sin asistencia de empleados. Al mismo tiempo, planificar dónde ubicarlos es más sencillo dado que ocupan menos espacio, son más fáciles de transportar y el proceso de instalación es más rápido que con los puntos de entrega compartidos tradicionales. Esto hace que puedan adaptarse a los cambios de urbanización de las ciudades, y que su horizonte de planificación sea de años y no décadas. Respecto a las empresas, esta alternativa presenta una mayor disponibilidad y flexibilidad para operarlos, en detrimento de una capacidad más restrictiva y limitada, como también una mayor inversión inicial. La capacidad es más restrictiva porque no todos los paquetes pueden ser almacenados en los lockers, por ejemplo por una restricción de tamaño. También, es más limitada porque el almacén de un punto de entrega compartido generalmente dispone de más espacio.

Si bien los lockers automatizados aún no reemplazan a los puntos de entrega compartidos, los beneficios en la operatoria y el ahorro en costos a largo plazo, hacen que sea una de las alternativas que más inversión esté recibiendo, y en dónde está puesto el foco de las empresas de logística hoy en día. Tanto es así, que es normal encontrar que múltiples agentes dentro de la cadena de suministro de productos implementan algún tipo de red de lockers.

Por un lado, tenemos las grandes distribuidoras y servicios de correo postal con sus propias redes de lockers, generalmente ubicados en estaciones de transporte público, supermercados y oficinas postales. Estos lockers son diseñados, planificados y manipulados exclusivamente por las empresas dueñas de los mismos, quienes deciden también si los clientes pueden solo retirar o al mismo tiempo dejar paquetes. Ejemplos de estos son *DHL* y *Australia Post*.

Fuera de las distribuidoras encontramos muchos agentes que decidieron invertir en lockers. Un caso muy destacado es el de las empresas de e-commerce, que con la intención de mejorar su operatoria, darle una mayor flexibilidad a sus usuarios y mejorar la experiencia de compra, están invirtiendo en formar su propia red de lockers. Estas empresas no tienen una red logística en sí, por lo que los paquetes generalmente llegan al locker de la mano de la distribuidora elegida por ellas o por el cliente, entre las opciones disponibles al realizar la compra. Este es el ejemplo de empresas como *Amazon*, con una red de lockers hoy presente en al menos Reino Unido, Canadá, Francia y los Estados Unidos, donde le permiten a sus clientes tanto retirar como devolver paquetes.

Al mismo tiempo que las empresas de e-commerce, los retailers o grandes cadenas de supermercados se volcaron a la venta por internet, que se desarrolló en coexistencia con su modelo de negocio presencial. Estos encontraron en los lockers una forma de poder disponer los productos más cerca de los usuarios, y son responsables de manipularlos y de reponer el stock de los productos en estos. Un ejemplo de esto es EDEKA en Alemania con su venta online de comida.

La demanda de lockers es tan grande que dio el pie al nacimiento de compañías que construyeron sus propias redes de lockers y hoy las ofrecen como servicios. Esto está destinado a todo tipo de distribuidoras y empresas que no cuentan con una red de lockers propia y/o quiera extender su red. Los contratos para usar este tipo de servicio incluyen, desde un mínimo número de casilleros reservados para el cliente en todo momento, hasta lockers exclusivos. También, hay esquemas de alquileres compartidos entre distintos clientes, con precios que varían dependiendo del largo del período utilizado y los tipos de rangos de horarios, por ejemplo si se utiliza en hora pico. Como resultado, un mismo locker puede ser utilizado por varias compañías. Con esta misma idea de negocio, las empresas de transporte público, como MVG, están comenzando a ofrecer su propia red de lockers como servicio. En este caso estos funcionan como una mezcla entre puntos de entrega y depósitos de almacenaje.

En el contexto de envíos indirectos, los problemas de ruteos de vehículos incluyen decisiones como (1) elegir uno de los destinos alternativos para cada pedido y (2) dónde ubicar los puntos de entrega compartidos o lockers automatizados. En lo que respecta a la primera decisión es donde encontramos el problema de ruteo de vehículos con opciones de entrega (o VRPDO por sus siglas en inglés). En este tipo de problemas, los puntos de entrega compartidos y las redes de lockers, proveen diferentes puntos geográficos en los cuales se puede entregar un paquete y a donde el cliente se acerca luego para retirarlo. Es de esta forma que la dirección de entrega de un pedido puede no estar fija de antemano y ser ahora una variable de decisión. Los puntos de entrega compartidos poseen diferentes características respecto a la entrega tradicional al domicilio del cliente. Si bien estos relajan la ventana de tiempo asociada a la entrega a domicilio, introducen nuevos tipos de restricciones y desafíos, principalmente por su capacidad.

Las restricciones de capacidad están relacionadas a dos cosas. Por un lado está la cantidad total de pedidos que pueden ser entregados en el locker. Esta está relacionada a su capacidad máxima, la cantidad de casilleros totales, y al porcentaje de ocupación de los mismos. También, la disponibilidad de módulos o casilleros puede variar en base a las características específicas que se necesitan. Por ejemplo, puede ser que hayan menos casilleros o lockers con control de temperatura, o pocos casilleros con un volumen grande.

Por último, dependiendo del modelo de negocio, las empresas pueden incurrir en un costo adicional por alquilar un casillero específico, o un locker completo, a un proveedor de

lockers por un tiempo determinado. En otros casos puede que la empresa tenga su propia red de lockers, no teniendo que pagar entonces por su uso, y quizás considere compensar a los usuarios por elegir un locker de esta red.

1.3 Revisión de la Literatura

VRPDO ha sido recientemente estudiado en [9], [1] y [8]. En todos estos trabajos, se considera que un pedido puede tener más de un punto de entrega, los cuales son elegidos por el cliente o por la empresa distribuidora. A los puntos de entrega que tienen un solo pedido asociado se los conoce como *entregas atendidas en domicilio* (AHD por sus siglas en inglés), y hacen comúnmente referencia las entregas en hogares. A los puntos de entrega que pueden tener más de un pedido asociado se los conoce como *puntos de entrega compartidos* (SDL por sus siglas en inglés), y engloban ambos tipos de envíos indirectos que detallamos en la sección 1.2. Se considera siempre que hay un solo depósito del cual inician todos los vehículos su trayectoria, y al cual tienen que volver al concluir con la entrega de sus pedidos.

Una restricción que se considera en Mancini et al [9] y Tilk et al [1] es la relacionada a la cantidad máxima de vehículos que se pueden utilizar. En ambos casos además se tiene en cuenta un costo fijo por la utilización del vehículo, que está incluido en el cálculo del costo de la ruta. Y ambos consideran que este es mayor en órdenes de magnitud respecto a otros costos, así predominante en el costo de la ruta. De esta manera las funciones objetivo de los problemas, que minimizan la suma de los costos de las rutas seleccionadas en la solución, minimizan primero la cantidad de vehículos utilizados y luego el costo de la ruta en sí.

Mancini et al. [9] considera que cada cliente informa una alternativa AHD para la entrega de su pedido, y todos los SDL disponibles pueden ser utilizados como puntos de entrega alternativos, pagando una compensación al cliente por el esfuerzo que este tiene que hacer para acercarse a uno de estos puntos a retirar su pedido. Cuando un repartidor llega a un punto SDL, incurre en un tiempo de preparación por todas las entregas que hace, que incluye el tiempo de estacionar el vehículo y lo que normalmente se conoce como tiempo de servicio. Luego deja la cantidad de pedidos que desee, estando limitado por la capacidad del SDL. También considera que la capacidad de recepción de un SDL es inferior a la capacidad de transporte del vehículo. De esta forma, alcanza con un solo vehículo para transportar todos los pedidos que se desean entregar en un mismo SDL, lo que significa que en una solución óptima, no habrá más de un vehículo dejando pedidos en un SDL. El trabajo propone por un lado una formulación compacta de programación lineal entera mixta y la compara con una meta-heurística basada en Large Neighborhood Search. Muestra como a medida que el tamaño de la entrada aumenta, la implementación de la formulación compacta se vuelve inviable por sus tiempos de cómputo, sin poder resolver instancias con más de 25 pedidos. En contraposición, su meta-heurística consigue rápido soluciones considerablemente buenas, resolviendo instancias de hasta 75 pedidos.

Por otro lado, Tilk et al [1] considera que todas las opciones de entrega alternativas para un mismo pedido son definidas por el cliente, y la empresa distribuidora tiene que elegir entre alguna de estas opciones. Estos puntos pueden ser tanto AHD como SDL, y cada uno tiene una preferencia o prioridad asociada. Luego, un mínimo nivel de calidad de servicio es requerido en la solución del problema. Por ejemplo, que la mitad de los clientes sean abastecidos con su opción de máxima prioridad, y que $\frac{3}{4}$ de los clientes con alguna de

sus dos opciones de máxima prioridad. Esto es modelado a través de restricciones que se agregan a la formulación de un modelo de programación lineal entera. Se considera que un SDL puede ser visitado múltiples veces en una misma ruta, incurriendo en cada ocasión en un tiempo de estacionamiento asociado, y también en un tiempo de servicio pero en este caso por cada pedido que es entregado en el SDL. Luego, a diferencia de Mancini et al [9], el tiempo de permanencia de un vehículo en el SDL depende de la cantidad de pedidos a entregar en él. Se propone una solución Branch-Cut-Price que utilizamos como punto de referencia en nuestro trabajo. También utiliza heurísticas para resolver el problema de pricing. Proponen dos formas de modelar la red de puntos de entrega, una que considera todas las alternativas de envíos como independientes, y otra que aprovecha el hecho de tener varias pedidos que pueden ser entregados en un mismo punto geográfico. Nosotros utilizaremos esta última.

Por último, [8] considera que cada cliente informa un punto AHD donde quiere recibir su pedido, y cero o más puntos SDL a los cuales está dispuesto a aproximarse para retirar su pedido recibiendo como compensación un descuento. A diferencia de los otros trabajos, se asume que la cantidad de vehículos es ilimitada y que su capacidad no es una limitante. Lo innovador de este trabajo es que asume que los lockers no son homogéneos. En cambio, cada locker dispone de distintos tipos de casilleros, y no todos los pedidos pueden ser almacenados en todos los casilleros. Propone un proceso heurístico iterativo, que consiste en resolver primero el problema de ruteo solo con los puntos AHD, es decir, sin utilizar ningún SDL. Luego, se eligen los pedidos que serán atendidos en puntos SDL, y se resuelve nuevamente el problema de ruteo ahora con esta consideración. Después de resolver cada problema de ruteo de vehículos, se optimiza el uso de los casilleros en los lockers resolviendo un problema de bin-packing. Para resolver el problema de ruteo y el problema de bin-packing se utilizan respectivamente Adaptive Large Neighborhood Search y una formulación heurística del problema de Set Covering utilizando programación lineal entera.

A pesar de que en los últimos años hubo una clara tendencia en la literatura de los problemas de ruteo de vehículos sobre aspectos relacionados a VRPDO, aún hay pendientes grandes desafíos sin estudiar. Por ejemplo, en todos los trabajos mencionados las capacidades de los puntos SDL, que es limitada y una de sus mayores desventajas, en especial en el caso de los lockers, se considera constante durante todo el horizonte de planificación. Aunque en realidad, esta puede variar. Modelar los retiros de paquetes podría tener entonces un gran impacto en la eficiencia y el uso de este tipo de puntos, y de aquí viene la importancia de estudiar alternativas en las que se consideren que la capacidad de estos puntos pueda incrementar.

La estructura de esta tesis es la siguiente. Primero, en el capítulo 2, se realiza una breve introducción a las técnicas utilizadas en la solución del problema. A continuación, comentamos el enfoque de este trabajo en el capítulo 3, presentando formalmente el problema que se quiere resolver con su respectiva formulación. El capítulo 4 detalla la solución Branch & Price propuesta, explicando primero como se modela y resuelve la fase de generación de columnas, seguido del esquema de branching utilizado. Realizamos una serie de experimentos para obtener una buena configuración de la solución. Esta consiste en determinar aspectos como qué heurísticas utilizar, en qué orden y con qué parámetros. Los pasos más importante los registramos en el capítulo 5, donde también mostramos los beneficios obtenidos por utilizar capacidades dinámicas. Finalmente el capítulo 6 resume las conclusiones aprendidas y lista opciones de trabajos a futuro.

2. PRELIMINARES

En este capítulo, introducimos brevemente los conceptos necesarios para poder entender las técnicas utilizadas en este trabajo. Iniciamos explicando resumidamente qué es la programación lineal con algunos ejemplos y continuamos con la explicación de la programación lineal entera, y sus diferencias. Proseguimos con la explicación de un algoritmo para resolver este tipo de problemas llamado *branch and bound*, y finalmente explicamos en qué se basa la técnica de generación de columnas. Ambos algoritmos son la base para comprender un algoritmo *branch and price*.

2.1 Programación lineal

La programación lineal es una técnica para resolver problemas de optimización. Estos consisten en minimizar (o maximizar) una función lineal, llamada función objetivo, cuyas variables están sujetas a un conjunto de restricciones, también lineales.

Un problema de programación lineal (PPL) puede formularse en forma canónica como:

$$\begin{aligned} &\min c^t\ x \\ &\text{sujeto a } A\ x \geq b \\ &x \geq 0 \end{aligned} \tag{2.1}$$

donde:

- $x \in \mathbb{R}^n$ es un vector de variables no negativas, y es objetivo del problema determinar su valor.
- $c^t\ x$ es conocida como la función objetivo, formada por el producto de un vector de coeficientes $c \in \mathbb{R}^n$ y el vector de variables. Es lo que se quiere minimizar.
- $A\ x \geq b$ son las restricciones, formadas por una matriz de coeficientes $A \in \mathbb{R}^{m \times n}$ que indican la participación de cada variable en cada restricción y un vector de términos independientes $b \in \mathbb{R}^m$. La i-ésima restricción del modelo no es otra cosa que $A_i\ x \geq b_i$, donde A_i es la i-ésima fila de A.

Cada restricción impone una condición que debe ser cumplida por las variables que participan en ella. Decimos que un vector $\bar{x} \in \mathbb{R}^n$ es una solución factible del PPL cuando satisface todas las restricciones, es decir, $A\ \bar{x} \geq b \wedge \bar{x} \geq 0$. El conjunto de soluciones factibles, también llamado región factible, es el espacio de valores $\{x \in \mathbb{R}^n | Ax \geq b \wedge x \geq 0\}$ que puede tomar el vector de variables x del PPL, sobre el cual se busca minimizar (o maximizar) la función objetivo. Cuando la región factible es vacía, decimos que el problema es *infactible*.

Una propiedad interesante de los PPL es que se pueden escribir de muchas formas: como una maximización, con ecuaciones, inecuaciones en distintos sentidos y con distintas variables. Aún así, siempre tienen una forma canónica equivalente.

A modo de ejemplo, supongamos que se quiere preparar al menos 2 litros de una solución para limpiar piletas de natación. Contamos con una cantidad ilimitada de dos químicos, en estado líquido, que deben mezclarse para obtener dicha solución. Por regla

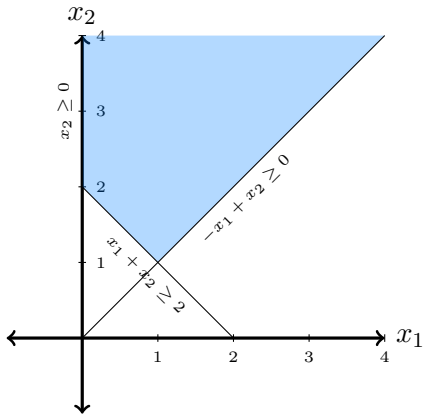

Fig. 2.1: Restricciones y región factible de la formulación 2.2

de la secretaría de limpieza, debe haber una cantidad de litros del segundo químico mayor o igual que la cantidad de litros del primero. Sabiendo que el precio de los químicos es de \$300 y \$500 respectivamente, queremos calcular cuantos litros de cada químico hay que utilizar en la solución de forma que minimicemos el costo total.

Este problema puede resolverse utilizando programación lineal. Siendo x_1 y x_2 la cantidad de litros del primer y segundo químico respectivamente, la siguiente es una formulación de un PPL que captura las reglas de nuestro problema:

$$\begin{aligned} \min\ & 300\ x_1 + 500 x_2 && && (2.2)\\ \text{sujeto a}\ & -x_1 + x_2 && \geq 0 && (2.3)\\ & x_1 + x_2 && \geq 2 && (2.4)\\ & x_1, x_2 && \geq 0 && (2.5) \end{aligned}$$

donde (2.3) impone que se utilice al menos tantos litros del segundo químico como del primero, y (2.4) que entre ambos químicos haya al menos 2 litros. Utilizando x_1 como el eje de abscisas y x_2 como el eje de ordenadas, la figura 2.1 muestra en celeste una porción de la región factible de la formulación anterior.

Dado que esta formulación busca minimizar una función lineal (minimizar el costo asociado a los litros de los químicos), y cuenta así mismo con restricciones lineales en las que participan exactamente las mismas variables (los litros de ambos químicos), se pueden combinar las restricciones (2.3) y (2.4) para acotar inferiormente la función objetivo. Como no sabemos cómo formar dicha cota, introducimos dos nuevas variables $y_1, y_2 \in \mathbb{R}_{\geq 0}$ que tienen como objetivo establecer dicha cota. Multiplicando la restricción (2.3) por la variable $y_1 \geq 0$, la restricción (2.4) por la variable nueva $y_2 \geq 0$, y sumando ambas desigualdades, resulta:

$$x_1(-y_1 + y_2) + x_2(y_1 + y_2) \geq 2\ y_2 \tag{2.6}$$

y ahora si $300 \geq -y_1 + y_2$ y $500 \geq y_1 + y_2$, vale que $2\ y_2$ es una cota inferior de la función objetivo. Reescribiendo estas condiciones sobre las variables y_1 e y_2, y buscando el máximo

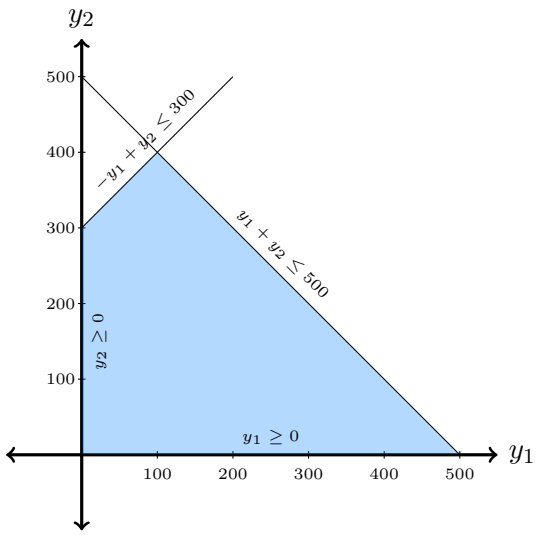

Fig. 2.2: Restricciones y región factible de la formulación 2.7

valor que puede alcanzar la expresión 2 y_2, encontramos una nueva formulación:

$$\begin{aligned} \max\ & 2\ y_2 && \\ \text{sujeto a}\ & -y_1 + y_2 && \leq 300 \\ & y_1 + y_2 && \leq 500 \\ & y_1, y_2 && \geq 0 \end{aligned} \tag{2.7}$$

cuya región factible, como podemos ver en la figura 2.2, es distinta de la correspondiente a las variables x_1 y x_2.

Decimos en este caso que el PPL de la formulación 2.7 es el *problema dual* de la formulación 2.2, que denominamos *problema primal*. De la misma forma decimos que x_1 y x_2 son *variables primales*, y que y_1 e y_2 son las *variables duales*.

De manera general, dado la formulación 2.1 de un PPL, descrita como una minimización y referida como problema primal, su problema dual se formula como .

$$\begin{aligned} \max\ & b^t\ y \\ \text{sujeto a}\ & A^t\ y \leq c \\ & y \geq 0 \end{aligned} \tag{2.8}$$

donde $y \in \mathbb{R}^m$ es un vector de variables duales no negativas cuyo valor hay que determinar.

Una observación importante es que un PPL es resoluble de manera eficiente. A pesar de conocerse algoritmos polinomiales para resolverlo, en la práctica es común utilizar *Simplex* por su desempeño, aunque su complejidad sea exponencial en el peor caso.

2.2 Programación lineal entera

Otro tipo de problemas similares también pueden ser formulados con una función objetivo y restricciones lineales como un PPL, pero cuentan con la característica de poseer variables discretas, en lugar de continuas. Cuando todas las variables son discretas, decimos que

estamos frente a un problema de programación lineal entera (PPLE), y cuando hay ambos tipos, problema de programación lineal entera mixta (PPLEM).

A modo de ejemplo, supongamos que una carpintería quiere decidir cuantas sillas y mesas debe fabricar de forma que gane la mayor cantidad de dinero. Dado que los productos deben estar terminados correctamente para poder venderse, las cantidades fabricadas de cada producto tienen que ser números enteros no negativos.

Para un segundo ejemplo, supongamos ahora que hay que comprar herramientas para fabricar ambos productos, y que las herramientas para fabricar una silla no sirven para fabricar una mesa, y viceversa. Dado que no podemos incurrir en el gasto de comprar las herramientas para ambos productos, necesitamos saber si conviene fabricar sillas o mesas. En este caso lo que queremos obtener, es una variable de decisión que nos indique si debemos comprar las herramientas para hacer sillas o mesas, comúnmente interpretada con los valores binarios 0 y 1.

La gran diferencia entre los PPLE y los PPL, es que los primeros pertenecen a la clase de problemas NP-hard, por lo que no se conoce un algoritmo polinomial que los resuelva en forma general.

Algunos algoritmos utilizados para resolver PPLE, utilizan el concepto de *relajación*. Esta consiste en quitar o modificar restricciones, ampliando el espacio de soluciones posibles, de forma tal que todas las soluciones factibles del PPLE estén contenidas en el nuevo problema. Aunque sea contradictorio, agrandar el espacio de soluciones puede llevarnos a una formulación más fácil de resolver. Un ejemplo de esto es relajar las condiciones sobre las variables de tomar valores enteros o binarios, conocidas como condiciones de integralidad, cambiando así el dominios de las variables de enteros a reales. Esto se conoce como *relajación lineal* y el problema resultante es un PPL.

En caso de ser un problema de minimización, un óptimo de la relajación lineal alcanza un valor de la función objetivo menor o igual que toda solución (entera) factible del PPLE del que se generó. Esto ocurre porque la región factible del PPL incluye al conjunto de soluciones factibles del PPLE. Cabe destacar que, por el mismo argumento, una solución factible del PPL, puede no ser una solución factible del PPLE, si no satisface alguna de las condiciones de integralidad. La diferencia entre los valores óptimos del PPL y del PPLE, se la conoce como *gap de integralidad* y suele utilizarse como una medida de la calidad de la relajación. Cuando no se conoce el valor óptimo del PPLE, porque aún no se ha resuelto el problema, se puede acotar superiormente este gap utilizando el valor óptimo de la relajación lineal y el menor valor, si se trata de una minimización, alcanzado entre todas las soluciones factibles del PPLE que hayan sido exploradas hasta el momento. Esto nos brinda también una medida de calidad sobre la solución entera que utilizamos para esta cota, como también una prueba de que es un óptimo del PPLE, cuando el gap es 0.

Un algoritmo utilizado para resolver PPLE, consiste en resolver primero su relajación lineal. Si el óptimo de la relajación lineal satisface las condiciones de integralidad, es también un óptimo del PPLE. De lo contrario, dividimos el conjunto de soluciones del PPLE en dos o más conjuntos disjuntos utilizando hiperplanos, de forma que la solución de la relajación no esté incluida en ninguno de estos, y repetimos el proceso sobre cada uno de estos. La mejor solución encontrada entre todos estos subconjuntos, es la mejor solución del PPLE inicial, ya que todas las soluciones de este están en alguno de estos conjuntos.

Este esquema recursivo puede interpretarse como un árbol enraizado en el PPLE inicial. Cada nodo del árbol tiene como objetivo encontrar un óptimo en su conjunto de soluciones,

y sus ramas son las divisiones de su conjunto de soluciones. Cada nodo en el árbol, excepto la raíz, es un subproblema de su ancestro.

El proceso de dividir el conjunto de soluciones en dos o más conjuntos, se lo conoce como *branching*, que en nuestra interpretación del árbol, se corresponde con agregar en un nodo ramas a otros nodos, hijos del primero, en el árbol. Cada nuevo conjunto representa un nuevo nodo en el árbol, como así también un nuevo subproblema, resultado de agregar una desigualdad al problema del nodo padre. Las desigualdades agregadas son una combinación lineal de las variables, dado que sigue siendo un sistema lineal.

Por ejemplo, cada vez que la relajación lineal no es una solución factible del PPLE, podemos dividir el problema en dos subproblemas, en base a la variable más fraccionaria sobre la que hay una condición de integralidad. Es decir, dado un nodo que representa al PPLE p de n variables, $\bar{x} \in \mathbb{R}^n$ una solución óptima de la relajación lineal de p y x_v la variable con valor más fraccionario en $\bar{x}$, generamos dos PPLE p_1 y p_2 idénticos a p, y le agregamos la desigualdad $x_v \leq \lfloor \bar{x}_v \rfloor$ a p_1 y $\lceil \bar{x}_v \rceil \leq x_v$ a p_2. En nuestro árbol el nodo p pasa a tener dos ramas que llevan a sus dos subproblemas p_1 y p_2, correspondientes a dos PPLE nuevos que describen una partición de la región factible de p y que en conjunto contemplan todas las soluciones enteras de p.

De la misma forma que hacemos podas en un árbol de backtracking, podemos evitar seguir explorando un nodo de este árbol cuando:

1. La relajación lineal del PPLE es infactible. Como no hay soluciones factibles de la relajación, por lo que tampoco hay del PPLE.

2. El valor óptimo de la relajación lineal es igual o peor, que el obtenido al haber resuelto otro PPLE. En este caso ya hemos resuelto el subproblema correspondiente a algún nodo del árbol, y obtenido una solución entera factible de esta. Como esta solución factible es mejor o igual que el óptimo de la relajación del nodo actual, lo es también respecto a las soluciones del PPLE actual.

3. El óptimo de la relajación lineal cumple las condiciones de integralidad. Esto equivale a haber resuelto un subproblema del nodo.

Este algoritmo se lo conoce como *branch and bound*, por sus etapas de exploración extendiendo un nodo (branching) y poda (bound). A continuación, el algoritmo 1 detalla un esquema conceptual del procedimiento. Los nodos extendidos del árbol que aún no fueron explorados y podados o extendidos se almacenan en el conjunto *PPLEs*. Z^*, inicializado en infinito, representa la mejor solución hasta el momento del PPLE inicial, mientras que Z es el óptimo de la relajación lineal del PPLE. *DividirPPLE* es el procedimiento de branching que depende de la estrategia elegida.

Un esquema de branching puede considerar distintas familias de inecuaciones en algún orden para dividir el PPLE en subproblemas más pequeños. Lo mínimo que debe cumplir un esquema, es que de no poder agregarse nuevas desigualdades, la solución en cuestión debe ser factible y entera.

2.3 Generación de columnas

En la práctica es posible encontrarse un PPL con un gran número de variables. Y en algunos casos este número es tan grande que no es posible considerarlas todas explícitamente en la formulación.

input : Un PPLE p
output: Z^* valor óptimo de p, o infinito si no tiene solución

1 *Inicializar* PPLEs $= \{p\}$, *y la* $Z^* = \infty$
2 **while** PPLEs $\neq \emptyset$ **do**
3 Tomar y remover un PPLE p de PPLEs
4 $Z \leftarrow$ óptimo de la relajación lineal de p
5 $x \leftarrow$ solución de la relajación lineal de p
6 **if** p *es infactible* **then** Ir a 2
7 **if** $Z > Z^*$ **then** Ir a 2
8 **if** x *satisface las condiciones de integralidad* **then**
9 $Z^* \leftarrow \min(Z^*, Z)$
10 Ir a 2
11 **end**
12 **foreach** *PPLE* p' *resultado de* `DividirPPLE` *(p, Z)* **do** Agregar p' a PPLEs
13 **end**

Algoritmo 1: Esquema conceptual de *branch and bound* sobre un problema de minimización.

Ejemplos de esto son las formulaciones de tipo *Set Partitioning* del problema de ruteo de vehículos, en donde hay una variable binaria por cada ruta factible. A continuación presentamos una formulación de este tipo para el problema estándar de ruteo de vehículos con un solo depósito.

$$\min \sum_{r \in \Omega} c_r \lambda_r \tag{2.9}$$

$$\text{s.t.} \sum_{r \in \Omega} \text{Atención}_{ir} \lambda_r = 1 \qquad i \in \mathit{Clientes} \tag{2.10}$$

$$\lambda_r \in \{0, 1\} \qquad r \in \Omega \tag{2.11}$$

donde c_r es el costo de la ruta $r \in \Omega$, λ_r es la variable asociada a la ruta r, Atención_{ir} para $i \in \mathit{Clientes}$ y $r \in \Omega$ es 1 cuando la ruta r entrega el pedido del i-ésimo cliente y Ω es el conjunto de todas las rutas factibles. Las restricciones (2.10) fuerzan que cada cliente sea atendido exactamente una vez, mientras que (2.11) pide que las variables tomen valores binarios.

En este caso una ruta factible es un ciclo simple. El recorrido de los vehículos inicia y termina en el único depósito, visitando en el medio uno o más clientes. En el VRP estándar hay un único punto en el que podemos entregar el pedido del i-ésimo cliente, por lo que al pedir que la ruta sea un ciclo simple estamos diciendo que el vehículo visita a lo sumo una vez cada domicilio, o también, que atiende a lo sumo una vez a cada cliente. Generalmente hay una cantidad exponencial de rutas factibles, que es el tamaño del conjunto Ω. Una solución en esta formulación, es un vector binario de variables de decisión, donde la variable λ_r indica si la ruta r es parte de la solución (1) o no (0). Este tipo de formulaciones tienen generalmente como ventaja una relajación lineal más ajustada, es decir, con menor gap de integralidad respecto a otras formulaciones para el mismo problema.

En estos casos, una estrategia para resolver la relajación lineal, es iniciar con un subconjunto pequeño de variables, que luego será expandido iterativamente según se necesite. La intuición de este método, se basa en que la mayoría de las variables participan en la solución óptima con un valor de 0, por lo que pueden no ser consideradas directamente.

Comenzamos entonces con un PPL llamado *problema maestro reducido* (PMR) con las mismas restricciones que el *problema primal o maestro* (PM) que queremos resolver, pero con un subconjunto pequeño de variables. Este subconjunto tiene que ser lo suficientemente grande como para que el PMR sea factible. En el caso del PM (2.9)-(2.11), su PMR es idéntico salvo por reemplazar el conjunto de todas las rutas Ω, por un subconjunto $\hat{\Omega} \subseteq \Omega$ del mismo, con $|\hat{\Omega}| \ll |\Omega|$. Resolvemos entonces el problema reducido, consiguiendo una solución (factible) óptima $\bar{\lambda} \in \mathbb{R}^{n'}$ con $n' = |\hat{\Omega}|$. Como el PMR tiene las mismas restricciones que el PM, $\bar{\lambda}$ puede extenderse para ser una solución factible del PM. Para esto, se completa con 0 el valor de todas las variables del PM no consideradas en el PMR. Sin embargo, al no estar considerando todas las variables, no tenemos garantías por el momento de que sea el óptimo del PM. Pasamos entonces a resolver un segundo problema, llamado *pricing*, que consiste en buscar una variable no considerada hasta el momento en el PMR, que al ser agregada, pueda mejorar el valor de la relajación lineal del PPLE, o de la función objetivo del PMR.

En el caso de una minimización, para mejorar la función objetivo, la nueva columna debe tener un costo reducido negativo. Definimos el costo reducido de la variable λ_r como:

$$c_r - \bar{y}^t \ A_r < 0 \tag{2.12}$$

donde A_r es la r-ésima columna de la matriz de coeficientes $A \in \mathbb{R}^{m \times n}$ de la formulación del PM que indica la participación de la variable λ_r, c_r es el coeficiente en la función objetivo de la r-ésima variable del PM y término independiente de la r-ésima restricción del problema dual, y $\bar{y}$ es el vector solución del problema dual asociado a $\bar{\lambda}$.

De no existir una variable con costo reducido negativo, no será posible mejorar el valor de la relajación lineal, y podemos concluir que la extensión de $\bar{\lambda}$ es un óptimo del PM. A continuación incluimos un esquema genérico conceptual de un algoritmo de generación de columnas. Las variables cuyos nombres aparecen en la explicación previa, toman el mismo rol que tomaron en la explicación. Por ejemplo, $\bar{x}$ e $\bar{y}$ son el valor del vector de variables primales en el PMR y en el problema dual asociado al PMR, respectivamente.

input : PPLE P
output: $\bar{x}$ óptimo de la relajación lineal de P

1 Inicializar PMR con las restricciones de P y un subconjunto de variables, $\bar{x} = \infty$
2 $\bar{x} =$ `ResolverRelajación`(PMR)
3 Sea $\bar{y}$ el vector solución del problema dual asociado a $\bar{x}$
4 **if** $\exists\ r \in \Omega / c_r - \bar{y}^t\ A_r < 0$ **then**
5 | Sea R un conjunto de variables con costo reducido negativo
6 | Agregar R a PMR
7 | Ir al paso 2
8 **end**

Algoritmo 2: Esquema conceptual de un algoritmo de generación de columnas

Cabe destacar que el problema dual del maestro tiene tantas restricciones como variables hay en el problema maestro, lo cual hace que buscar una variable con costo reducido

negativo sea también un problema generalmente difícil.

3. PROBLEMA

En este capítulo presentamos y definimos formalmente la variante del VRPDO que vamos a estudiar. Comenzamos detallando las asunciones que hacemos del problema. Luego introducimos la notación que utilizaremos a lo largo del trabajo. Concluimos el capítulo con una formulación de tipo set partitioning.

3.1 Nuestro enfoque

Como explicamos en la sección 1.2, la capacidad de los puntos de entrega compartidos es limitada y una de sus mayores desventajas, en especial en el caso de los lockers. Es por esto que modelar los retiros de paquetes podría tener un gran impacto en la eficiencia y uso de estos, y de aquí viene la importancia de estudiar alternativas que consideren que su capacidad de recepción puede incrementarse.

Asumimos entonces que los puntos SDL comienzan el horizonte de planificación con una capacidad inicial que puede incrementarse durante su ventana de tiempo debido al retiro de paquetes. Realizando consultas informales con empresas distribuidoras locales, nos informaron que el tiempo promedio de un paquete en un SDL es del orden de días, por lo que tiene sentido en la práctica asumir que solo los pedidos que se encuentran en el SDL al inicio del horizonte de planificación, son candidatos a ser retirados. Para reducir el alcance, consideramos también que no hay otros agentes fuera de la empresa distribuidora dejando paquetes en los puntos SDL, esto excluye tanto a otras empresas distribuidoras como clientes haciendo devoluciones. De esta forma, la capacidad en los puntos SDL puede interpretarse como una función partida, constante a trozos, cuyo valor es estrictamente creciente entre los intervalos en los que se define.

Dado que existen trabajos que estudian los aspectos de utilizar prioridades y niveles de servicio, preferimos concentrarnos en otro aspecto, como la capacidad, y no los consideraremos en este trabajo. Asumimos entonces que el cliente es quien define las opciones de entrega y todas tienen igual prioridad. También, no se puede enviar el paquete a otro punto que no sea uno de estos.

Al igual que en Tilk et al [1], consideramos que se incurre en un tiempo de servicio por cada paquete que es entregado, sea en un punto SDL o un punto AHD. También, asumimos que contamos con único depósito, en el cual se encuentra una flota homogénea ilimitada de vehículos. Estos vehículos parten del depósito, entregan uno o más pedidos, y vuelven al depósito.

Por último, consideramos que en la operatoria diaria de los choferes podría resultar difícil o anti-intuitivo realizar a lo largo de una misma ruta múltiples visitas a un mismo punto geográfico. Por esto es que asumimos que los SDL pueden ser visitados a lo sumo una vez durante una misma ruta.

3.2 Definición del problema

Tenemos un conjunto de puntos $L = \{0, 0'\} \cup L^{SDL} \cup L^{AHD}$, donde 0 y $0'$ son respectivamente el depósito inicial y final, y $L^{SDL} \cup L^{AHD}$ son los puntos en los cuales se pueden entregar pedidos. Si bien consideramos que tenemos un solo depósito del que parten todos

los vehículos, y al que regresan una vez que terminan de entregar sus pedidos, definimos artificialmente dos puntos para facilitar la notación después.

Como dijimos antes, hay dos tipos de puntos de entrega: específicos para un cliente (L^{AHD}) y compartidos (L^{SDL}). Para cada punto $l \in L$ hay un tiempo dedicado a la preparación de las entregas, como lo es el *tiempo para estacionar* el vehículo (p_l) y un tiempo relacionado a la entrega del pedido en sí, llamado *tiempo de servicio* (s_l). Esto se introduce para diferenciar el tiempo incurrido al entregar un pedido en un SDL, donde comúnmente hay estacionamiento reservado para los vehículos de carga y no hay que interactuar con el cliente. Por simplicidad definimos $s_0 = s_{0'} = p_0 = p_{0'} = 0$. En este trabajo consideramos que el tiempo de servicio incurrido por entregar un pedido en un punto SDL, no depende del pedido. Es por esto que los tiempos de servicio, que intuitivamente están asociados a los pedidos, podemos asociarlos en este caso a los puntos de entrega. Como dijimos previamente, consideramos al igual que en Tilk et al, que se incurre en un tiempo de servicio por cada pedido entregado en un punto SDL. También, cada punto $l \in L$ tiene un período de tiempo $[a_l, b_l]$ en el cual está disponible para recibir pedidos, que llamamos *ventana de tiempo*. Observar que el servicio de la entrega debe comenzar dentro de la ventana de tiempo del punto en donde se realice. Mientras los puntos L^{AHD} están limitados por la presencia del cliente para recibir el paquete, los puntos L^{SDL} lo están por el contexto en el cual operan. Por ejemplo, un locker ubicado dentro de un supermercado que cierra de noche. Entre dos puntos $i, j \in L$ hay un *tiempo de viaje* t_{ij} y un *costo de viaje* c_{ij}.

Los puntos de entrega tienen una capacidad de recepción en términos de la cantidad de pedidos que pueden ser recibidos en el lugar. Si bien el objetivo de esto es modelar el incremento de la capacidad en los puntos L^{SDL}, podemos extender este concepto sobre todos los puntos L. En un punto $l \in L^{AHD}$, como es utilizado para la tradicional entrega a domicilio, a lo sumo se puede entregar el pedido de ese cliente particular, por lo que podemos interpretar esto como tener una capacidad de recepción de 1 pedido que es constante durante todo el horizonte de planificación. En el caso del depósito, no se entregan pedidos, por lo que su capacidad de recepción se interpreta como nula y también constante. En base a la asunción sobre la estructura de la capacidad de los SDL en la sección 3.1, para cada $l \in L$ se define la función $C_l(t)$ que indica la capacidad de recepción del punto l en el tiempo $t \in [a_l, b_l]$. Como dijimos, esta función es constante a trozos. Llamamos $T(l) = \{[a_{l,1}, b_{l,1}], \ldots, [a_{l,h}, b_{l,h}]\}$ a la partición de la *ventana de tiempo* del punto l inducida por el dominio de los trozos de la función C_l. Cada intervalo $[a_{l,i}, b_{l,i}] \in T(l)$, llamado *intervalo de capacidad*, está delimitado por los extremos de la *ventana de tiempo* o momentos en los que ocurre el retiro de un paquete, y tiene asociado un valor no negativo de capacidad de recepción C_l^i.

A modo de ejemplo, tenemos un pequeño locker z que inicia el horizonte de planificación con 1 casillero libre y 2 ocupados. Este locker se encuentra dentro de una estación de tren que opera de 8 a 22 horas. Durante el día sabemos que vendrán 2 clientes a retirar sus pedidos, a las 10 y 14 horas, respectivamente. Entonces a las 10 horas, el locker pasa a tener 2 casilleros disponibles, lo que significa que puede recibir hasta 2 pedidos o que su capacidad de recepción es de 2. Se mantiene así hasta las 14 horas, en donde pasa a tener una capacidad de recepción de 3 pedidos hasta el final día. En la figura 3.1 se encuentra la función de capacidad de recepción de z, $C_z(t)$. En este ejemplo, z tiene definidos los intervalos de tiempo $T(z) = \{[8, 10], [10, 14], [14, 22]\}$, con las capacidades de recepción asociadas $C_z^0 = 1$, $C_z^1 = 2$ y $C_z^2 = 3$.

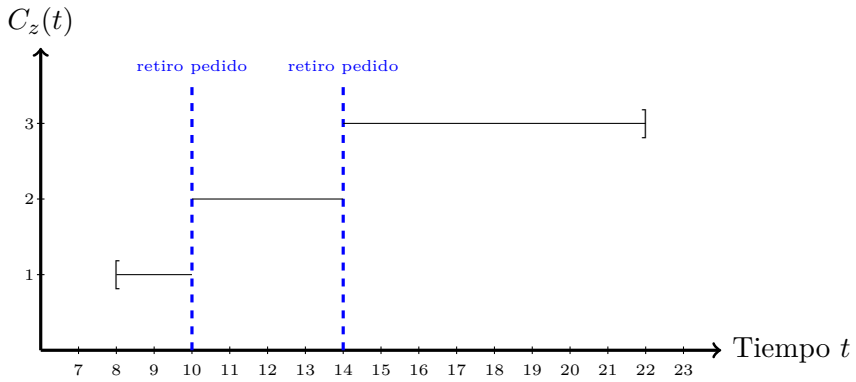

Fig. 3.1: Función de capacidad de recepción de un locker que inicia su ventana de tiempo $[8, 22]$ con una capacidad recepción de 1 pedido. En azul están marcados los momentos en los que se retiran paquetes, aumentando la capacidad de recepción.

Por otro lado, tenemos un conjunto de pedidos $N = \{1, \ldots, k\}$, donde cada pedido $n \in N$ tiene asociado una demanda q_n. Así mismo contamos con una flota homogénea ilimitada de vehículos disponibles con una capacidad Q. Estos valores se utilizan para modelar los recursos del vehículo que son consumidos al transportar un pedido. Por ejemplo, el volumen que un pedido ocupa dentro del acoplado de un camión, o el peso del pedido y el máximo peso que puede transportar el vehículo. Para ser consistentes luego, definimos también la demanda en el depósito como $q_0 = q_{0'} = 0$.

Consideramos que el cliente es quien decide los lugares donde puede recibir su pedido. Cada combinación (n, l), con $n \in N$ el pedido del cliente, y $l \in L$ un lugar elegido por el cliente, es lo que llamamos *opción de entrega*. Tenemos entonces un conjunto de opciones $W \subset N \times L$ definido por los usuarios. Por simplicidad, dada una opción $w = (n, l) \in W$, definimos $n_w = n$, $l_w = l$, $q_w = q_n$, $a_w = a_l$, $b_w = b_l$, $p_w = p_l$, y $s_w = s_l$. También, para cada pedido $n \in N$ definimos el conjunto de opciones asociadas al pedido $W(n) = \{w \in W | n_w = n\}$, y para cada punto $l \in L$ definimos el conjunto de opciones asociadas al punto $W(l) = \{w \in W | l_w = l\}$. Finalmente definimos $W^{AHD} = \{w \in W | l_w \in L^{AHD}\}$ y $W^{SDL} = \{w \in W | l_w \in L^{SDL}\}$.

A modo de ejemplo, en la figura 3.2, el conjunto de opciones asociadas al SDL_1 es $W(SDL_1) = \{(n_1, SDL_1), (n_2, SDL_1), (n_3, SDL_1)\}$. También, el conjunto de opciones asociado al pedido n_3 es $W(n_3) = \{(n_3, SDL_1), (n_3, SDL_2), (n_3, AHD_3)\}$.

Decimos que una ruta $r \in \Omega$ es una secuencia de opciones $(0, w_1, \ldots, w_n, 0')$ que comienza y termina en el depósito. Tiene asociada una secuencia de tiempos $(t_0, t_{w_1}, \ldots, t_{0'})$ que son los inicios de los tiempos de servicio de cada una de las opciones. Un vehículo puede llegar al punto l_{w_i} y postergar la entrega del pedido cuanto quiera, mientras el *tiempo de servicio* comience dentro de la *ventana de tiempo* del punto, es decir, $a_{w_i} \leq t_{w_i} \leq b_{w_i}$. Decimos que un pedido es atendido por una ruta cuando una de sus opciones está incluida en ella. También, el costo de una ruta r es la suma de los costos de cada tramo, que definimos como $c_r = \sum_{(i,j) \in r} c_{ij}$.

Consideramos que una ruta r es factible cuando:

1. tiene a lo sumo una opción de cada pedido, es decir, sean $w_i, w_j \in r$, $w_i, w_j \in W \wedge n_{w_i} = n_{w_j} \implies i = j$.

2. su secuencia de inicios de servicios respeta el tiempo necesario para trasladarse y preparar el siguiente pedido, es decir, dadas $(w_i, w_{i+1}) \in r$ un par de opciones visitadas continuas en la ruta, $t_i + s_{w_i} + t_{i\,i+1} + p_{w_{i+1}} \leq t_{i+1}$.

3. todos los servicios comienzan durante de la ventana de tiempo del punto en cuestión, es decir, $a_w \leq t_w \leq b_w$ para $w \in r$.

4. la suma de las demandas de las opciones es menor o igual que la capacidad del vehículo, es decir, $\sum_{w \in r} q_w \leq Q$.

5. la cantidad de pedidos entregados en un punto SDL es en cualquier momento menor o igual a la capacidad de recepción del intervalo correspondiente. Formalmente podemos escribir esto para cada opción $w_i \in r$ como $|\{w_j | w_j \in r, j \leq i, l_{w_j} = l_{w_i}\}| \leq C_{l_{w_i}}(t_i)$.

6. no visita al mismo punto SDL más de una vez, es decir, si dos opciones de entrega en la ruta están asociadas al mismo punto, todas las opciones atendidas por la ruta entre estas dos están también en ese mismo. Sean $w_i, w_j \in r$, esto lo escribimos formalmente como $l_{w_i} = l_{w_j} \implies (\forall i \leq i' \leq j)\ l_{w_i} = l_{w_{i'}}$.

y decimos que Ω es el conjunto de todas las rutas factibles.

El problema de ruteo de vehículos con ventanas de tiempo, entregas alternativas y capacidades dinámicas (VRPDOTW-DC por sus siglas en inglés) consiste entonces en encontrar un conjunto de rutas factibles, que atienda exactamente una vez a todos los pedidos, sin exceder en conjunto las capacidades de recepción de los SDL en ninguno de sus intervalos, y cuya suma de costos de rutas sea mínima.

3.2.1 ¿Cómo afectan las capacidades dinámicas a las rutas?

Analizaremos la factibilidad de las rutas en la figura 3.2 respecto a las capacidades de cada punto SDL.

Lo primero que se puede validar, es que una ruta que deja más pedidos en un intervalo que lo permitido por su capacidad de recepción, es siempre infactible. Por lo que la ruta azul es factible solo si existe un intervalo de capacidad en SDL_2 de al menos 1 pedido, es decir, si hay un $t \in [a_{SDL_2}, b_{SDL_2}]$ con $0 < C_{SDL_2}(t)$. Con el mismo argumento, la ruta negra y la verde serán infactibles si todos los intervalos del SDL_1 tiene una capacidad de recepción menor o igual a 1 pedido, es decir, si para todo $t \in [a_{SDL_1}, b_{SDL_1}]$ vale que $C_{SDL_1}(t) \leq 1$.

Si dentro del horizonte de planificación $[0, T]$, decimos que el punto SDL_1 tiene una capacidad de recepción de 1 pedido durante el período $[0, 12]$ y una capacidad de 2 en el período $[12, T]$, significa que: SDL_1 comienza su *ventana de tiempo* pudiendo recibir 1 pedido, un pedido preexistente es retirado antes de las 12, y a partir de las 12, el punto SDL_1 tiene una capacidad de recepción de 2 pedidos. Esto es lo mismo que decir que tiene definidos los intervalos $T(SDL_1) = \{[0, 12], [12, T]\}$ con las capacidades $C^0_{SDL_1} = 1$ y $C^1_{SDL_1} = 2$.

En este escenario se puede por ejemplo entregar solo 1 pedido en cualquiera de los dos intervalos, por lo que la ruta marrón es factible siempre y cuando inicie los servicios en las ventanas de tiempo de cada punto. También se puede entregar 2 pedidos, siendo ambos durante $[12, T]$ o uno en cada intervalo. Por lo que la ruta negra será factible siempre

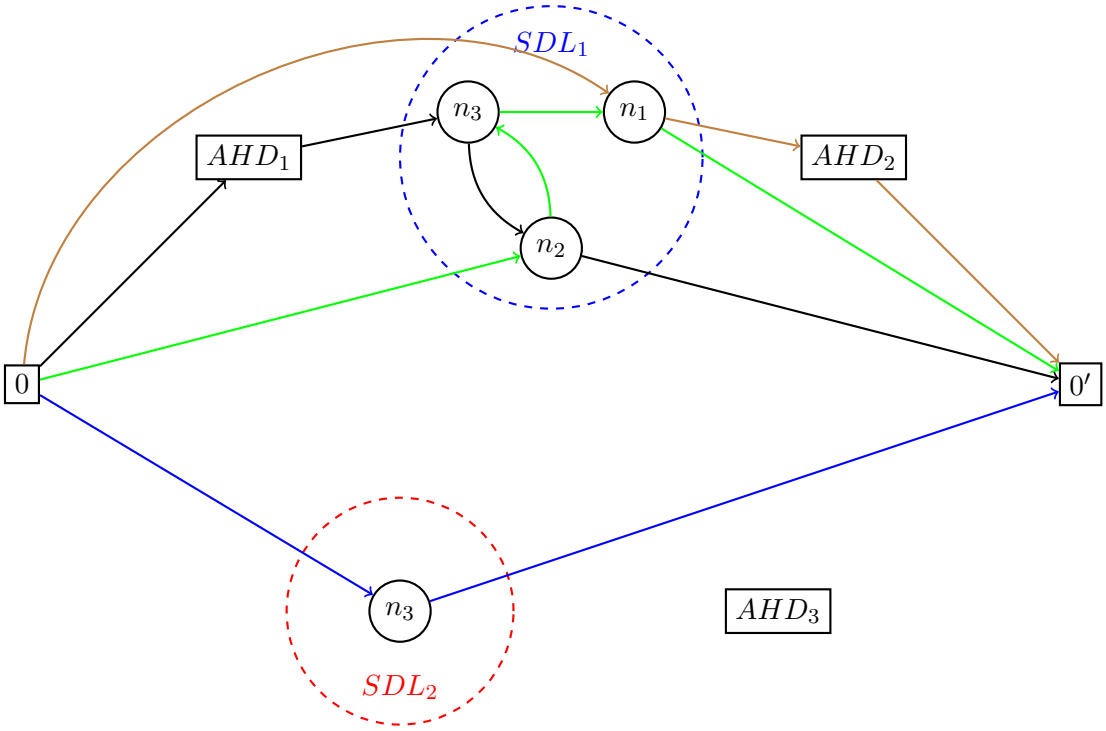

Fig. 3.2: Red con 3 pedidos y 2 puntos SDL. Los 3 pedidos tienen asociado un domicilio y al SDL_1 como opciones de entrega. El pedido n_3 también agrega al SDL_2.

y cuando atienda al pedido n_2 en el segundo intervalo, e inicie todos los servicios en las respectivas ventanas de tiempo.

Como no se puede entregar 3 pedidos en ninguno de los intervalos de capacidad del SDL_1, ni tampoco entregar 1 pedido durante $[0, 12]$ y 2 durante $[12, 16]$, la ruta verde es siempre infactible. Notar que el último caso no es factible porque el pedido entregado durante $[0, 12]$ permanece en el punto durante el resto de la planificación, ocupando 1 unidad de recepción de las 2 disponibles durante $[12, T]$.

Cabe destacar que si bien las rutas marrón y negra pueden ser factibles individualmente, no podrán ser elegidas ambas al mismo tiempo en la solución del problema, dado que nunca hay una capacidad de recepción de al menos 3 pedidos durante la ventana de tiempo del SDL_1. Esto implica la necesidad de una restricción de sincronización entre todas las rutas que usen el mismo SDL.

También, es importante notar que el espacio de rutas factibles decrece a medida que resumimos o juntamos los intervalos de capacidad de un SDL. Por ejemplo, si la capacidad del locker z de la figura 3.1 fuera a ser reportada de manera diaria, es decir, con un solo valor de capacidad de recepción durante toda la planificación, no se puede decir que esta es de 2 o 3 pedidos, ya que cualquier ruta que deja 3 o más pedidos luego de las 14 horas, o 2 o más pedidos antes de las 10 horas, es infactible. Por otro lado, en todo momento de la planificación hay capacidad para recibir al menos 1 pedido, por lo que este puede ser indicado como la capacidad durante este período. De forma más general, si queremos reportar la capacidad de recepción de un punto $l \in L^{SDL}$ durante todo el período $[t_1, t_2]$ esta será la mínima capacidad de recepción disponible durante ese intervalo, es decir, $\min_{t \in [t_1, t_2]} C_l(t)$.

Si en lugar de medir capacidades diariamente, lo hacemos cada 7 horas, obtenemos los intervalos $T(z) = \{[8, 15], [15, 22]\}$, cada uno con una capacidad de recepción de 1 y 3 pedidos. En la figura 3.3 podemos ver como esta forma de agregar el input aproxima la función de capacidad. La capacidad diaria está marcada en rojo, mientras que la capacidad cada 7 horas está marcada en verde. Ahora el espacio de rutas factibles puede contener rutas que dejan 2 o 3 pedidos, potencialmente tantas como en el ejemplo original. De esta manera vemos que mientras menos resumida sea la información, o dicho de otra forma, más chicos sean los intervalos con los que se reporta la capacidad (horas en lugar de días o minutos en lugar de horas), hay más rutas factibles, potencialmente reduciendo el costo logístico.

Como sucede en el escenario anterior con el pedido n_2 en la ruta negra, el intervalo de capacidad en el que un vehículo entrega un pedido en un punto SDL es importante para determinar si la ruta es factible. Este tiempo es afectado, por ejemplo, por el horario de partida desde el depósito inicial, desde AHD_1, y desde n_3 en el SDL_1. Consideramos que un vehículo inicia su viaje a una ubicación tan pronto como termina el servicio en el punto de origen, y puede esperar en el punto destino tanto como desee previo a realizar el servicio en este. En la práctica, esperar tiene sentido solamente cuando el punto destino tiene más de un intervalo de capacidad o cuando el vehículo llega previo al inicio de su ventana de tiempo. De esta forma, en lugar de considerar múltiples horarios de partida desde el origen de un tramo cualquiera, consideramos múltiples horarios de inicio de servicio en el destino, potencialmente uno por cada intervalo de capacidad. Por ejemplo, supongamos que un vehículo transitando la ruta negra, que se encuentra en el punto AHD_1 en el tiempo 2. Este puede optar por llegar al SDL_1 e inmediatamente atender al pedido n_3 en el intervalo $[0, 12]$, o llegar y esperar hasta que comience el intervalo $[12, T]$ para comenzar el servicio.

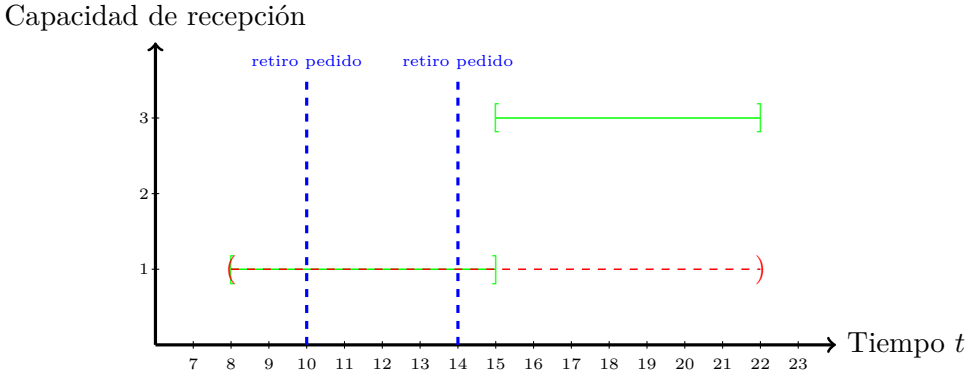

Fig. 3.3: Mismo locker que de la figura 3.1, ahora reportando la capacidad de manera diaria en rojo, y de a 7 horas en verde.

De manera implícita, lo que estamos haciendo con la red para cada $l \in L^{SDL}$, es replicar las opciones $w \in W(l)$ por cada intervalo en $T(l)$.

Una vez que una ruta llega al depósito final, se puede actualizar el tiempo de partida desde el depósito inicial para acortar el tiempo de total de viaje. Si el tiempo de espera no es considerado en el costo de la ruta, como en nuestro caso, este último paso puede ser descartado. En ambos casos, no necesitamos modelar los posibles tiempos de partida desde el depósito inicial hasta que la secuencia de opciones de una ruta se completa.

3.3 Formulación Extendida

Para la siguiente formulación, dada $r \in \Omega$, definimos las variables binarias λ_r cuyo valor en una solución factible es 1 cuando la ruta r es seleccionada en la solución y 0 en caso contrario. Dadas un opción de entrega $w \in W$ y una ruta $r \in \Omega$, el coeficiente binario Entrega_{wr} es 1 cuando la opción w está en la secuencia de la ruta r y 0 en caso contrario. Para calcular de forma sencilla el uso acumulado de la capacidad de recepción de una ruta en cada intervalo de capacidad, definimos de manera similar para cada opción de entrega $w \in W$, ruta $r \in \Omega$ e intervalo de capacidad $i \in T(l_w)$ el coeficiente binario Entrega^i_{wr}, que vale 1 cuando la opción w está en la secuencia de la ruta r y su tiempo de servicio ocurre durante el intervalo i o antes, es decir, $t_w \leq b_{l_w,i}$, y 0 en caso contrario.

A continuación mostramos la formulación por set partitioning para el VRPDOTW-DC:

$$\min \sum_{r\in\Omega} c_r\lambda_r \tag{3.1}$$

$$\text{s.t.} \sum_{r\in\Omega} \sum_{w\in W(n)} \text{Entrega}_{w,r}\lambda_r = 1 \qquad n \in N \tag{3.2}$$

$$\sum_{r\in\Omega} \sum_{w\in W(l)} \text{Entrega}^i_{w,r}\lambda_r \leq C^i_l \qquad l \in L^{\text{SDL}},\ i \in T(l) \tag{3.3}$$

$$\lambda_r \in \{0,1\} \qquad r \in \Omega \tag{3.4}$$

La función objetivo (3.1) busca reducir la suma de los costos de las rutas seleccionadas. Las restricciones (3.2) aseguran que cada cliente sea atendido exactamente una vez entre todas las rutas seleccionadas. Para cada SDL, la restricción (3.3) asegura que el número

de pedidos entregados entre todas las rutas hasta cualquier intervalo nunca exceda su capacidad de recepción.

En la práctica, se suele relajar la restricción (3.2) pidiendo que se atienda cada pedido al menos una vez ($\geq$), porque obtiene mejores resultados en cuanto a tiempos de cómputo. Esto se puede hacer cuando los tiempos y costos de viaje satisfacen la desigualdad triangular, ya que una solución óptima considerando Ω, atiende cada pedido una sola vez. Por ejemplo, si dos rutas $(\dots, w_{i_k}, w_{i_{\overline{k}}}, w_{i_k+1}, \dots)$ y $(\dots, w_{i_j}, w_{i_{\overline{k}}}, w_{i_j+1}, \dots)$ son seleccionadas en una solución óptima, podemos reemplazar los tramos $(w_{i_k}, w_{i_{\overline{k}}})$ y $(w_{i_{\overline{k}}}, w_{i_k+1})$ con (w_{i_k}, w_{i_k+1}), y la ruta sigue siendo factible respecto a tiempos de servicio porque los tiempos de viaje satisfacen la desigualdad triangular. Como también vale para los costos de viaje, la nueva solución tiene un costo menor o igual.

4. ALGORITMO BRANCH AND PRICE

En esta sección presentamos los componentes principales de nuestro algoritmo exacto para abordar el VRPDOTW-DC. El mismo se basa en un esquema de tipo branch and price, donde primero se resuelve la relajación lineal de la formulación (3.1)-(3.4). Como parte de la resolución de la relajación lineal, el problema de pricing es abordado mediante algoritmos de labeling basados en programación dinámica. Si la solución de esta no satisface las condiciones de integralidad, se aplica un algoritmo de tipo branch and bound, revisando el esquema de branching propuesto en busca de una regla por la cual subdividir el problema en dos más pequeños. Mostramos que si no encontramos ninguna desigualdad con nuestro esquema, la solución asociada debe ser entera.

4.1 Relajación Lineal

El modelo (3.1)-(3.4) tiene generalmente una cantidad exponencial de variables, por lo que para resolver su relajación lineal utilizamos la técnica generación de columnas introducida en la sección 2.3. De manera similar al ejemplo de esa sección, nuestro PMR resulta ser la formulación(3.1)-(3.4) restringiendo el conjunto de todas las rutas factibles Ω a un subconjunto inicial $\hat{\Omega} \subseteq \Omega$ con $|\hat{\Omega}| \ll |\Omega|$.

En nuestro caso, como nuestras variables primales son decisiones sobre elección de rutas, el problema de pricing consta de encontrar una ruta no considerada en el PMR cuyo costo reducido sea negativo. Este último, como vimos en la sección 2.1, se escribe a partir de una combinación de las restricciones del problema primal, por lo que precisamos introducir las variables duales de la formulación (3.1)-(3.4). Para todo pedido $n \in N$, Π_n es la variable dual asociada a la restricción (3.2) de atención de pedidos. Para todo punto $l \in L^{SDL}$, y todo $i \in T(l)$, $\Phi_{l,i}$ es la variable dual relacionada a la restricción (3.3) de capacidad de l durante su intervalo i. Los coeficientes que explican la participación de cada una de estas variables duales en el costo reducido son los mismos que aparecen en la formulación (3.1)-(3.4), $\text{Entrega}_{w,r}$ y $\text{Entrega}^i_{w,r}$. Así, el costo reducido de una ruta r se puede escribir como:

$$\bar{c}_r = c_r - \sum_{n \in N} \sum_{w \in W(n)} \Pi_n \, \text{Entrega}_{w,r} + \sum_{l \in L^{SDL}} \sum_{w \in W(l)} \sum_{i \in T(l)} \Phi_{l,i} \, \text{Entrega}^i_{w,r} < 0 \qquad (4.1)$$

Dado que toda opción $w \in W$ tiene un pedido asociado, y que podemos extender el concepto de intervalo a puntos $l \in l^{AHD}$, este puede reescribirse como:

$$\bar{c}_r = c_r - \sum_{w \in W} \Pi_{n_w} \, \text{Entrega}_{w,r} + \sum_{w \in W} \sum_{i \in T(l_w)} \Phi_{l_w,i} \text{Entrega}^i_{w,r} < 0 \qquad (4.2)$$

El problema de pricing consiste entonces en encontrar una ruta $r \in \Omega \setminus \hat{\Omega}$ con costo reducido $\bar{c}_r < 0$. En caso de que dicha ruta exista, se añade al subconjunto $\hat{\Omega}$ pasando a ser considerada por el PMR y repetimos el proceso. Caso contrario, hemos probado que estamos frente a un óptimo de la relajación lineal.

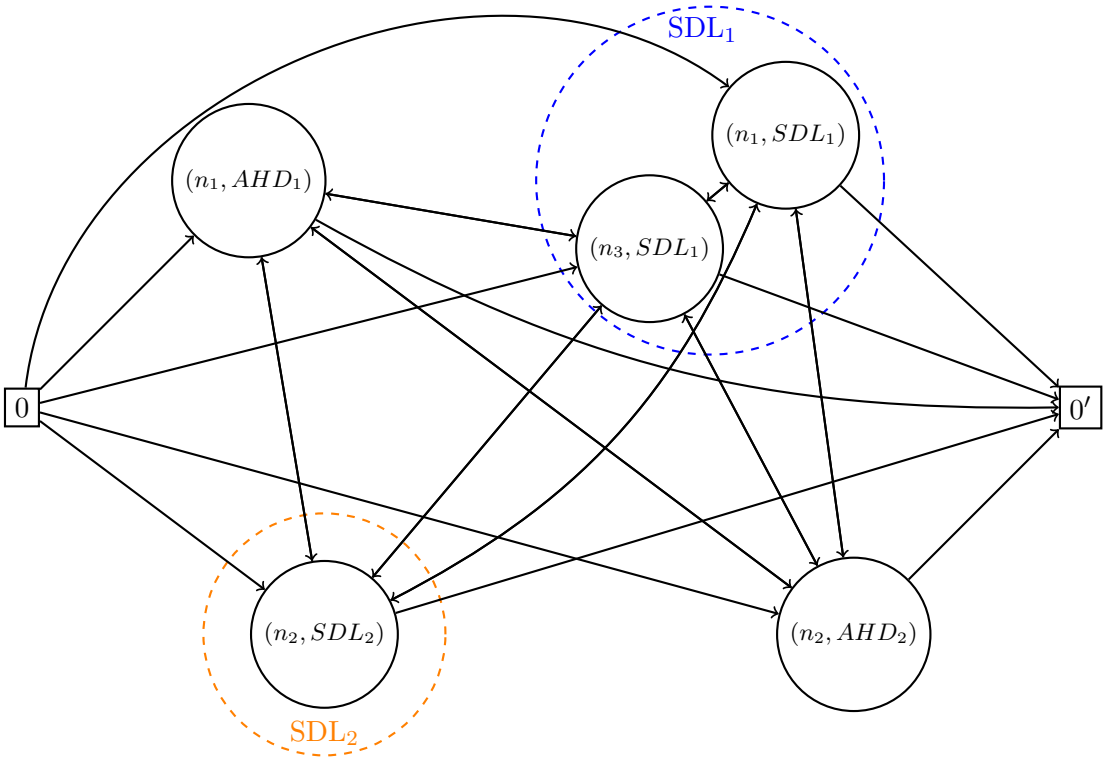

Fig. 4.1: Red logística con 3 pedidos, 2 puntos SDL, 2 puntos AHD y los depósitos. $W = \{(n_1, AHD_1), (n_1, SDL_1), (n_3, AHD_2), (n_2, SDL_1), (n_2, SDL_2)\}$.

4.1.1 Red subyacente

En general, la estructura de los problemas de pricing de la formulación set partitioning del VRP son una variación de un problema de caminos mínimos sobre grafos. Para esto representamos la red logística a través de un grafo dirigido $G = (V, A)$, al que luego le haremos modificaciones motivadas por características específicas del problema. El conjunto de vértices V esta formado por las opciones $w \in W$ y los depósitos $\{0, 0'\}$. También, hay un arco $(v_1, v_2) \in A$ por cada par de vértices $v_1, v_2 \in V$ con $l_{v_1} \neq l_{v_2}$, y si $v_1, v_2 \in W$, con $n_{v_1} \neq n_{v_2}$. Como nos interesan las rutas que entregan al menos 1 pedido, el arco $(0, 0') \notin A$. Dado que los tiempos de servicio de los pedidos en un mismo punto $l \in L^{SDL}$ son iguales, el orden en que se entreguen dentro de l es indistinto, por lo que podemos modelar las entregas en l utilizando el orden enumerado de los pedidos para quitar la simetría entre las opciones de $w \in W(l)$. Definimos entonces de manera compacta al conjunto de arcos como $A = \{(v_1, v_2) \in V \times V | l_{v_1} \neq l_{v_2} \wedge (v_1, v_2 \in W \implies n_{v_1} \neq n_{v_2}) \wedge (v_1, v_2) \neq (0, 0')\} \cup \{(v_1, v_2) \in W^{SDL} \times W^{SDL} | l_{v_1} = l_{v_2} \implies n_{v_1} < n_{v_2}\}$.

A modo de ejemplo, partiendo de la red logística de VRPDOTW-DC en la figura 4.1, generamos su grafo correspondiente como dijimos previamente y lo mostramos en la figura 4.2 . Mientras que la primera es una ilustración que luce como un grafo para facilitar la comprensión, la segunda es el grafo propiamente dicho. La diferencia entre estas es que la segunda no tiene simetrías dentro de los puntos $l \in L^{SDL}$. En este caso esto se traduce en que el arco $((n_3, SDL_1), (n_1, SDL_1)) \notin A$.

Para reducir el número de arcos de G, agrupamos las opciones $w \in W$ asociadas a un mismo punto $l \in L^{SDL}$ agregando un par de vértices artificiales para cada uno de estos grupos, que denotamos $SDL_{in}(l)$ y $SDL_{out}(l)$, y representen la entrada y salida del punto l respectivamente. Ahora todas las entregas de pedidos en un punto $l \in L^{SDL}$ están en la secuencia de la ruta precedidas por un vértice de entrada $SDL_{in}(l)$ y continuadas por un vértice de salida $SDL_{out}(l)$, sin vértices de otros puntos geográficos en el medio. Para cada $l \in L^{SDL}$, se define $l_{SDL_{in}(l)} = l_{SDL_{out}(l)} = l$, $a_{SDL_{in}(l)} = a_{SDL_{out}(l)} = a_l$ y $b_{SDL_{in}(l)} = b_{SDL_{out}(l)} = b_l$. Definimos un nuevo conjunto de vértices $\hat{V} = V \cup SDL_{in} \cup SDL_{out}$, donde SDL_{in} y SDL_{out} son los conjuntos de puntos artificiales de entrada y salida respectivamente. En cuanto a los arcos, se mantienen todos los arcos $(v_1, v_2) \in A$ entre opciones asociadas a un mismo punto $(v_1, v_2 \in W^{SDL})$ y los que no inciden en vértices que son opciones en puntos de entrega compartida $(v_1, v_2 \notin W^{SDL})$. De manera compacta, se mantienen los arcos $A_{\text{persistidos}} = \{(v_1, v_2) \in A | (v_1, v_2 \notin W^{SDL}) \vee (v_1, v_2 \in W^{SDL} \wedge l_{v_1} = l_{v_2})\}$. Agregamos arcos desde la entrada de cada $l \in L^{SDL}$ a sus opciones, llamamos a este conjunto $A_{in} = \{(SDL_{in}(l), w) | l \in L^{SDL}, w \in W(l)\}$, y de la misma manera, desde sus opciones a la salida $A_{out} = \{(w, SDL_{out}(l)) | l \in L^{SDL}, w \in W(l)\}$. Notar que de la misma forma que no nos interesan las rutas que no atienden clientes, no permitimos que se visite un punto $l \in L^{SDL}$ sin que se atienda al menos un pedido. Por último, cada arco $(v_1, v_2) \in A$ entre vértices de distintos puntos $(l_{v_1} \neq l_{v_2})$ y donde al menos uno de los dos es una opción en un punto de entrega compartido $(v_1 \in W^{SDL} \vee v_2 \in W^{SDL})$, es descartado y en su lugar se agrega un arco donde la cola es $SDL_{out}(l_{v_1})$ si $v_1 \in W^{SDL}$ o v_1 de lo contrario, y la cabeza es $SDL_{in}(l_{v_2})$ si $v_2 \in W^{SDL}$ y v_2 de lo contrario. Esto lo podemos escribir como $A_{\text{colapsados}} = \{(f(v_1), g(v_2)) | (v_1, v_2) \in A, l_{v_1} \neq l_{v_2} \wedge (v_1 \in W^{SDL} \vee v_2 \in W^{SDL})\}$ donde

$$f(v) = \begin{cases} SDL_{out}(l_v) & v \in W^{SDL} \\ v & \text{de lo contrario} \end{cases}$$

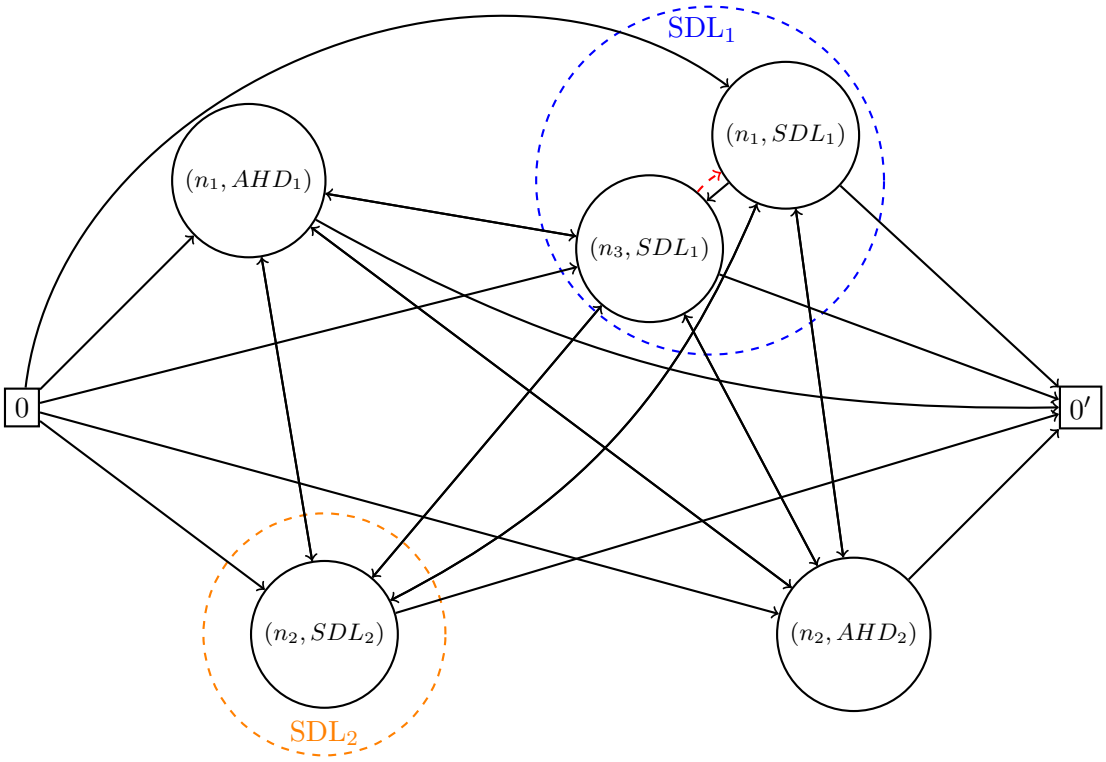

Fig. 4.2: grafo dirigido G generado a partir de la instancia de la figura 4.1, utilizando el orden $n_1 < n_2 < n_3$. En rojo el único arco eliminado en el proceso.

$$g(v) = \begin{cases} SDL_{in}(l_v) & v \in W^{SDL} \\ v & \text{de lo contrario} \end{cases}$$

De esta manera, el nuevo conjuntos de arcos queda definido como $\hat{A} = A_{\text{persistidos}} \cup A_{out} \cup A_{in} \cup A_{\text{colapsados}}$, y definimos el grafo reducido de un grafo de VRPDOTW-DC G, como $G_{\text{reducido}} = (\hat{V}, \hat{A})$. Observar que potencialmente todos los vértices $v \in V$ que representan opciones en puntos L^{AHD} tienen arcos salientes a los vértices $\bar{v} \in V$ que representan opciones en puntos L^{SDL}. También, los vértices $v \in V$ que representan opciones en un mismo punto $l \in L^{SDL}$ pueden tener arcos a todos los vértices que representan opciones en otros puntos $L^{SDL} - \{l\}$. El haber introducido dos vértices artificiales por cada SDL nos permitió remover todos estos arcos.

Continuando con el ejemplo de la figura 4.2, aplicando estos cambios sobre el grafo dirigido G resulta en el grafo reducido que se encuentra en la figura 4.3. Tenemos en este ejemplo todos los casos de reducción de arcos. Se puede ver por ejemplo que los nuevos vértices SDL_{in} y SDL_{out} colapsan los arcos entre vértices que representan opciones de entrega en SDL_1 y en SDL_2.

Hasta el momento un vértice codifica en sí el punto en donde se encuentra, y si es una opción, el pedido que abastece. Para facilitar la explicación y notación de los algoritmos que siguen, es útil que el intervalo en que se está realizando la entrega también esté codificado en el vértice. Para esto definimos un nuevo conjunto llamado las opciones extendidas, que consiste en $W_{ext} = \{(w, i) | w \in W, i \in T(l_w)\}$. Y por simplicidad, definimos las mismas operaciones que utilizamos sobre W y sus elementos, dada una opción extendida $w = (\hat{w}, i) \in W_{ext}$, definimos $n_w = n_{\hat{w}}$, $l_w = l_{\hat{w}}$, $q_w = q_{\hat{w}}$, $a_w = a_i$, $b_w = b_i$, $p_w = p_{\hat{w}}$, $s_w = s_{\hat{w}}$ y $i_w = i$. Utilizar los límites del intervalo de capacidad de recepción como ventana de tiempo mantiene a las opciones $w \in W^{AHD}$ de igual manera ya que $|T(l_w)| = 1$ y sea $i \in T(l_w)$, $a_w = a_i \wedge b_w = b_i$. Dada un punto $l \in L^{AHD} \cup L^{SDL}$, definimos $W_{ext}(l) = \{w \in W_{ext} | l_w = l\}$. Dada un pedido $n \in N$, definimos $W_{ext}(n) = \{w \in W_{ext} | n_w = n\}$. También, denotamos $W_{ext}^{SDL} = \{w \in W | l_w \in L^{SDL}\}$ y $W_{ext}^{AHD} = \{w \in W | l_w \in L^{AHD}\}$. Con este nuevo conjunto de opciones, definimos un nuevo conjunto de vértices $V_{ext} = \hat{V} - W \cup W_{ext}$, donde reemplazamos las opciones con opciones extendidas. En cuanto al conjunto de arcos, se mantienen los arcos $(v_1, v_2) \in \hat{A}$ entre vértices que no son opciones de entrega en puntos L^{SDL}, cambiando las cabezas o colas de arcos que son opciones en puntos L^{AHD}, por su respectiva y única opción extendida. Esto podemos escribirlo más formal como $\hat{A}_{\text{persistido}} = \{(h(v_1), h(v_2)) | (v_1, v_2) \in \hat{A}, (v_1 \in W \rightarrow l_{v_1} \in L^{AHD}) \wedge (v_2 \in W \rightarrow l_{v_2} \in L^{AHD})\}$. Donde h está definida sobre $\hat{V} - W^{SDL}$ de la siguiente manera:

$$h(v) = \begin{cases} w & v \in W^{AHD}, w \in W_{ext}(l_v) \\ v & sino \end{cases}$$

En el caso de los vértices relacionados a un punto $l \in L^{SDL}$, lo que estamos haciendo es replicar las opciones $W(l)$ por cada uno de los intervalos $i \in T(l)$. Debe haber un arco desde la entrada del punto $l \in L^{SDL}$ a los nuevos vértices, y también estos deben tener un arco a la salida de l. Esto podemos escribirlo formal y compacto como lo hicimos previamente, siendo $\hat{A}_{in} = \{(SDL_{in}(l), w) | l \in L^{SDL}, w \in W_{ext}(l)\}$ y $\hat{A}_{out} = \{(w, SDL_{out}(l)) | l \in L^{SDL}, w \in W_{ext}(l)\}$. Manteniendo un orden total entre los pedido, habrá un arco entre dos opciones cuando asumiendo un tiempo de viaje nulo, desde el intervalo del vértice origen se puede llegar antes de que termine el intervalo del

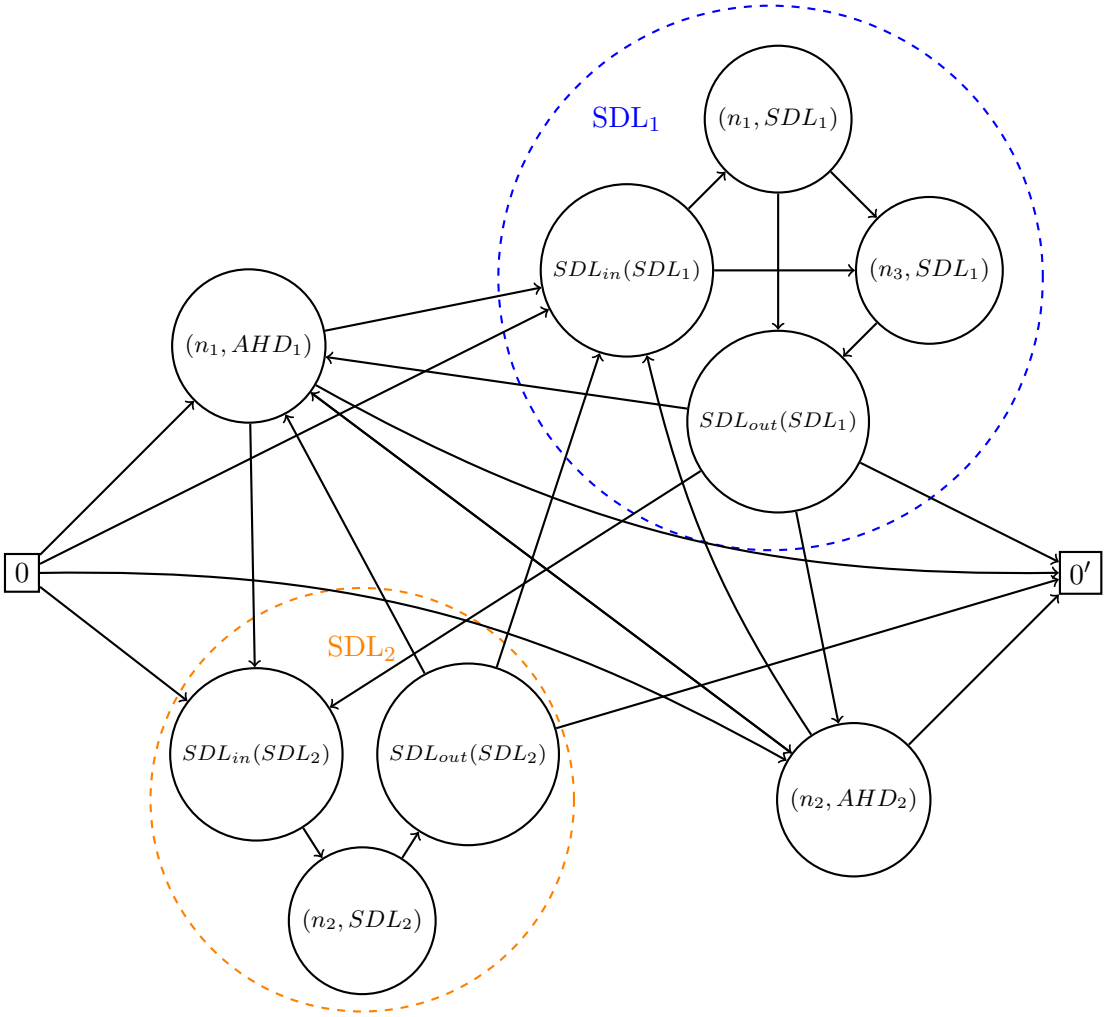

Fig. 4.3: El resultado de reducir los arcos del grafo de la figura 4.2. Para cada $l \in L^{SD}$, sus vértices de entrada y salida están etiquetados como in_l y out_l respectivamente.

vértice destino, es decir, dado un punto $l \in L^{SDL}$ y dos opciones $w_1, w_2 \in W(l)$ asociadas al punto, hay un arco (w_1, w_2) si $n_{w_1} < n_{w_2}$ y $a_{i_{w_1}} \leq b_{i_{w_2}}$. Esto lo escribimos como $\hat{A}_{SDL} = \{(w_1, w_2) \in W_{ext}^{SDL} \times W_{ext}^{SDL} | l_{w_1} = l_{w_2} \wedge n_{w_1} < n_{w_2} \wedge a_{i_{w_1}} \leq b_{i_{w_2}}\}$. Entonces definimos el nuevo conjunto de arcos $A_{ext} = \hat{A}_{\text{persistido}} \cup \hat{A}_{in} \cup \hat{A}_{out} \cup \hat{A}_{SDL}$, y con este, el grafo extendido $G_{ext} = (V_{ext}, A_{ext})$, resultado de aplicar distintas transformaciones desde G. Con este trabajaremos a partir de ahora. Por simplicidad, definimos el vecindario de salida para un vértice $v \in V_{ext}$ como $\delta^-(v) = \{\hat{v} \in V_{ext} | (v, \hat{v}) \in A_{ext}\}$.

A modo de ejemplo, partiendo de la figura 4.3 vamos a generar un grafo extendido. Supongamos que SDL_1 tiene los intervalos $T(SDL_1) = \{[0, 12], [12, T]\}$, con capacidades de recepción $C^0_{SDL_1} = 1$ y $C^1_{SDL_1} = 2$. Y que SDL_2 tiene el intervalo $T(SDL_2) = \{[0, 14]\}$, con una capacidad de recepción de $C^0_{SDL_2} = 1$. Con esta información podemos construir el grafo extendido, en la figura 4.3

Vamos a asignar costos a los arcos de G_{ext} de forma que el costo de un camino que parte del depósito inicial (0) y llega al depósito final $(0')$ sea el costo reducido de la ruta que este camino representa. Para esto utilizamos la siguiente función partida definida sobre pares de vértices de V_{ext}:

$$\overline{c}_{v_1,v_2} = \begin{cases} c_{l_{v_1},l_{v_2}} & v_2 \in \{0, 0'\} \cup SDL_{in} \cup SDL_{out} \\ c_{l_{v_1},l_{v_2}} - \pi_{n_{v_2}} & l_{v_2} \in L^{AHD} \\ c_{l_{v_1},l_{v_2}} - \pi_{n_{v_2}} + \sum_{\hat{i} \in T(l_{v_2}), a_{v_2} \leq b_{\hat{i}}} \Phi_{l_{v_2},\hat{i}} & \text{en otro caso} \end{cases}$$

donde $c_{l_1 l_2}$ es la función de costos de viaje entre dos puntos de la red, input del problema. Dada una ruta $r \in \Omega$, un punto de entrega compartido $l \in L^{SDL}$ y un intervalo del punto $i \in T(l)$, si r utiliza una unidad de capacidad de recepción por entregar un pedido en el intervalo i, impacta tanto en el intervalo i como en los siguientes del mismo punto, es decir todos los intervalos $\hat{i} \in T(l)$ tal que $a_i \leq b_{\hat{i}}$, decrementando la capacidad en una unidad en cada uno de estos. Por esto mismo es que en el costo de un arco encontramos las duales asociadas a las restricciones (3.3) de intervalos posteriores.

El problema de pricing se transforma ahora en buscar un camino simple entre 0 y $0'$ en G_{ext} de costo mínimo, que tiene que cumplir ciertas restricciones sobre recursos en cada nodo, como que la carga acumulada del camino no exceda la capacidad del camión. Este problema fue introducido en [4] y se lo conoce como el Problema de Camino Mínimo Elemental con Restricciones de Recursos. Pertenece a la clase de problemas NP-hard y cuando el conjunto de recursos es lo suficientemente restrictivo, es común abordarlo con algoritmos de labeling como el que detallamos a continuación.

4.1.2 Algoritmos de Labeling

Estos algoritmos se basan en la generación de *labels* que representan un camino parcial desde una fuente, en nuestro caso el depósito inicial (0), y que son extendidos hasta llegar al sumidero, en nuestro caso el depósito final $(0')$. Cada label factible está relacionado a un vértice final $v \in V$ y puede ser extendido a través de los arcos salientes de v a sus vecinos $\hat{v} \in V$, formando nuevos labels que están relacionados a estos y representan los caminos parciales $0 \to \cdots \to v \to \hat{v}$. Dado que la cantidad de caminos factibles es exponencial, se utilizan distintas técnicas para recortar el espacio de búsqueda garantizando mantener el óptimo.

Estos algoritmos los analizamos en varias partes. Primero damos la estructura del label para nuestro problema puntual, con todas las condiciones que debe cumplir para ser

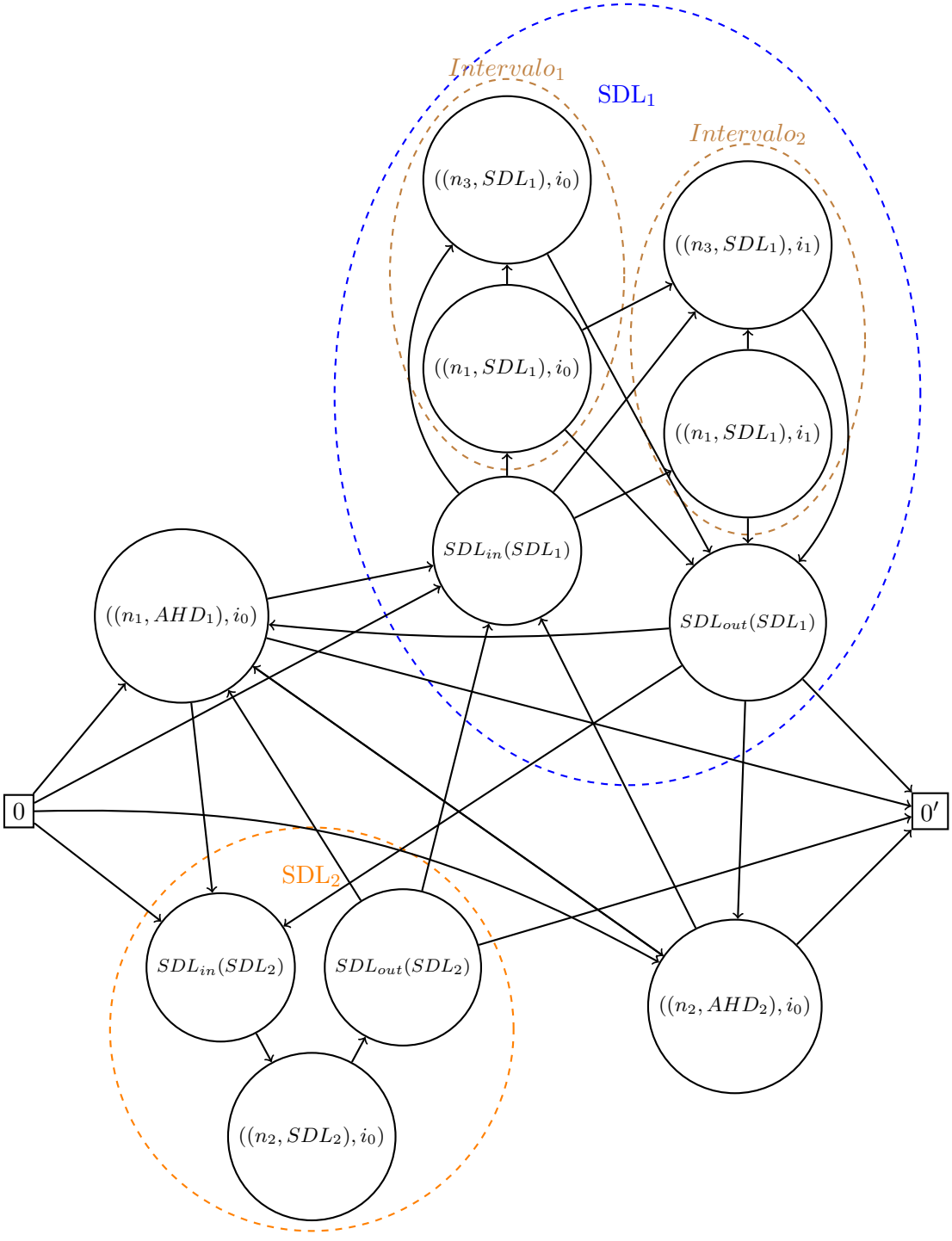

Fig. 4.4: Un posible grafo extendido generado a partir del grafo en la figura 4.2 con suposiciones sobre los intervalos de capacidad.

factible. Luego presentamos la función de extensión de recursos y las reglas de dominación. Por último, explicamos las mejoras que aplicamos para reducir los tiempos de cómputo.

Estructura del Label

Un camino parcial $p = (0, \ldots, v)$ comienza en el depósito inicial 0, termina en algún vértice $v \in V$ y tiene un label asociado $m = (Prev, V, Q, C, T, S, SDLK)$ donde $Prev$ es el label desde el cual se realizó esta extensión y que representa al camino parcial sin el último vértice, $V(m)$ es el último vértice visitado, es decir v, $Q(m)$ es la demanda acumulada atendida, $C(m)$ es el costo reducido acumulado, $T(m)$ es el tiempo más temprano en el que se puede comenzar el servicio en v para luego partir a destino, $S(m) \subseteq V_{ext}$ es un conjunto de vértices que no puede ser visitados o *inalcanzables* porque violan alguna restricción, por ejemplo que el camino sea simple o que no pueda realizar su servicio dentro de su ventana tiempo, y finalmente $SDLK(m)$ indica la capacidad utilizada de v si $v \in W_{ext}^{SDL}$, y está fijo en 0 en caso contrario. Este último valor solo tiene sentido cuando $v \in W_{ext}^{SDL}$, que es cuando es mayor a 0. Los demás vértices no presentan una restricción en cuanto a la capacidad de recepción que debamos considerar, por lo que en esos vértices vale 0.

A modo de ejemplo, sea $v \in SDL$ un punto SDL cuya función de capacidad $C_v(t)$ se corresponde con la figura 3.1. Veamos como el valor de $SDLK$ cambia a medida que seguimos una secuencia de extensiones desde un label que recorre este punto SDL. Al momento en que un label es extendido a la entrada del SDL, el valor de $SDLK$ es 0. Supongamos que una extensión del label anterior, entrega un pedido en el tiempo 9. Este nuevo label tiene un valor $SDLK$ de 1. Luego es extendido a un vértice que representa un pedido del siguiente intervalo. Esto simboliza que el vehículo espera hasta el comienzo del siguiente intervalo, es decir el tiempo 10, para entregar otro pedido, pasando a tener un valor $SDLK$ de 2 en el nuevo label extendido, ya que en conjunto con el pedido anterior está utilizando 2 unidades en el intervalo $[10, 14]$. Por último, el label se extiende a la salida del SDL, y a pesar de que los pedidos entregados siguen alojados en él, el valor $SDLK$ pasa a ser 0 nuevamente. Las siguientes extensiones de este label no podrán ser a vértices del mismo SDL, dado que no se permiten múltiples visitas a este tipo de puntos. Por esta razón no es necesario conservar un acumulado del uso de cada SDL, sino que basta con saber si se visitó o no.

El label inicial, que representa el camino parcial desde el depósito inicial (0), es $(\perp, 0, 0, 0, a_0, \{0\}, 0)$.

Función de Extensión de Recursos

Un label m con $V(m) = v_1 \in V_{ext}$ se extiende sobre el arco $(v_1, v_2) \in A_{ext}$, obteniendo el label $\hat{m}$, usando la siguiente *función de extensión de recursos*:

$$Prev(\hat{m}) = m \tag{4.3}$$
$$V(\hat{m}) = v_2 \tag{4.4}$$
$$Q(\hat{m}) = \begin{cases} Q(m) + q_{v_2} & v_2 \in W_{ext} \\ Q(m) & \text{de lo contrario} \end{cases} \tag{4.5}$$
$$C(\hat{m}) = C(m) + \bar{c}_{v_1,v_2} \tag{4.6}$$
$$T(\hat{m}) = \max(T(m) + s_{v_1} + t_{v_1,v_2} + p_{v_2}, a_{v_2}) \tag{4.7}$$
$$S(\hat{m}) = \begin{cases} S(m) \cup W_{ext}(n_{v_2}) & v_2 \in W_{ext} \\ S(m) \cup \{v_2\} & v_2 \in SDL_{in} \\ S(m) \cup W_{ext}(l_{v_2}) \cup \{v_2\} & v_2 \in SDL_{out} \\ S(m) & \text{de lo contrario} \end{cases} \tag{4.8}$$
$$SDLK(\hat{m}) = \begin{cases} SDLK(m) + 1 & v_2 \in W_{ext}^{SDL} \\ 0 & \text{de lo contrario} \end{cases} \tag{4.9}$$

Solamente los vértices que significan la atención de un pedido pueden aumentar la carga acumulada en el vehículo, y esto es lo que la indica (4.5). En el caso del costo, la función que definimos previamente captura todos los casos y (4.6) la usa. $T(\hat{m})$ tiene que indicar el tiempo más temprano posible en el cual el label m puede comenzar el servicio. Al extender puede ocurrir tanto que el inicio de la ventana de tiempo en el destino sea menor al tiempo que le toma al vehículo estar preparado para entregar el pedido, como al revés, que el vehículo llegue antes de que comience la ventana de tiempo en el destino, (4.7) considera ambos. Para forzar que el camino sea simple, siempre se puede agregar el vértice destino al que extendemos al conjunto de inalcanzables. De esta manera podemos por ejemplo forzar que los puntos $l \in L^{SDL}$ sean visitados a lo sumo una vez, ya que $SDL_{in}(l)$ queda incluido una vez visitado. En algunos casos tenemos más información, por ejemplo si el destino es la atención de un pedido, ninguna otra de sus opciones se podrá visitar porque una orden se atiende exactamente una vez. Por lo mismo, sabemos al salir de un punto $l \in L^{SDL}$, que ninguna de sus opciones $W_{ext}(l)$ será visitada, y podemos agregarla. Todo esto es capturado por (4.8). Como explicamos en el ejemplo de las extensiones de labels en un punto SDL, (4.9) indica que el recurso $SDLK$ incrementa en 1 solo cuando extendemos hacia una opción de entrega en un locker y sino es 0.

El label resultante $\hat{m} = (Prev, V, Q, C, T, S, SDLK)$ es válido o factible si:

$$Q(\hat{m}) \leq Q \tag{4.10}$$
$$T(m) + s_{V(m)} + t_{V(m),V(\hat{m})} + p_{V(\hat{m})} \leq b_{V(\hat{m})} \tag{4.11}$$
$$V(\hat{m}) \notin S(m) \tag{4.12}$$
$$SDLK(\hat{m}) \leq C_{l_{V(\hat{m})}}(T(\hat{m})) \tag{4.13}$$

Estas condiciones formalizan consideraciones y comportamientos esperados y explicados en la sección 3.1. De esta manera evitamos que los labels representen caminos infactibles de nuestro problema. Como dijimos, la demanda acumulada en un camino no puede exceder la capacidad del vehículo en ningún momento (4.10). El nuevo label $\hat{m}$ debe comenzar su servicio antes de que termine la ventana de tiempo asociada (4.11). El nuevo vértice final debe ser factible respecto a $S(m)$, por ejemplo no tiene que haber sido

visitado previamente, o no puede ser un pedido ya atendido en la ruta, esto es capturado por (4.12). Por último, la capacidad de recepción que $\hat{m}$ utiliza hasta el momento en el punto no puede ser mayor que la función de capacidad del punto en el tiempo del label.

En base al significado de S y la forma en que extendemos ese recurso, podemos establecer la siguiente observación:

Observación 1. *Dado que se permite una única visita a un punto $l \in L^{SDL}$ para una misma ruta $r \in \Omega$, un label m visitó un punto $l \in L^{SDL}$ si y solo si $SDL_{in}(l) \in S(m)$. Habiendo visitado al punto $l \in L^{SDL}$, el label m concluyó su visita en l si y solo si $SDL_{out}(l) \in S(m)$.*

Dominación de labels

Como dijimos antes, para reducir la cantidad de labels enumerados buscamos encontrar condiciones que nos permitan podar el espacio de búsqueda o descartar labels factibles, garantizando que el óptimo se mantiene. Llamamos *sufijos factibles* de un label m a todos los caminos desde $V(m)$ hasta el depósito final $0'$ que pueden ser extendidos arco a arco a partir de m generando todos labels factibles. Dados dos labels factibles m y $\hat{m}$ con $V(m) = V(\hat{m})$, y sean $SF(m)$ y $SF(\hat{m})$ el conjunto de los sufijos factibles de m y $\hat{m}$ respectivamente, si $SF(\hat{m}) \subseteq SF(m)$ y $C(m) \leq C(\hat{m})$, entonces para todo sufijo factible $\alpha \in SF(\hat{m})$ vale que $C(m + \alpha) = C(m) + C(\alpha) \leq C(\hat{m}) + C(\alpha) = C(\hat{m} + \alpha)$. Entonces, m puede extenderse con los mismos sufijos que $\hat{m}$ y generar rutas con igual o menor costo reducido. Esto significa que podemos descartar $\hat{m}$ y sus extensiones con la certeza de que de haber una ruta con costo reducido negativo formada a partir de él, también la habrá a partir de m.

En nuestro problema y con nuestra representación G_{ext}, en un mismo punto $l \in L^{SDL}$, podemos encontrar la misma relación de dominancia entre labels m y $\hat{m}$ cuyos vértices finales son $V(m), V(\hat{m}) \in W_{ext}(l)$ y $V(m) \neq V(\hat{m})$. Por ejemplo, en la figura 4.5, suponiendo que el SDL_1 tiene una capacidad de recepción de 3 o más, siendo m el label que representa el camino $(0, SDL_{in}(SDL_1), ((n_1, SDL_1), i_0))$ y $\hat{m}$ el label que representa el camino parcial $(0, ((n_1, AHD_1), i_0), SDL_{in}(1), ((n_2, SDL_1), i_0))$, se puede ver que vale $SF(\hat{m}) \subseteq SF(m)$. El primer arco de cualquier sufijo $\alpha \in SF(\hat{m})$ agrega el mismo costo reducido tanto a m como $\hat{m}$, porque los costos de viaje dentro de un mismo punto $l \in L^{SDL}$ son iguales, y el resto de α es igual para ambas extensiones del primer arco. Entonces a pesar de que $V(m) \neq V(\hat{m})$, si $C(m) \leq C(\hat{m})$, podemos descartar $\hat{m}$. Cabe destacar que dado $l \in L^{SDL}$ los labels m con $V(m) = SDL_{in}(l)$ no pueden dominar labels $\hat{m}$ con $V(\hat{m}) \in W_{ext}(l)$, dado que los vértices que representan opciones de entrega asociadas a l tienen un camino $\alpha \in SF(\hat{m})$ que pasa en un arco al vértice $SDL_{out}(l)$ y esto no es posible desde $SDL_{in}(l)$. Al mismo tiempo dado un punto $l \in L^{SDL}$, los labels m con $V(m) \in W_{ext}(l)$ no pueden dominar labels $\hat{m}$ con $V(\hat{m}) = SDL_{out}(l)$ dado que los $SF(m)$ tienen a $SDL_{out}(l)$ en todos sus caminos.

Para actualizar las reglas de dominación considerando estos casos, introducimos la función $Loc : V_{ext} \to (\mathbb{N})$ cuyas salida es la misma para las opciones de entrega asociadas a un mismo punto SDL, y distinta en cualquier otro caso. No se puede utilizar directamente la ubicación geográfica como Loc porque se quiere separar los vértices artificiales que significan la entrada y salida de los puntos L^{SDL}, de los vértices que representan opciones $W_{ext}(l)$. Loc es cualquier función definida sobre V_{ext} que cumple que para cada par de vértices $v_1, v_2 \in V_{ext}$, $Loc(v_1) = Loc(v_2) \Leftrightarrow (v_1, v_2 \in W_{ext} \subset V_{ext} \wedge l_{v_1} = l_{v_2} =$

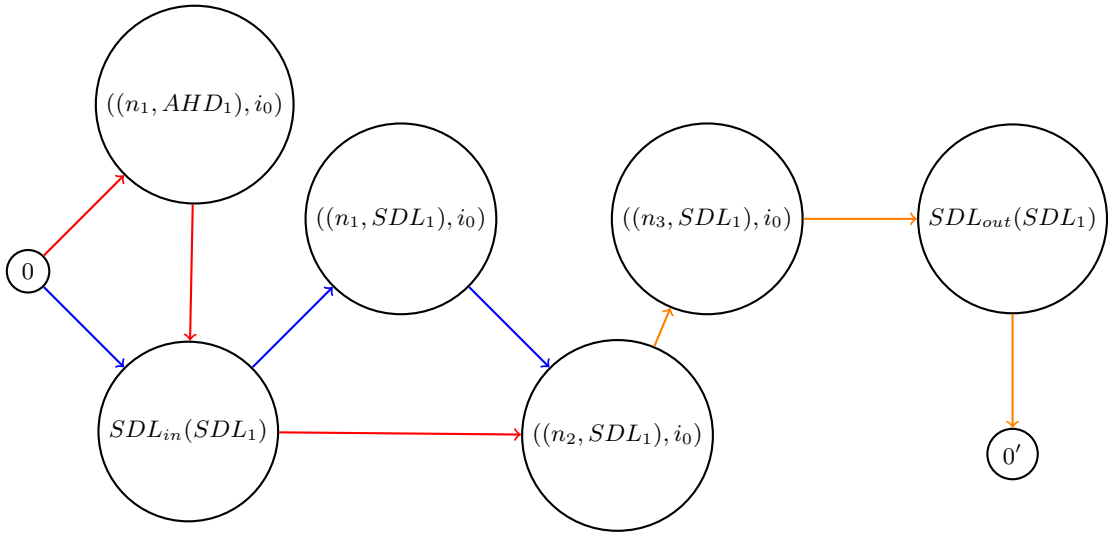

Fig. 4.5: Ejemplo con 3 pedidos ($\{n_1, n_2, n_3\}$), una opción AHD para n_1 asociada a AHD_1 y un SDL (SDL_1) en el que se se pueden atender todos los pedidos. SDL_1 tiene una capacidad de recepción de 2 unidades en su intervalo $i_0 \in T(SDL_1)$. En azul y en rojo, dos caminos parciales desde el depósito inicial 0 a los vértices $((n_1, SDL_1), i_0)$ y $((n_2, SDL_1), i_0)$ respectivamente. En naranja, un camino parcial desde el vértice $((n_2, SDL_1), i_0)$ al depósito final $0'$.

$l \in L^{SDL}$). A modo de ejemplo, en la figura 4.5, vemos que $Loc(SDL_{in}(SDL_1)) \neq Loc(((n_1, SDL_1), i_0))$ y $Loc(SDL_{in}(SDL_1)) \neq Loc(SDL_{out}(SDL_1))$, mientras que $Loc(((n_2, SDL_1), i_0)) = Loc(((n_1, SDL_1), i_0))$.

Como computar el conjunto SF de un label es costoso, presentamos condiciones suficientes para descartar un label. Decimos que m *domina a* $\hat{m}$ o $m \prec \hat{m}$ cuando:

$$Loc(m) = Loc(\hat{m}) \tag{4.14}$$
$$Q(m) \leq Q(\hat{m}) \tag{4.15}$$
$$C(m) \leq C(\hat{m}) \tag{4.16}$$
$$T(m) \leq T(\hat{m}) \tag{4.17}$$
$$S(m) \subseteq S(\hat{m}) \tag{4.18}$$
$$SDLK(m) \leq SDLK(\hat{m}) \tag{4.19}$$

Sin la condición (4.19), un label puede descartarse erróneamente. Un ejemplo de esto sucede en la figura (4.5), donde se ven dos caminos parciales desde el depósito 0. Sean $\hat{m}$ y m los labels asociados a los caminos parciales rojo y azul respectivamente. Estos satisfacen las condiciones (4.14), (4.15) y (4.18). Si se satisfacen las condiciones (4.16) y (4.17), $\hat{m}$ se marcaría como dominado por m a pesar de tener un sufijo factible, marcado en naranja, que no es factible para m debido a la capacidad de recepción del SDL_1.

En la perspectiva de los sufijos factibles no importa la capacidad de recepción que los labels m y $\hat{m}$ hayan utilizado de otro punto $l \in L^{SDL}$ distinto de $Loc(V(m))$ o $Loc(V(\hat{m}))$, sino más bien si lo utilizaron o no. Si ambos visitaron un punto $l \in L^{SDL}$, tienen su entrada en su conjunto S ya que también incluye visitados, es decir $SDL_{in}(l) \in S(m)$ y $SDL_{in}(l) \in S(\hat{m})$, y entonces ningún sufijo factible deja pedidos en l. Si solo uno de los

dos visitó a l, entonces el conjunto de sufijos factibles para alguno de los dos puede incluir caminos que lo visitan. En caso que m haya visitado l y $\hat{m}$ no, entonces $SDL_{in}(l) \in S(m)$, $SDL_{in}(l) \notin S(\hat{m})$ y m no podrá dominar $\hat{m}$ porque no vale 4.18.

Observación 2. *Dado un label m y su extensión $\hat{m}$, siempre vale que:*

- $Q(m) \leq Q(\hat{m})$ *porque a los sumo hay un pedido más en el vehículo.*
- $T(m) \leq T(\hat{m})$ *porque todos los tiempos que se pueden incurrir son no negativos.*
- $S(m) \subseteq S(\hat{m})$ *porque siempre se añaden vértices o se mantiene el mismo conjunto.*

Hay un conjunto de labels que no son descartados por dominancia y llamamos *labels procesados*. Así como haremos más adelante al analizar la complejidad del algoritmo 3, es una práctica frecuente en los algoritmos de labeling para ESPPRC analizar la complejidad en términos de la cantidad de labels procesados que se generan, y por lo general mientras más chico sea este conjunto, menor tiempo de cómputo necesita el algoritmo. Dicho esto, hay un compromiso constante entre reducir este conjunto en detrimento de una mayor cantidad de operaciones que se realizan por cada label, que puede llevarnos en la práctica a tiempos de cómputo más altos.

En nuestro caso, experimentos preliminares mostraron reducciones en los tiempos de cómputo al agregar dos procedimientos para reducir la cantidad de labels procesados. Ambos consisten en agregar más vértices al conjunto S cambiando su significado semántico, pasando de ser el conjunto de vértices visitados por el camino parcial que representa el label y las opciones de los pedidos entregados, a ser un conjunto de vértices *inalcanzables* a los que ya no se puede extender, por lo anterior y por la violación de alguna restricción sobre labels. La intuición de ambas ideas es que al agregar más vértices a S, es más fácil encontrar otros labels que cumplan la condición de dominación (4.18) y así es más fácil que un label sea dominado. También de esta forma reducimos la cantidad de valores distintos de S que se generen, y con esto la cantidad de labels generados y procesados.

El primero se basa en el hecho de que dentro de un punto $l \in L^{SDL}$, al entregar un pedido utilizando una opción extendida $w \in W_{ext}(l)$, todos los vértices que significan entregar un pedido $\hat{w} \in W_{ext}(l)$ en un intervalo previo, es decir $b_{\hat{w}} \leq a_w$, y los pedidos con índice menor, es decir $n_{\hat{w}} < n_w$, ya no podrán ser visitadas. Este conjunto de vértices que se añade a S se puede precomputar al inicio de la solución ya que es independiente de las iteraciones de la generación de columnas.

El segundo consta de dos partes. Primero se precomputa el valor *tiempo más tardío de partida* (TTP) para cada par de vértices del grafo. Como indica el nombre, dado el par de vértices $v_1, v_2 \in V_{ext}$, $TTP(v_1, v_2)$ indica el máximo tiempo en el que se puede partir de v_1 para llegar al vértice v_2 en un tiempo $t \in [a_{v_2}, b_{v_2}]$. Luego de extender un label $\hat{m}$, se itera sobre todos vértices del grafo, añadiendo a S aquellos que ya no son alcanzables dado el tiempo del label. Es decir, todo $v \in V_{ext}$ tal que $TTP(V(\hat{m}), v) < T(\hat{m})$.

4.1.3 Propiedades dentro de un punto de entrega compartido

Además de reducir la cantidad de arcos del grafo que representa la red logística, haber agregado los conjuntos de vértices sintéticos SDL_{in} y SDL_{out} nos permiten hacer la siguiente observación:

Observación 3. *Dado un label m_1 y su extensión m_2, si ambos tienen como vértice final una opción extendida dentro del mismo punto $l \in L^{SDL}$, entonces también satisfacen las condiciones (4.14) y (4.19).*

En este caso, dado $l \in L^{SDL}, l = l_{V(m_1)} = l_{V(m_2)}$, cualquier sufijo factible α desde m_2 hasta $SDL_{out}(l)$ $(V(m_2), \ldots, SDL_{out}(l))$ es también factible para m_1. Si además vale que $C(m_1) \leq C(m_2)$, entonces extendiendo arco a arco de α a partir de ambos labels m_1 y m_2, obtenemos dos nuevos labels $\hat{m}_1$ y $\hat{m}_2$ con $V(\hat{m}_1) = V(\hat{m}_2) = SDL_{out}(l)$, y $\hat{m}_1$ domina a $\hat{m}_2$. Por lo tanto, podemos decir que m_1 domina a m_2. Esto nos da una aplicación práctica: si ordenamos los arcos salientes de un vértice por costo reducido ascendentemente, podemos dejar de extender un label al encontrar el primer arco con costo reducido mayor o igual que 0 y pasar a procesar otro label nuevo. De esta forma generamos menos labels factibles que luego serían dominados y también aceleramos el procesamiento de cada label. Esto último se aprecia más cuando hay para cada vértice, pocos arcos con costo negativo en el grafo, algo que generalmente pasa en las primeras iteraciones de la generación de columnas, cuando hay menos variables duales Π_n asociadas a restricciones de atención (3.2) tomando valores positivos más grandes que el costo de viaje promedio. Cabe aclarar que sea cual fuese el costo del arco en el grafo, siempre se considera extender el label sobre el arco que va hacia la salida $SDL_{out}(l)$.

4.1.4 Pseudocódigo

El algoritmo 3 presenta un pseudocódigo del algoritmo de labeling explicado en esta sección. Este es una algoritmo exacto para el problema de pricing, que explora de forma exhaustiva todas las rutas factibles, realizando podas por optimalidad sobre el espacio de labels factibles a través de la dominación. Este tipo de algoritmos se los denomina *mono-direccional* dado que los labels se extienden en un solo sentido, desde el depósito inicial hacia el depósito final.

Hay dos estructuras que el algoritmo inicializa y actualiza constantemente durante su ejecución. Por un lado está el conjunto de labels procesados que denotamos D. Una explicación detallada sobre la estructura de este conjunto se encuentra en la sección 4.1.5. En segundo lugar está una cola de prioridad que denotamos U y almacena los labels factibles que aún no fueron considerados, es decir que solamente fueron creados o extendidos. Por cada label removido de U, se revisa primero si este es dominado por otro label en D. Si es dominado, se descarta, y de lo contrario, se procesa, que en este contexto significa agregarlo a D y probar extenderlo por cada uno de sus arcos salientes validando que cada extensión sea factible. A las extensiones que llegan al depósito final $(0')$ con un costo reducido negativo, se les reconstruye su camino parcial asociado utilizando $Prev$ y se agregan a R. A las extensiones válidas que no llegan al depósito final, se las encola en U. Por último, a las extensiones inválidas se las descarta. Los labels son procesados en orden de T, Q, C, $SDLK$ y $|S|$, de manera ascendente.

La función *Extender* genera un nuevo label como se explica en la sección 4.1.2. La función *EsFactible* devuelve un valor de verdad verdadero si el label parámetro de la función satisface todas las condiciones (4.10)-(4.13). Las extensiones que no son factibles se descartan. R tiene todas las rutas con costo reducido negativo encontradas. Podemos devolver todo el conjunto, sólo una con costo mínimo, o las primeras k rutas con costo mínimo. Esta es una decisión que se incluye en la configuración del algoritmo.

A continuación hacemos un breve análisis de la complejidad temporal del algoritmo 3.

input : Grafo dirigido G =(V,A), Instancia I de VRPDOTW-DC
output: Un conjunto de rutas R con costo reducido negativo

1 Inicializar $R = D = \emptyset, U = \{m_0\}$, donde m_0 es el label inicial $(\perp, 0, 0, 0, 0, 0, \{0\})$
2 **while** $U \neq \emptyset$ **do**
3 Tomar y remover un label m de U
4 **foreach** *label* $\hat{m} \in D$ **do if** $\hat{m} \prec m$ **then** ir a 2
5 $D \leftarrow D \cup m$
6 **foreach** $v \in \delta(V(m))^-$ **do**
7 Sea $\hat{m} =$ `Extender`(m, v)
8 **if** $\neg$`EsFactible`$(\hat{m})$ **then** ir a 6
9 **if** $v = 0'$ **then**
10 **if** $C(\hat{m}) < 0$ **then** $R \leftarrow R \cup$ `Camino`(m)
11 **else**
12 $U \leftarrow U \cup \hat{m}$
13 **end**
14 **end**
15 **end**

Algoritmo 3: Esquema general del algoritmo de labeling mono-direccional

Podemos pensar a los labels generados por el algoritmo como un árbol enraizado en el label inicial, cuyos vértices internos son labels procesados, y las hojas son labels dominados o dominantes sin extensiones factibles. Se ejecutan las líneas 3-5 por cada label extendido, y las líneas 6-15 por cada label procesado.

Validar si un label $\hat{m}$ domina a m toma $\mathcal{O}(|V|)$ en el peor caso dado que la operación de inclusión $S(\hat{m}) \subseteq S(m)$ es lineal sobre el tamaño de los conjuntos. En la práctica se utilizan conjuntos de tamaño acotado con una estructura similar a *bitset*, lo que hace que sea constante. Cada label extendido repite esta validación en el peor caso con todos los labels en D, es decir $\mathcal{O}(|D|)$ veces. Sea X la cantidad de labels factibles extendidos, las líneas 3-5 toman en total $\mathcal{O}(|X| \times |V| \times |D|)$ en el peor caso. Como cada label puede extenderse a los vértices en su vecindario de salida, podemos acotar $|X| \leq |D| \times |V|$ concluyendo en $\mathcal{O}(|V|^2 \times |D|^2)$. Esto es una cota en principio no ajustada, ya que en la práctica este árbol tiene menos ramificaciones cerca de las hojas, ya que los caminos son simples.

El ciclo 6-14 itera sobre los arcos salientes del vértice del label, que son $\mathcal{O}(|V|)$ en el peor caso, haciendo $\mathcal{O}(|V|)$ operaciones para *Extender*, *EsFactible* y *Camino*. En los primeros dos casos porque el conjunto de S tiene un tamaño lineal en la cantidad de vértices, y en el caso de *Camino* porque esta es una cota de la longitud de un camino simple en G. También insertar en U toma en el peor caso $\mathcal{O}(\log |X|)$. Estas lineas toman en total en el peor caso $\mathcal{O}(|D| \times (|V|^2 + \log |X|))$, que como $|V| \ll |D|$, podemos acotar $|X|$ y queda $\mathcal{O}(|D| \times (|V|^2 + \log(|D|)))$. Y en total todo algoritmo toma en el peor caso $\mathcal{O}(|V|^2 \times |D|^2 + |D| \times |V|^2 + |D| \log |D|) = \mathcal{O}(|V|^2 \times |D|^2)$.

4.1.5 Mejoras al labeling estándar

Estructura de dominación

Como vimos en la sección anterior, la fase de dominación es cuadrático en $|D| \times |V|$ en el peor caso entre todas las iteraciones, por lo que cualquier mejora en esta parte puede resultar en un impacto positivo para la performance total del algoritmo. Si bien hay muchos labels candidatos en D que pueden llegar a dominar al que removemos de U, se puede plantear una estructura especial para D y un orden de recorrido de los labels de U, de forma de reducir la cantidad de dominaciones no exitosas. Generalmente se procesan los labels de U en orden ascendente sobre algún recurso no decreciente, en nuestro caso utilizamos el tiempo. De esta forma el label que se remueve de U tiene un tiempo mayor o igual que todos los que están en D hasta el momento y la condición (4.17) se verifica siempre. Así mismo los labels en D se encuentran ordenados por costo reducido creciente y agrupados por buckets de 3 niveles: $(Loc, SDLK, \lfloor Q \rfloor)$. Entonces D primero separa los labels en casilleros por la función Loc con un arreglo. Luego cada casillero tiene un diccionario ordenado cuyas claves son valores de $SDLK$ de los labels con igual valor de Loc. Los valores de este diccionario son así mismo otro diccionario ordenado, cuyas claves son los valores de Q de los labels, tomando parte entera, con igual valor de Loc y $SDLK$. Por último, los valores de este diccionario son arreglos de labels ordenados por costo reducido creciente. De esta forma el conjunto de labels con el que se va a comparar verifica siempre las condiciones (4.14), (4.15) y (4.19).

Labeling bidireccional

Otra estrategia muy utilizada en la literatura para problemas similares, para reducir la cantidad de labels generados, consta de utilizar dos labelings mono-direccionales. Basado en la técnica *meet in the middle*, se extienden caminos desde el depósito inicial y desde el final en sentido opuesto (reverso), avanzando ambos hasta alguna condición de corte sobre un atributo que sea monótonamente no decreciente o no creciente, en nuestro caso elegimos el tiempo. Seguido de esto se intentan pegar los prefijos de rutas generados en el sentido normal con los sufijos de rutas generados a partir del labeling en sentido reverso.

Para utilizar la misma estructura semántica de los labels, función de extensión, condiciones de validez, estrategia de dominación y algoritmo, el labeling en sentido reverso trabaja sobre una réplica de la instancia de VRPDOTW-DC sobre el que se aplican las siguientes transformaciones:

- Si el horizonte de planeamiento es $[t_0, T]$, pasa a ser $[-T, -t_0]$.
- Todas las ventanas de tiempo $[a, b]$ pasan a ser $[-b, -a]$.
- El vértice inicial pasa a ser el depósito final, y el final, el depósito inicial.
- Se define un nuevo conjunto de arcos A_R donde para cada arco $(v_1, v_2) \in A$, hay un arco $(j, i) \in A_R$ con costo reducido $\bar{c}_{i,j}$ y tiempo de viaje $t_{i,j} + s_i + p_j$.
- El reverso utiliza el grafo extendido $G_R = (V, A_R)$.
- Los vértices de entrada SDL_{in} son los nuevos de vértices de salida y los de salida SDL_{out} son los nuevos vértices de entrada.

Fig. 4.6: Función de capacidad de recepción del locker de la figura 3.2.1 transformado para ser operado en sentido inverso. Su ventana de tiempo es ahora $[-22, -8]$, comenzando con una capacidad de recepción de 3 pedidos. En azul están marcados los momentos en los que se retiran pedidos, decrementando la capacidad.

El label inicial del reverso, que representa el sufijo (0'), es $m_{0'} = (\perp, 0', 0, 0, -T, \{0'\}, 0)$.

En el reverso, avanzar de intervalo significa que el SDL pierde unidades de capacidad para atender pedidos. Para ejemplificar esto, en la figura 4.6 se ve el resultado de aplicar la transformación descrita sobre el locker de la figura 3.2.1. Lo primero que se puede remarcar, es que los tiempos en el eje de abscisas son negativos, esto se debe a que el labeling en dirección reverso comienza en el tiempo -T, y suman tiempos de viaje positivos, aproximándose al valor 0. Supongamos que hay un label m reverso cuyo vértice final es una opción de entrega en dicho locker, durante el intervalo $[-22, -14]$. Este tiene un valor de $SDLK$ de 1. Una posible extensión de m es por un arco a otra opción dentro del mismo intervalo, representando que se deja un pedido en el mismo intervalo y pasando a tener un SDLK de 2. Otra extensión posible de m es por un arco a una opción del intervalo $[-14, -10]$, representando en el sentido normal, que un pedido se entrega en el segundo intervalo y luego se espera para entregar otro en el tercero (el que entrega m). En el caso de la última extensión, la capacidad del SDL cambia de 3 a 2, desapareciendo el casillero que se vuelve en orden derecho a las 14 horas. En el orden reverso, consideramos que primero se usan los casilleros, entre los disponibles, que están por desaparecer, o que más tarde se hicieron disponibles en el orden normal. Entonces el casillero ocupado entre las $[-22, -14]$ desaparece a las -14 horas y $SDLK$ pasa a ser 0, luego entrega el pedido que iba a dejar en el intervalo $[-14, -10]$ y $SDLK$ pasa a ser nuevamente 1. Si este último label, se extiende para realizar una entrega en el período $[-10, -8]$, se repite el mismo procedimiento. Una vez que un label es extendido al vértice que simboliza la salida del locker en el sentido reverso, $SDLK$ pasa a valor 0, como ocurre en el sentido normal al salir de un SDL.

Para contemplar esto en la función de extensión, debemos modificar 4.9 de la siguiente forma:

$$SDLK(\hat{m}) = \begin{cases} SDLK(m) + WSDL(V(\hat{m})) & C_{l_{V(m)}}(T(m)) \leq C_{l_{V(\hat{m})}}(T(\hat{m})) \\ ActSDLK(m, \hat{m}) + WSDL(V(\hat{m})) & \text{caso contrario} \end{cases} \tag{4.20}$$

donde la función $ActSDLK(m, \hat{m})$ devuelve el nuevo valor de SDLK, teniendo en cuenta

que se está pasando a un intervalo con menor capacidad de recepción, esto se calcula de la siguiente forma: $\max(0, SDLK(m) - (C_{l_{V(m)}}(T(m)) - C_{l_{V(\hat{m})}}(T(\hat{m}))))$. La función $WSDL(v)$ devuelve 1 si el vértice es una opción extendida de entrega en el vértice $v \in W_{ext}^{SDL}$ y 0 sino.

Ambos labelings comienzan con su label inicial, y para balancear la extensión de labels en cada dirección, se extiende el labeling con menos labels en su cola de prioridad. El punto de corte sobre el tiempo se decide para cada instancia de manera dinámica, modificándose en ambos sentidos, siguiendo las indicaciones de [2] hasta que llegan ambos al mismo tiempo t y $-t$ respectivamente. Al fijarse el punto de corte, como las colas de prioridad de ambos labelings ordenan los labels primero por T, podemos avanzar ambos labelings hasta que el tope de sus respectivas colas pasen la condición de corte o terminen. Descartamos los labels encolados del labeling reverso, y los del sentido normal los intentamos unir con sufijos factibles de forma que se obtenga una ruta con costo reducido negativo. Estos sufijos factibles se buscan en la estructura de dominación del reverso, que denotamos D_R.

Sea $f \in D$ un label factible procesado en el sentido normal. Sea $\hat{f} \in U$ una extensión de f que pasó el punto de corte. Sea $b \in D_R$ un label factible procesado en el sentido reverso. Decimos que el label reverso b es un sufijo factible del label normal $\hat{f}$ si se cumplen las siguientes condiciones:

$$V(f) = V(b) \tag{4.21}$$

$$Q(f) + Q(b) \leq Q \tag{4.22}$$

$$C(f) + C(b) < 0 \tag{4.23}$$

$$T(f) - T(b) < T \tag{4.24}$$

$$(\forall v \in Camino(f))\ v \notin S(b) \vee (\forall v \in Camino(b))\ v \notin S(f) \tag{4.25}$$

$$SDLK(f) + SDLK(b) \leq C_{l_{V(\hat{f})}}(T(\hat{f})) \tag{4.26}$$

Notar que en (4.25) no se puede utilizar la condición tradicional $S(\hat{f}) \cap S(\hat{b}) = \{V(b)\}$ debido a que S no significa ahora el camino recorrido, sino que también contiene las opciones de cada pedido y al extender a los vértices $v \in SDL_{out}$, se agregan también todas las opciones dentro del mismo aunque no hayan sido atendidos. A modo de ejemplo, en la figura 4.4, podemos tomar un label $\hat{f}$ que represente el camino parcial $(0, ((n_1, AHD_1), i_0), SDL_{in}(SDL_1))$ y otro label b que represente al camino parcial reverso $(SDL_{in}(SDL_1),\ ((n_3, SDL_1), i_0),\ SDL_{out}(SDL_1),\ 0')$, y entonces $S(\hat{f}) \cap S(b) = \{SDL_{in}(1), ((n_1, SDL_1), i_0), ((n_1, SDL_1), i_1)\}$. Si bien la condición (4.23) no es necesaria para que b sea un sufijo factible de $\hat{f}$, se agrega porque solo nos interesan las rutas con costo reducido negativo. Utilizamos igualdad de vértices en la condición (4.21) en lugar de *Loc* porque los valores de SDLK usados en (4.26) de cada label representan cuanto usaron de la capacidad en el intervalo en que se encuentran. Al utilizar *Loc*, puede que $n_{V(\hat{f})} \neq n_{V(b)}$, por lo que hay que cambiar la condición (4.26) como también deberían agregarse condiciones sobre el orden de los pedidos $n_{V(\hat{f})} \leq n_{V(b)}$ y los intervalos $i_{V(\hat{f})} \leq i_{V(b)}$. Es por esto que para la estructura D_R no agrupa labels utilizando *Loc* sino que lo hace por vértice. Para acelerar el proceso de unión, se mantiene una segunda estructura con los labels de D_R pero que indexa estos por su último arco transitado en lugar de último vértice. De esta forma, $\hat{f}$ se intenta unir solamente con aquellos labels reversos cuyo último arco extendido sea $(V(b), v')$, con $v' \in \delta^-(V(\hat{f})) - S(\hat{f})$.

La técnica que explicamos se la conoce como *labeling bidireccional asimétrico* y se puede leer más del tema en [2].

4.1.6 Heurísticas para el problema de pricing

Incluso con las mejoras detalladas en la sección 4.1.5, resolver el problema de pricing con una solución exacta resulta computacionalmente costoso. Debido a esto, es una práctica común resolverlo de forma heurística. En este contexto, el comportamiento esperado de una heurística es que si devuelve una ruta para agregar al PMR, entonces es factible y tiene costo reducido negativo, y si falla, no hay certeza de que una ruta factible con costo reducido negativo no exista, por lo que no hay certeza de que estemos en el óptimo de la relajación lineal.

El procedimiento es entonces ejecutar una o más heurísticas, y si todas fallan, se ejecuta el algoritmo exacto descrito en la sección 4.1.2. Algo que se hace en la práctica cuando se usan varias heurísticas, es ordenarlas por tiempo de cómputo creciente. Ejecutando las más costosas sólo cuando todas las anteriores fallaron.

A lo largo de esta tesis nos encontramos con iteraciones de la generación de columnas en las que la cantidad de labels procesados en los vértices $v \in V_{ext}$ ubicados en puntos L^{SDL} se vuelve inmanejable. Dado un punto $l \in L^{SDL}$, este tipo de escenario surge cuando las opciones que pueden ser atendidas en el punto $w \in W_{ext}(l)$, están asociadas a pedidos $n \in N$ cuyas variables duales Π_n relacionadas a las restricciones (3.2) de atención de pedidos, toman todas valores positivos altos. En estos casos atender un pedido más en l reduce el costo reducido, entonces dos labels $m, \hat{m}$ con $l_V(m) = l_V(\hat{m}) = l$, que validan la condición 4.18, generalmente no cumplen la condición 4.16. Como resultado menos labels son dominados.

En estos casos aplica una estrategia común para desarrollar heurísticas, que consta de relajar reglas de dominación. Hemos desarrollado dos heurísticas del primer tipo: la primera elimina la condiciones (4.18) y (4.19), y la segunda solo (4.18). Relajar reglas de dominación significa permitir que un label domine a otro cuando en realidad no debería, en estos casos puede ocurrir que el label dominante no pueda ser extendido con todas los sufijos factibles del dominado. El resultado de esto es una reducción en el conjunto de labels procesados, lo cual vimos tiene gran impacto en la performance del algoritmo. Por ejemplo, sean $\hat{m}$ y m los labels asociados a los caminos parciales rojo y azul de la figura 4.5 respectivamente. En la primera heurística m puede dominar a $\hat{m}$. En el caso de la segunda heurística, un label que represente el camino rojo desde 0 hasta el vértice $SDL_{in}(SDL_1)$ puede dominar a otro que represente el camino azul desde 0 hasta el mismo vértice. La primera heurística está pensada específicamente para reducir la cantidad de labels procesados en los puntos L^{SDL}.

Otra estrategia para desarrollar heurísticas es agregar más condiciones para que un label sea factible, agregando restricciones, que no son realmente necesarias. El resultado esperado es el mismo que antes. Hemos desarrollado la siguiente heurística.

4.1.7 Limited Discrepancy Search

Limited Discrepancy Search (LDS) es un concepto de *constraint programming* que se introdujo en programación lineal entera por primera vez en [7]. Esta técnica permite, de forma heurística, acelerar los tiempos de ejecución del algoritmo de pricing buscando un compromiso con el espacio de búsqueda. Para aplicar LDS, se extiende la estructura de cada label m con un recurso nuevo que llamamos $ArcosMalos(m) \in \mathbb{N}_{\geq 0}$. Este vale 0 para el label inicial y puede incrementarse en labels extendidos. Para cada vértice $v \in V_{ext}$, se define su conjunto de *arcos buenos* $A^+(v)$ como los GN arcos salientes con menor costo

reducido más el arco $(v, 0')$ si existe, y sus *arcos malos* $A^-(v)$ como el resto de sus arcos salientes. $GN \in \mathbb{N}$ indica el tamaño del vecindario bueno y es un parámetro de la heurística. El valor $ArcosMalos$ para un label $\hat{m}$ extendido a partir de m se calcula de la siguiente manera:

$$ArcosMalos(\hat{m}) = \begin{cases} ArcosMalos(m) & (V(m), V(\hat{m})) \in A^+(V(m)) \\ ArcosMalos(m) + 1 & \text{de lo contrario} \end{cases} \tag{4.27}$$

Como generalmente el costo está dado en función de la distancia euclidiana, se espera que los tramos de menor longitud sean aquellos con menor costo asociado en el grafo. En este sentido $ArcosMalos(m)$ se puede interpretar como el número de veces que se elige "alejarse" de la zona un pedido, de una opción extendida en este caso, a lo largo del camino parcial que representa el label.

A las condiciones de factibilidad (4.1.2) de un label, se agrega también ahora que para todo label m:

$$ArcosMalos(m) \leq DISC \tag{4.28}$$

donde $DISC \in \mathbb{N}$ es otro parámetro de la heurística.

Aunque esto se presenta como una heurística, se puede hacer la siguiente observación:

Observación 4. *Si $DISC$, o GN toman valores suficientemente grandes, la heurística equivale al algoritmo 3*

Por ejemplo, en una instancia con 25 vértices en su grafo asociado, todos los labels tienen un valor de ArcosMalos menor o igual a 25, por lo que elegir este como DISC es lo mismo que no aplicar ninguna poda sobre el conjunto de labels factibles, y así tampoco sobre el de labels procesados. En el mismo ejemplo, fijar GN en 25 hace que todos los arcos sean buenos.

En [7] se ejecutan varias heurísticas LDS con el parámetro GN fijo en 2 y diferentes valores para DISC. El procedimiento consiste en comenzar con un valor de DISC de 1, que se incrementa en 1 cada vez que la heurística falla, hasta llegar a un límite. Este límite es $\frac{\text{DISC}_{\max}}{4}$, donde $\text{DISC}_{\max}$ es la cantidad máxima de vértices que pueden ser atravesados en una ruta. Este valor incluye también a los depósitos y los vértices $v \in SDL_{in} \cup SDL_{out}$ incluidos en la secuencia de la ruta. Si todas las heurísticas fallan, se ejecuta nuevamente LDS pero de manera exacta, con $\text{DISC}_{\max}$.

Se puede calcular una cota superior de $\text{DISC}_{\max}$ por medio de un problema de la mochila, donde los productos son los vértices y los pesos son las demandas. Como no todos los vértices tienen una demanda asociada, hay que forzar que sean incluidos en la solución. Este es el caso de los vértices sintéticos agregados, como el depósito final, SDL_{in} y SDL_{out}. La formulación de nuestro problema de la mochila es el siguiente:

$$z_{\text{demanda}} = \max\ 1 + 2 * |\text{SDL}| + \sum_{n \in N} y_n \tag{4.29}$$

$$\text{s.t.} \sum_{n \in N} q_n\ y_n \leq Q \tag{4.30}$$

$$y_n \in \{0, 1\} \qquad \forall n \in N \tag{4.31}$$

Este se puede resolver de manera óptima con la heurística del *menor-peso-primero*, ya que los objetos tienen costos unitarios en la función objetivo.

El algoritmo de LDS es muy similar al esquema genérico de labeling mono-direccional que se muestra en el algoritmo 3. Las diferencias son que el label tiene ahora un atributo más, $ArcosMalos$, `Extender` debe manipular este atributo como se describe en (4.27), y `EsFactible` debe considerar también la condición (4.28). Solo el primer cambio se podría apreciar en el esquema, dado que no se detalla la implementación de las dos funciones. Validar esta nueva condición de factibilidad, así como también actualizar el valor $ArcosMalos$ al extender un label, toman tiempo constante, por lo que la misma complejidad temporal en el peor caso es la misma.

4.1.8 Configuración de la solución de labeling

Hasta ahora se detalló una solución al problema de pricing que se compone de una secuencia de heurísticas, seguida de la ejecución de un labeling exacto. Qué heurísticas, qué algoritmo exacto, en qué orden, y agregando cuántas rutas en cada iteración, son parámetros de la solución. Hay muchas configuraciones de parámetros posibles, y es trabajo de la sección 5 elegir una buena.

4.1.9 Conjunto inicial $\hat{\Omega}$

Para poder utilizar generación de columnas necesitamos partir de un problema cuya relajación lineal es factible, es decir, que haya al menos una solución con variables reales que satisface todas las restricciones. Por ejemplo, si las rutas consideradas solo atienden la mitad de los clientes, incluso tomando valores reales nunca se pueden satisfacer las restricciones de atención de clientes (3.2) de la otra mitad. Por esto, inicializamos el PMR con una ruta ficticia que atiende a todos los pedidos sin utilizar ningún recurso de los SDL. Como no queremos que esta ruta sea parte de la solución su costo en la función objetivo es varios órdenes de magnitud más grande que el promedio. En nuestro caso utilizamos el valor 10^6.

4.2 Branching

Si al resolver la relajación lineal de la formulación (3.1)-(3.4) obtenemos una solución fraccionaria, es necesario aplicar un algoritmo branch & bound como el explicado en el algoritmo 1. En este capítulo explicamos las ideas detrás de las dos reglas de branching utilizadas en el esquema de branch and price.

4.2.1 Regla tradicional de VRP

Hay reglas de branching llamadas *robustas*, que tienen la característica de no cambiar la estructura del problema de pricing. Este es el caso de la regla que explicamos a continuación. Dados un subconjunto de rutas $\hat{\Omega} \subseteq \Omega$, un arco de la red $(i,j) \in A$ y una solución (fraccionaria) de la formulación (3.1)-(3.4) $\bar{\lambda}$, definimos al flujo sobre (i,j) asociado a $\hat{\Omega}$ como $\bar{e}_{ij} = \sum_{r \in \hat{\Omega}} \bar{\lambda}_r \, e^r_{ij}$, donde e^r_{ij} es 1 si el arco (i,j) es atravesado por la ruta r y 0 de lo contrario. Si bien varias rutas pueden participar con una pequeña contribución menor a 1 cada una, el flujo es intuitivamente la cantidad de vehículos que transitan un tramo de la red cuando nos limitamos a considerar un cierto conjunto de rutas. Decimos que el flujo asociado a $\hat{\Omega}$ es entero cuando para todo arco $(i,j) \in A$ vale que $\bar{e}_{ij} \in \mathbb{N}_0$. Definimos la matriz de flujo asociada al conjunto $\hat{\Omega}$ como una matriz $M \in \mathcal{R}^{|V| \times |V|}$ donde cada entrada de la matriz es $M_{i,j} = \bar{e}_{ij}$ asociado a dicho conjunto. Usamos $\hat{\Omega}$ para denotar el conjunto de variables de rutas consideradas en el PMR.

Proposición 1. *Cuando $\bar{\lambda}$ es una solución factible entera de la formulación (2.9)-(2.11), el flujo asociado a Ω, el conjunto de todas las rutas, es entero.*

En el VRP tradicional, un vértice que no es el depósito representa la atención de un pedido. Estos solo se pueden visitar (o atender) una vez, y por esto tienen también un solo sucesor. Esto significa que en una solución factible entera, todos los flujos $\bar{e}_{ij}$ asociados al subconjunto de rutas $\hat{\Omega}$ son 0 o 1, y el flujo es entero. Como todas las rutas $r \in \Omega, r \notin \hat{\Omega}$ no son seleccionadas en la solución $\bar{\lambda}$, los valores $\bar{\lambda}_r$ valen 0 y no aportan cambios al flujo al considerarlas, por lo que lo anterior vale para Ω también. La regla de branching tradicional de VRP, hace uso de la proposición 1. Ante una solución fraccionaria se elige el arco $(i,j) \in A$ cuyo flujo $\bar{e}_{ij}$, asociado al conjunto $\hat{\Omega}$, sea más cercano a 0.5. Luego se ramifica sobre el valor de este flujo a través de las desigualdades $\bar{e}_{ij} \leq 0$ y $1 \leq \bar{e}_{ij}$. En el caso del VRP tradicional, también vale que cuando todos los flujos $\bar{e}_{ij}$ asociados al subconjunto de rutas $\hat{\Omega}$ son enteros, la solución $\bar{\lambda}$ asociada es entera. Esto nos dice que estas reglas son suficientes para resolver instancias de VRP de manera exacta.

En nuestro problema, si bien los pedidos también son atendidos exactamente una vez como en el VRP tradicional, la visita (o atención) puede ocurrir en cualquiera de sus opciones definidas. Con esto, todo vértice opción $w \in W_{ext}$ pueden tener a lo sumo un predecesor y sucesor. Luego vale también que en una solución factible entera $\bar{\lambda}$ de la formulación (3.1)-(3.4), todos los arcos $(i,j) \in (W_{ext} \times V) \cup (V \times W_{ext})$ que inciden en al menos un vértice opción, tienen un flujo $\bar{e}_{ij}$ asociado a $\hat{\Omega}$ con valor 0 o 1. Por otro lado, nuestra red subyacente se representa con un grafo G_{ext} que agrupa las opciones por ubicación, y agrega dos vértices artificiales por cada $l \in L^{SDL}$, que pueden ser visitados más de una vez entre todas las rutas. Un ejemplo de esto es una instancia con todas sus opciones en un mismo $l \in L^{SDL}$, para lo cual se necesitan al menos dos vehículos para transportar toda la demanda, por lo que la entrada y salida de l se visitan al menos dos

veces, y tienen potencialmente al menos dos sucesores distintos. La figura 4.7 es un contraejemplo de la proposición 1 para la formulación (3.1)-(3.4). En este ejemplo, las rutas consideradas por el RMP son $r_0 = (0, SDL_{in}(SDL_1), ((n_1, SDL_1), i_0), SDL_{out}(SDL_1), ((n_3, AHD_1), i_0), 0')$, $r_1 = (0, SDL_{in}(SDL_1), ((n_1, SDL_1), i_0), SDL_{out}(SDL_1), 0')$, $r_2 = (0, SDL_{in}(SDL_1), ((n_2, SDL_1), i_0), SDL_{out}(SDL_1), 0')$ y $r_4 = (0, SDL_{in}(SDL_1), ((n_2, SDL_1), i_0), SDL_{out}(SDL_1), ((n_3, AHD_1), i_0), 0')$, y sus variables asociadas son respectivamente λ_0, λ_1, λ_2 y λ_3. Dada la solución fraccionaria $\bar{\lambda} = (0.5, 0.5, 0.5, 0.5)$, se puede ver fácilmente que el flujo asociado a $\hat{\Omega}$ es entero porque por cada arco $(i, j) \in A_{ext}$ hay exactamente dos rutas atravesándolo ($\bar{e}_{ij} = 1$), o ninguna ($\bar{e}_{ij} = 0$).

La estructura general de este tipo de contraejemplos consta de dos subconjuntos de rutas $\Omega_1, \Omega_2 \subset \Omega$ y cuyas rutas $r \in \Omega_1 \cup \Omega_2$ tienen un valor asociado en la solución $0 < \bar{\lambda}_r$. Las rutas $r \in \Omega_1$ tienen un mismo prefijo, que consta de un camino desde el depósito 0 hasta algún vértice $v \in SDL_{in} \cup SDL_{out}$, en este caso $SDL_{out}(SDL_1)$. En la figura 4.7, $\Omega_1 = \{r_0, r_1\}$ y $\Omega_2 = \{r_2, r_3\}$. Lo mismo sucede con las rutas $r \in \Omega_2$, y el mismo vértice anterior v. A partir de v, se forman dos nuevos conjuntos de rutas no vacíos $\Omega_3 \cup \Omega_4 = \Omega_1 \cup \Omega_2$, donde todas las rutas $r \in \Omega_3$ tienen un mismo sufijo desde v, en este caso desde $SDL_{out}(SDL_1)$ hasta el depósito final $0'$, y de manera análoga sucede con Ω_4. En la figura 4.7, $\Omega_3 = \{r_1, r_3\}$ y $\Omega_4 = \{r_0, r_2\}$.

En Tilk et al [1] se propone un esquema de branching de cuatro niveles: primero se decide sobre la cantidad de vehículos, luego sobre cuál es la opción elegida para cada pedido, seguido de la regla clásica de VRP mencionada anteriormente (con leves modificaciones), y por último si falla lo anterior se utiliza una implementación del método de *flow-splitting*. Este último nivel se añade para resolver casos como el ejemplo de la figura 4.7.

4.2.2 Método flow-splitting

Este método fue introducido en [6] y se basa en la siguiente proposición:

Proposición 2. *Una solución $\bar{\lambda}$ de la formulación (2.9)-(2.11) es entera si y solo si el flujo asociado a cualquier subconjunto de rutas $\Lambda \subseteq \Omega$ es entero.*

A partir de esta podemos armar un procedimiento para buscar desigualdades cuando estamos frente a una solución fraccionaria $\bar{\lambda}$ con la cual la regla tradicional falla, que consiste en primero buscar un subconjunto de rutas $\Lambda \subseteq \Omega$ que no tenga asociado un flujo entero. Luego, elegimos un arco $(i, j) \in A$ que incida en al menos una opción $w \in W_{ext}$ cuyo flujo $\bar{e}_{ij}$, asociado a Λ, sea más fraccionario, y ramificamos sobre este valor. Mientras nos aseguremos de estar considerando todos los subconjuntos $\Lambda \in \mathcal{P}(\Omega)$ necesarios, que potencialmente son todos, estas reglas son suficientes para resolver instancias de nuestro problema de forma exacta. La estrategia para recorrer el conjuntos de partes $\mathcal{P}(\Omega)$ es algo a decidir. Por ejemplo, una forma es recorrerlos en orden de cardinalidad decreciente, desempatando con otra métrica. En la práctica podemos intentar aprovechar propiedades del problema que queremos resolver para hacerlo más eficiente.

Para que nuestra solución sea exacta, proponemos una regla de branching distinta a la presentada en Tilk et al, pero que es también una implementación del método *flow-splitting*.

4.2.3 Aplicación del método flow-splitting

Proponemos como espacio de búsqueda los siguientes conjuntos $\Lambda \in \mathcal{P}(\Omega)$:

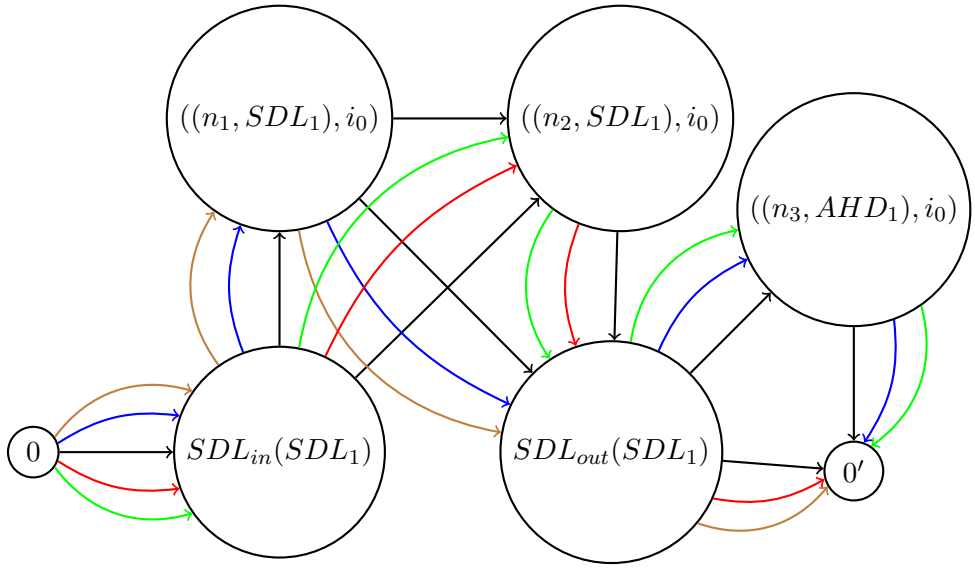

Fig. 4.7: Ejemplo de una instancia con 1 SDL y 1 AHD. Los arcos $(i, j) \in A_{ext}$ de la red están dibujados en negro, mientras que las rutas consideradas en el PMR con colores: r_0 (azul), r_1 (marrón), r_2 (roja) y r_3 (verde).

- Λ_{AHD}: el conjunto de todas las rutas $r \in \Omega$ que solo tienen opciones de tipo AHD y los depósitos.

- Λ_{ew}: el conjunto de todas las rutas $r \in \Omega$, tal que el arco $(e, w) \in A_{ext}$ con $e \in SDL_{in}, w \in W_{ext}(l_e)$ es atravesado por la ruta r.

Los conjuntos Λ_{ew} pueden intersecarse, dado que una misma ruta r puede visitar dos o más puntos SDL. También, la unión de estos conjuntos son todas las rutas, $\Omega = \Lambda_{AHD} \cup_{e \in SDL_{in}} \cup_{w \in W_{ext}(l_e)} \Lambda_{ew}$. Esto se debe a que si una ruta r tiene en su secuencia al menos una opción $w \in W_{ext}$ ubicada en un punto $l_w \in L^{SDL}$, entonces su secuencia contiene también al vértice entrada $e \in SDL_{in}$ con $l_e = l_w$, y hay un arco $(e, \tilde{w}) \in A_{ext}, (e, \tilde{w}) \in r$, con $\tilde{w} \in W_{ext}(l_e)$. En caso contrario, r contiene solo opciones en puntos $l \in L^{AHD}$, por lo que $r \in \Lambda_{\text{AHD}}$.

A continuación proponemos y demostramos proposiciones para concluir con reglas de branching que nos permiten resolver instancias de VRPDOTW-DC de forma exacta.

Proposición 3. *Si el flujo asociado a Λ_{AHD} es entero, entonces todas las variables relacionadas a rutas en Λ_{AHD} tienen un valor binario en la solución.*

Proof. Sea G_{ext} la red detallada en la sección 4.1.1. Sea G_{AHD} el subgrafo de G_{ext} inducido por el conjunto de vértices $V_{ext}^{AHD} = \{0, 0'\} \cup W_{ext}^{AHD}$. Notar que V_{ext}^{AHD} son los vértices visitados por las rutas $r \in \Lambda_{\text{AHD}}$. Como partimos de una solución $\bar{\lambda}$ de la formulación (3.1)-(3.4), las restricciones (3.3) relacionadas a la capacidad utilizada de los SDL se asumen satisfechas y las ignoramos. La proposición 1 y su vuelta valen ahora para el problema restringido a la red G_{AHD}. Es decir que todo vértice salvo los depósitos son visitados (o atendidos) a lo sumo una vez en una solución óptima, y por lo tanto tienen un solo sucesor. Luego el lema es verdad. □

Proposición 4. *Si el flujo asociado a Λ_{ew} es entero, entonces todas las variables relacionadas a rutas $r \in \Lambda_{ew}$ tienen un valor binario en la solución.*

Proof. Sea $\bar{\lambda}$ la solución fraccionaria de la que partimos. Como $w \in W_{ext}(l_e)$ representa una opción de entrega y un pedido es atendido exactamente una vez en una solución óptima por la restricción (3.2), el flujo $\bar{e}_{ew}$ asociado a Λ_{ew} es binario. Si es 0, entonces todas las rutas $r \in \Lambda_{ew}$ están asociadas a valores en la solución $\bar{\lambda}_r$ que son 0.

Supongamos que es 1. Sea $r \in \Lambda_{ew}$ una ruta asociada a un valor $\bar{\lambda}_r$ fraccionario. Sea $\hat{\Lambda}_{ew} \subseteq \Lambda_{ew} - \{r\}$ el conjunto de todas las rutas $k \in \Lambda_{ew}$ asociadas a un valor $\bar{\lambda}_k$ fraccionario, sin incluir r. Vale que $\bar{e}_{ew} = 1 = \bar{\lambda}_r \; e^r_{ew} + \Sigma_{k \in \Lambda'_{ew}} \bar{\lambda}_k \; e^k_{ew}$. Como son rutas distintas a r, hay al menos un arco $z \in r$ que no es atravesado por todas las rutas en $\hat{\Lambda}_{e,w}$. Entonces el flujo asociado a Λ_{ew} sobre z es fraccionario, lo que contradice las precondiciones. □

Proposición 5. *Si los flujos asociados a Ω y todo conjunto Λ_{ew} son enteros, entonces el flujo asociado a Λ_{AHD} es entero.*

Proof. Sea M la matriz de flujo asociada a Ω, $\hat{M}$ la matriz de flujo asociada a $\cup_{l \in SDL} \cup_{j \in W_{ext}(l)} \Lambda_{elj}$, y $M^* = M - \hat{M}$.

Como M y $\hat{M}$ son matrices enteras y para todo arco $(i,j) \in A$ vale $\hat{M}_{ij} \leq M_{ij}$, M^* es una matriz entera donde todo arco $(i,j) \in A$ cumple $0 \leq M^*_{ij}$. M^* es la matriz de flujo asociada al conjunto de rutas en Ω que no pasan por los puntos L^{SDL}, lo que significa que es la matriz de flujo asociada a Λ_{AHD}. □

Proposición 6. *Si los flujos asociados con Ω y todo Λ_{ew} son enteros, entonces cualquier ruta en Ω tiene un valor entero en la solución.*

Proof. Usando las proposiciones 5 y 3, el flujo asociado a Λ_{AHD} es entero y toda ruta $r \in \Lambda_{\text{AHD}}$ tiene asociada un valor binario en la solución $\bar{\lambda}$. Usando la proposición 4, $\forall l \in L^{\text{SDL}}, \forall j \in W_{ext}(l)$, toda ruta $k \in \Lambda_{ew}$ tiene también asociada un valor binario en $\bar{\lambda}$. Como $\Lambda_{\text{AHD}} \cup_{l \in \text{SDL}} \cup_{j \in W_{ext}(l)} \Lambda_{ew} = \Omega$, toda ruta $r \in \Omega$ tiene asociada un valor binario en $\bar{\lambda}$. □

Corolario 1. *Si el flujo asociado a $\hat{\Omega} \subseteq \Omega$ es entero pero la solución asociada $\bar{\lambda}$ es fraccionaria, entonces tiene que haber una variable con valor fraccionario en la solución asociada a una ruta que visita al menos un $l \in L^{SDL}$.*

Proof. De no haber dicha variable, los flujos asociados con Ω y todo Λ_{ew} son enteros, entonces por la proposición 6, estamos frente a una solución entera. □

El corolario 1 nos dice que siempre que falle la regla de branching tradicional y estemos frente a una solución $\bar{\lambda}$ fraccionaria, hay una variable fraccionaria cuya ruta pertenece a alguno de los conjunto Λ_{ew}. Sea $\Lambda_{\hat{e}\hat{w}}$ dicho conjunto. Como hay una ruta $r \in \Lambda_{\hat{e}\hat{w}}$ que tiene asociada un valor $\bar{\lambda}_r$ fraccionario, y el flujo $\bar{e}_{\hat{e}\hat{w}}$ asociado a $\Lambda_{\hat{e}\hat{w}}$ es binario, entonces vale $\bar{e}_{\hat{e}\hat{w}} = 1$. También, hay al menos dos rutas $r \in \Lambda_{\hat{e}\hat{w}}$ asociadas a valores fraccionarios en $\bar{\lambda}$. Como son distintas, hay al menos dos arcos con valor fraccionario en la matriz de flujo asociada a Λ_{ew}.

Así concluimos que tomando cualquier ruta $r \in \Omega - \Lambda_{AHD}$ que tenga asociado un valor fraccionario en la solución, podemos obtenemos una desigualdad para hacer branching. Por último la proposición 6 nos dice que cuando no hayan reglas de esta forma para utilizar, la solución asociada $\bar{\lambda}$ es entera.

En el ejemplo de la figura 4.7, podemos utilizar tanto $\Lambda_{SDL_{in}(SDL_1)\ ((n_1,SDL_1),i_0)} = \{r_0, r_1\}$ como $\Lambda_{SDL_{in}(SDL_1)\ ((n_2,SDL_1),i_0)} = \{r_2, r_3\}$. En ambos casos, las matrices de flujo asociadas tienen los arcos $(i,j) \in \{(SDL_{out}(SDL_1), ((n_3, AHD_1), i_0)), (SDL_{out}(SDL_1), 0'), (((n_3, AHD_1), i_0), 0')\}$ con un flujo $\bar{e}_{ij} = 0.5$. Cabe destacar que si bien $\bar{e}_{SDL_{out}(SDL_1),0'}$ asociado a Ω puede ser mayor a 1, restringido a un subconjunto Λ_{ew} toma valores binario. Eligiendo el primer subconjunto $\Lambda_{SDL_{in}(SDL_1)\ ((n_1,SDL_1),i_0)} = \bar{\Lambda}$ y el arco $(i,j) \in A_{ext}$, las desigualdades elegidas para llevar a cabo la ramificación tienen la misma forma que antes, es decir $\sum_{r\in\bar{\Lambda}} \bar{\lambda}_r\ e^r_{ij} \leq 0$ y $1 \geq \sum_{r\in\bar{\Lambda}} \bar{\lambda}_r\ e^r_{ij}$.

4.2.4 Esquema de branching

Dada una solución fraccionaria $\bar{\lambda}$, en primer lugar se busca si existe un arco $(i,j) \in A_{ext}$ con flujo $\bar{e}_{ij}$ fraccionario asociado las rutas $\hat{\Omega} \subseteq \Omega$ del PMR para utilizar las reglas de la sección 4.2.1. Si no existe dicho arco, se aplica el método definido en la sección 4.2.3.

En el primer nivel, los arcos considerados son aquellos que inciden en alguna opción $w \in W_{ext}$. Esto son los arcos $(i,j) \in \{(i,j) \in A_{ext} | i \in W_{ext} \vee j \in W_{ext}\}$. El procedimiento realizado para elegir el arco es el siguiente: primero calculamos la matriz de flujo asociada a Ω, luego tomamos los 10 arcos incidentes a opciones con flujo $\bar{e}_{i,j}$ más cercano a 0.5, y por último elegimos el arco entre estos 10 cuya relajación lineal es más estrecha, es decir, que el mínimo entre la relajación lineal de ambas ramas sea máximo.

Como dijimos, este nivel agrega reglas de branching robustas, y se manejan manipulando el conjunto de aristas del grafo G_{ext} de la red subyacente, definiendo un nuevo grafo $\hat{G}_{ext} = (V_{ext}, \hat{A}_{ext})$. Para el caso $\bar{e}_{i,j} \leq 0$ simplemente quitamos el arco $(i,j) \in A_{ext}$, es decir $\hat{A}_{ext} = A_{ext} - \{(i,j)\}$. En el caso $\bar{e}_{i,j} \geq 1$, lo que queremos es que el arco sea utilizado en la solución. Si $i \in W_{ext}$, hay que forzar tanto que el pedido n_i se atienda a través de i como también que desde i solo se pueda ir a j. Esto significa que hay que quitar los arcos salientes desde todas las opciones del pedido n_i que no son i, es decir $I_1 = \{(s,t) \in A_{ext} | n_t = n_i \wedge t \neq i\}$, como también todos los arcos salientes desde i que no son a j, es decir $I_2 = \{(i,t) \in A_{ext} | t \neq j\}$. Si $i \notin W_{ext}$, entonces definimos $I_1 = I_2 = \emptyset$. De manera análoga, si $j \in W_{ext}$, hay que quitar $J_1 = \{(s,t) | (s,t) \in A_{ext}, n_t = n_j \wedge t \neq j\}$ y $J_2 = \{(s,j) \in A_{ext} | s \neq i\}$, y se definen como $J_1 = J_2 = \emptyset$ si $j \notin W_{ext}$. Entonces $\hat{A}_{ext} = A_{ext} - I_1 - I_2 - J_1 - J_2$. Por último, las rutas $r \in \hat{\Omega}$ que atraviesan algún arco eliminado, no deben ser consideradas en el PMR. Esto lo hacemos agregando las restricciones $\lambda_r = 0$.

En el caso del segundo nivel, tomamos una ruta r que pase por un punto SDL y tenga un valor asociado $\bar{\lambda}_r$ fraccionario. Tomamos el primer arco $(e,w) \in r$ con $e \in SDL_{in}$ y calculamos la matriz de flujo asociada a Λ_{ew}. Elegimos un arco entre los posibles utilizando el procedimiento del primer nivel. Luego hacemos branching sobre el flujo del arco elegido considerando solo el conjunto de rutas Λ_{ew}. Este nivel impacta en la estructura del label agregando recursos dinámicos que deben ser considerados en el problema de pricing.

Hay una cantidad $\mathcal{O}(|L^{SDL}| \times |N| \times T_{\max})$ de conjuntos Λ_{ew} a considerar en el peor caso, donde $T_{\max} = \max_{l\in SDL} |T(l)|$. Cada uno de estos puede decidir sobre el flujo de los arcos que inciden en al menos una opción de entrega, el mismo conjunto de arcos que considera el primer nivel, este es $\{(w,v) \in W_{ext} \times V_{ext}\} \cup \{(v,w) \in V_{ext} \times W_{ext}\}$, y en el peor caso son $\mathcal{O}(|W_{ext}| \times |V_{ext}| + (|V_{ext}| - |W_{ext}|) \times |W_{ext}|) = \mathcal{O}(|W_{ext}| \times |V_{ext}| + (|L^{SDL}|) \times |W_{ext}|) = \mathcal{O}(|W_{ext}| \times (|V_{ext}| + |L^{SDL}|)) = \mathcal{O}(|W_{ext}| \times |V_{ext}|)$.

5. EXPERIMENTACIÓN

En este capítulo realizamos un estudio experimental sobre el VRPDOTW-DC a fin de evaluar la eficiencia del algoritmo propuesto y, adicionalmente, analizar el impacto de considerar las capacidades variables de los puntos SDL en un potencial contexto práctico.

El capítulo está organizado de la siguiente forma. Primero, explicamos características del grupo de instancias utilizadas en los experimentos. Segundo, analizamos algunas variaciones que se pueden aplicar sobre la heurística *Limited Discrepancy Search* (LDS) para definir la configuración de la heurística experimentalmente. Tercero, mostramos el aporte de LDS a la solución, eligiendo así la configuración final del algoritmo de labeling que mejores resultados obtenga. Por último, utilizamos la mejor configuración para analizar como impacta la cantidad de intervalos de capacidad en los tiempos del algoritmo, así como también en la calidad de la solución y el beneficio de esto.

5.1 Benchmark

Como benchmark utilizamos dos conjuntos distintos de datos. Uno es el conjunto de instancias propuestas en [1] que es de acceso público. En este las capacidades de los SDL son diarias, es decir, se mantienen constantes durante todo el horizonte de planificación. Nos basamos en este conjunto para desarrollar nuestra solución, hacer experimentos preliminares y validar la solución final. Para esto último, realizamos modificaciones sobre nuestro modelo, agregando las restricciones relacionadas al nivel de servicio y el límite de vehículos.

Hasta donde sabemos no hay conjuntos de instancias publicadas donde los SDL tengan capacidades variables, por lo que el segundo conjunto de instancias es generado de manera ad-hoc para VRPDOTW-DC, siguiendo los lineamientos propuestos en [1] y [9].

Resumidamente, se considera un cuadrado de 50x50 como mapa, en donde distribuimos el depósito y los AHD de manera aleatoria uniforme. A continuación añadimos al mapa de la misma manera una cantidad de $\frac{|N|}{5}$ puntos SDL. El tiempo y costo de viaje de cada tramo es igual a la distancia euclidiana entre el punto de partida y destino, multiplicado por 10 y transformando a valores enteros tomando parte entera superior. Los tiempos de servicio (s_l) son de 2 y 5 minutos para las opciones ubicadas en SDL y en AHD, respectivamente. De la misma forma los tiempos de estacionamiento (p_l) son de 4 y 6 minutos. Ambos no se consideran para los costos, pero si en los tiempos de viaje. La capacidad del vehículo (Q) es de 150 y las demandas (q_n) son elegidas de manera aleatoria uniforme en el cerrado $[10, 20]$. Suponemos que hay una cantidad ilimitada de vehículos disponibles en el depósito, y como no pretendemos reducir la cantidad utilizada, el costo de uso de estos es de 0. La ventana de tiempo tanto para el depósito inicial como final es todo el horizonte de planificación, que es de 720 minutos. una instancia puede ser de dos tipos, *Chica* o *Mediana*. En el primer tipo, las ventanas de tiempo son de 60 y 120 minutos para los puntos AHD y SDL respectivamente, mientras que el segundo, son de 240 y 480. Los extremos de las ventanas de tiempo, los tiempos de servicio y de estacionamiento deben sufrir las mismas modificaciones que los tiempos de viaje para corresponderse.

Aplicamos algunas modificaciones a la metodología común. Para mejorar la utilidad de los SDL, elegimos sus posiciones tomando los centroides resultantes de correr *k-means*

sobre las posiciones de los AHD, con $k = |SDL|$. También, para garantizar la factibilidad de las instancias, todos los pedidos tienen una opción de tipo AHD por fuera de la cantidad de opciones de tipo SDL. Las opciones alternativas para cada pedido las seleccionamos tomando los k SDL más cercanos al AHD del pedido en cuestión, utilizando la distancia como criterio. Para evitar que hayan opciones inalcanzables desde el depósito inicial, debido a la disposición de las ventanas de tiempo y los tiempos de viaje, elegimos el inicio de la ventana de tiempo de los puntos $l \in L^{AHD} \cup L^{SDL}$, de manera aleatoria uniforme utilizando el primer rango no vacío entre:

1. $[a_0 + \bar{t}, b_0 - \bar{t} - |b_l - a_l|]$
2. $[a_0 + \bar{t}, b_0 - |b_l - a_l|]$
3. $[a_0, b_0 - |b_l - a_l|]$

donde $\bar{t} = \max_{(i,j)\in A} t_{ij}$ es el máximo tiempo de viaje entre todos los tramos de la instancia.

Para generar la capacidad variable, discretizamos el horizonte de planificación utilizando el minuto como la menor unidad de tiempo considerable. De esta manera, los retiros de pedidos en un punto $l \in L^{SDL}$ pueden ocurrir en cualquier minuto durante la ventana de tiempo $[a_l, b_l]$. Notar que fuera de esta no es posible, ya que l no está operativo. La capacidad de recepción con la que l inicia el horizonte de planificación se elige de manera aleatoria uniforme en $[1, 5]$, y luego se elige una secuencia de retiros de pedidos también de manera aleatoria. Por ejemplo, si l fuese el punto SDL descrito en la figura 3.1, y el eje de coordenadas fuesen minutos, la capacidad inicial y la secuencia de retiro de pedidos elegidas son respectivamente 1 y $[10, 14]$, ya que que los retiros ocurren en los minutos 10 y 14. Todas las secuencias que describen de 0 a $retiros_{\max}$ retiros de pedidos son consideradas de manera equiprobable. Para un l, el valor $retiros_{\max}$ es la cantidad de pedidos que pueden ser entregados en él, es decir $|W(l)|$. No consideramos secuencias con más retiros porque la capacidad de recepción excedente no es utilizada en la solución.

Si bien este procedimiento es genérico, para los siguientes experimentos utilizamos instancias generadas con las siguientes características:

- Una sola opción SDL por fuera de la opción AHD.
- Instancias de 25 y 50 pedidos.
- Instancias con ambos tipos de tamaño de ventanas de tiempo: chicas y medianas.
- 42 como semilla.

generando así 10 instancias por cada combinación de cantidad de pedidos y tipos de tamaño de ventanas de tiempo, denotando cada conjunto con una letra para el tipo, `C` o `M`, y un numero para los pedidos, 25 o 50.

Tenemos a disposición las instancias de [9], las cuales tienen también capacidades diarias. En estas instancias, dejar 1 pedido o todos en un SDL toma el mismo tiempo total de servicio en el punto. Esto significa en la práctica que el tiempo dentro de un SDL es constante independientemente de la cantidad de pedidos que se entreguen. Esta consideración hace que las instancias sean muy difíciles de resolver por nuestros algoritmos de labeling, y remarca el buen trabajo de las heurísticas planteadas en [9]. Debido a esta razón, no las utilizamos.

5.2 Configuración de Limited Discrepancy Search

Como vimos en la subsección 4.1.7, el algoritmo LDS tiene dos parámetros: GN y DISC. Cuando estos parámetros toman valores chicos, por ejemplo 2 y 1 respectivamente, menor es la cantidad de labels que se generan, y con esto menor el espacio de rutas que se explora y el tiempo de cómputo del algoritmo. Este espacio, en conjunto con el tiempo, crece a medida que incrementamos los parámetros hasta que pasa a ser exactamente el mismo espacio considerado por un labeling mono-direccional estándar, que ocurre cuando alguno de sus parámetros llega a su valor maximal.

Llamamos configuración de LDS a una ejecución de este algoritmo con dos parámetros determinados. Generalmente LDS se ejecuta con distintas configuraciones de manera iterativa, utilizando una secuencia de 1 o más configuraciones de LDS distintas. La i-ésima configuración se ejecuta solo cuando las primeras $i-1$ configuraciones fallan en encontrar rutas con costo costo reducido negativo. Esta secuencia generalmente se la ordena por dificultad y tiempos de cómputo de manera creciente. De esta manera, en una iteración de la generación de columnas se ejecutan potencialmente más de una configuración de LDS. El largo de esta secuencia y los parámetros utilizados en cada configuración de esta, es lo que llamamos meta configuración de LDS, y que nos interesa decidir de manera empírica a través de los siguientes experimentos.

5.2.1 Información compartida entre configuraciones

El espacio de rutas y labels explorado puede intersecarse entre dos configuraciones de LDS. Si ambos parámetros de una configuración son mayores o iguales que los de otro, entonces el primero contiene todos los labels factibles que explora el segundo. Como solo pasamos a una configuración con mayores parámetros cuando las anteriores en la secuencia, con parámetros más chicos, fallaron en encontrar al menos una ruta con costo reducido negativo, esto no solo significa considerar varias veces el mismo conjunto de labels, sino que también un conjunto del que sabemos no obtendremos ninguna ruta que nos interesa.

Cuando las configuraciones en la secuencia varían GN, hay que recomputar la cantidad de arcos malos recorridos para cada camino parcial desde el depósito inicial, lo cual puede ser caro computacionalmente. Por otro lado, cuando varían DISC y mantienen GN, todos los caminos parciales explorados mantienen su cantidad de arcos malos. Más aún, los labels factibles que se exploran en la $i+1$-ésima configuración y que no fueron previamente considerados, son extensiones de los descartados en la i-ésima configuración por haber sobrepasado su valor de DISC. Si en lugar de descartarlos, los guardamos, la siguiente configuración puede comenzar directamente desde estos y evitar generar todos los labels factibles desde el depósito inicial.

Los labels sin procesar los mantenemos en una cola de prioridad ordenada primero por tiempo. De esta manera sabemos al remover un label, que todos los labels dominantes hasta el momento tiene un tiempo menor o igual, y con esto podemos evitar comparar la condición 4.17. Cuando mantenemos GN y solo incrementamos DISC, el conjunto de labels dominantes contiene a los anteriores. Sin embargo no podemos compartir este conjunto directamente entre configuraciones, dado que pueden haber labels dominantes con tiempo mayor o igual a los labels que fueron descartados por sobrepasar DISC.

Proponemos entonces 3 implementaciones alternativas para compartir información entre las configuraciones LDS de una secuencia, cuando fijamos GN y solo incrementamos DISC:

- *Comienzo Frío (CF)*: una primera opción e implementada en [7], es no compartir nada de información entre configuraciones. Cada ejecución de LDS inicializa la cola de labels extendidos (sin procesar) con el label inicial (0), que parte del depósito en el tiempo inicial de su ventana. El conjunto de labels dominantes empieza vacío siempre.

- *No Compartir Labels Procesados (NC)*: otra opción es evitar generar todos labels factibles de la configuración anterior. Para esto mantenemos dos colas de prioridad en cada configuración, una donde todos los labels satisfacen $ArcosMalos \leq DISC$ y otra con los labels que sobrepasan el valor de $DISC$. Los labels en ambas colas son factibles respecto a otras características como carga, tiempo, etc. La primera configuración comienza con una primera cola con el label inicial (0) y una segunda cola vacía. Si la i-ésima configuración explora todos los labels de su primera cola sin encontrar una ruta con costo reducido negativo, se ejecuta la $i+1$-ésima configuración utilizando la segunda cola de la i-ésima configuración como su primera, y una cola vacía como su segunda. Como la $i+1$-ésima configuración tiene un valor de $DISC$ mayor al de la i-ésima, y el incremento de $ArcosMalos$ en una extensión es menor o igual que 1, los labels en la primera cola de la $i+1$-ésima configuración satisfacen $ArcosMalos \leq DISC$. Si encontramos una ruta con costo reducido negativo, entonces la configuración actual de LDS no falla y no se precisa ejecutar las siguientes configuraciones en la secuencia para esta iteración de la generación de columnas. De esta manera no tiene sentido seguir acumulando labels en la segunda cola, por lo que dejamos de hacerlo cuando se encuentra al menos una ruta con esta característica.

- *Comienzo Caliente (CC)*: igual que la opción anterior, pero compartiendo el conjunto de labels procesados. Para esto persistimos los labels procesados en dos estructuras, una es la descrita en la sección 4.1.5 que utilizamos para verificar si un label es dominado previo a procesarlo. La segunda mantiene todos los labels procesados encontrados hasta el momento entre todas las configuraciones LDS de la secuencia que fueron ejecutadas, ordenados por tiempo. Cuando tomamos el siguiente label m a procesar, previo a verificar si es dominado, actualizamos la primera estructura con todos los labels de la segunda que pueden llegar a dominar a m. Estos son todos los labels m' con

$$t(m') \leq t(m) \tag{5.1}$$
$$Loc(m') = Loc(m) \tag{5.2}$$

  Cada vez que un nuevo label es procesado, lo agregamos en ambas.

La elección de alguna de estas 3 opciones se añade a la meta configuración de LDS como su configuración de inicio. Para comparar estas 3 configuraciones utilizamos una misma configuración base del algoritmo de labeling cambiando solamente este aspecto de LDS. Esta consta de ejecutar una misma meta configuración de LDS seguida de un labeling bidireccional asimétrico exacto. Cada configuración en la secuencia es ejecutada si las previas fallaron en encontrar al menos una ruta con costo reducido negativo. Del mismo modo el exacto cuando todas las heurísticas fallaron. El largo de la secuencia es dinámico pero fijo para una misma instancia. La secuencia comienza con una configuración cuyo valor del parámetro DISC es 1, y le siguen más configuraciones incrementando en 1

este valor hasta llegar a $\frac{\text{DISC}_{\max}}{4}$. Decimos que es dinámico dado que $\text{DISC}_{\max}$, estimado a través del modelo (4.29)-(4.31), depende de cada instancia. Todas las configuraciones tienen fijo el parámetro GN en 2, elegido en base a experimentos preliminares. Tanto las configuraciones LDS como el exacto tienen una cota superior r_{max} sobre la cantidad de rutas que pueden agregar al modelo, eligiendo en cada caso las r_{max} rutas con costo mínimo y menor que cero entre las encontradas. En base a experimentos preliminares se eligió la siguiente política:

- $r_{max} = 1$ para la primera configuración de LDS, es decir, reporta una ruta factible con costo mínimo y menor que cero entre las que encuentre.
- $r_{max} = 8000$ para el labeling exacto.
- Cada configuración en la secuencia reporta como máximo la cantidad r_{max} de la configuración anterior más $\frac{8000}{|LDS|+5}$, donde $|LDS|$ es la cantidad de configuraciones LDS en la secuencia. Tanto ejecutar las configuraciones LDS con un valor r_{max} más grande, como el labeling exacto con un valor de r_{max} más chico, se desempeñó peor en experimentos preliminares. La constante sumada en el denominador tiene como objetivo separar el máximo valor de r_{max} de las configuraciones LDS y el valor de r_{max} del labeling exacto.

El espíritu de esta elección es prorratear el cómputo por ruta. Cuanto más arduo el trabajo y más tiempo se invierte, más rutas se reportan.

Las heurísticas de relajación se ignoran en este punto dado que solo queremos analizar el impacto en LDS. El labeling exacto se agrega porque las rutas elegidas pueden diferir, y con esto también la convergencia de la generación de columnas, llegando a precisar ejecutar el exacto en momentos distintos y una cantidad de veces distinta. Esto también debe ser analizado dado que en la práctica, ejecutar el exacto generalmente es más caro computacionalmente que ejecutar una heurística LDS con una configuración determinada.

Ejecutamos las 3 configuraciones sobre nuestro grupo de instancias generadas, resolviendo solamente la relajación lineal (RL) y con un límite de tiempo de 1800 segundos por instancia. Para cada instancia los óptimos de las relajaciones lineales son los mismos, por lo que debemos comparar otras características, como tiempos y cantidad de relajaciones lineales resueltas.

En la figura 5.1 tenemos un boxplot con los tiempos de ejecución sobre las instancias que fueron resueltas por las 3 alternativas. Lo primero que se puede ver es que la configuración `NC` obtiene los peores tiempos, necesitando en promedio más del doble de tiempo para resolver una instancia. Esto la convierte en la peor de las tres. También se puede ver que las 3 alternativas encuentran las instancias con ventanas de tiempo *Medianas* más difíciles de resolver que *Chicas* para una misma cantidad de pedidos, algo esperable que suceda.

En la figura 5.2 repetimos el mismo procedimiento quitando `NC` para poder apreciar mejor las diferencias entre `CC` y `CF`. En este caso se consideran las instancias resueltas por ambos, que son más que antes, y por esto cambia la escala de las cajas del gráfico. Ahora se puede ver que ambas logran resolver una instancia del tipo *Chicas 50*. También se puede ver que las instancias que se agregaron ahora de *Medianas 25* generaron que las distribuciones de tiempo muestren más dispersión. Si bien los tiempos promedios son similares, la alternativa `CC` es consistentemente mejor en tiempos en las tres categorías en cuanto al tiempo promedio. También muestra menor dispersión, lo que brinda más

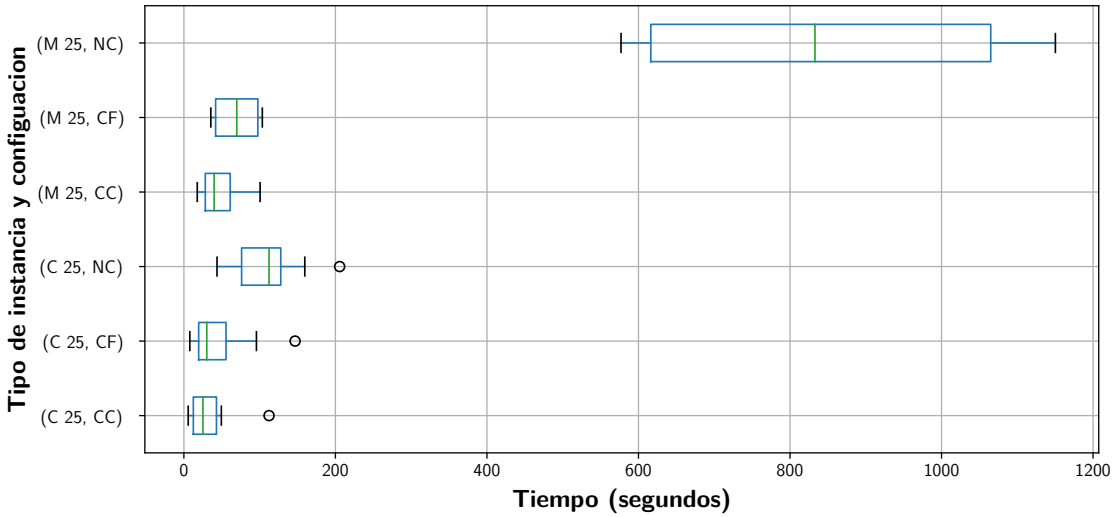

Fig. 5.1: Tiempos de ejecución por tipo de instancia chicas (C) y medianas (M) para configuraciones de inicio de LDS: `CC`, `CF` y `NC`. Considerando solo las instancias resueltas por las tres.

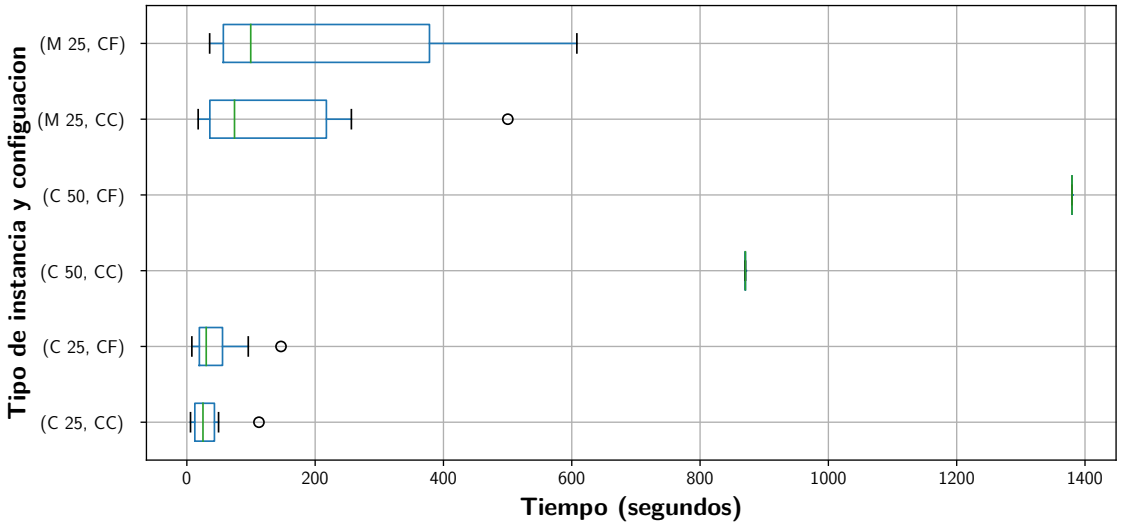

Fig. 5.2: Tiempos de ejecución por tipo de instancia para configuraciones de inicio de LDS: `CC` y `CF`. Considerando solo las instancias resueltas por ambas.

Tipo	Exp	Cant. RL Resueltas	Tiempo total	Tiempo heurística	Tiempo exacto
C 25	CC	10	33.29	12.74	20.29
C 25	CF	10	47.68	19.34	28.1
C 25	NC	10	110.2	81.76	28.08
C 50	CC	2	1321.32	1122.11	196.61
C 50	CF	1	1379.73	1326.22	51.43
C 50	NC	0	-	-	-
M 25	CC	6	159.14	38.95	119.89
M 25	CF	7	389.87	97.09	292.47
M 25	NC	4	848.29	812.78	34.95

Tab. 5.1: Resultados por tipo de instancia para configuraciones de inicio de LDS: `CC`, `CF` y `NC`. Valores promediados entre las instancias resueltas para cada combinación de tipo de instancias y configuración.

confianza sobre su elección. Al igual que antes, se puede ver que hay un tipo de instancias que les resulta más difícil a las dos alternativas, que en este caso *Chicas 50*.

La cantidad de instancias resueltas por cada configuración está en la tabla 5.1. Ahí podemos encontrar agrupadas por tipo de instancia, la información de cada alternativa sobre las instancias que cada una resuelve. Es importante destacar que ninguna configuración fue capaz de resolver una instancia del tipo *Medianas 50*.

`NC` es la que menos instancias resuelve. Es también la única configuración en exceder el límite de memoria RAM de 12 GB impuesto en todos los experimentos. A medida que las instancias son más grandes, se aprecia una diferencia mayor en el tamaño del conjunto de labels procesados en `NC` respecto a `CF` y `CC`. Si bien no lo reportamos en este informe, para las configuraciones LDS con DISC 4 y 5 en *Medianas 25*, esta diferencia llega a ser de 1 a 3 órdenes de magnitud, llegando a ser millones contra decenas de miles. Una de las diferencias entre estos conjuntos es que `CF` (por recalcularlos) y `CC` (por compartirlos) incluyen labels procesados de iteraciones anteriores, y en estos encontramos labels correspondientes a caminos parciales cortos. Al mismo tiempo, los labels que pasan a considerarse en la configuración de LDS con DISC 4, son caminos que ya parten de una longitud mínima de 4 o 5, por lo que esperamos `NC` tenga labels procesados más largos. Un camino parcial corto puede dominar más que uno largo dada la condición (4.18) sobre el conjunto S. Una posibilidad es entonces que estos labels que no tiene `NC` sean responsables de la mayor cantidad de las dominaciones en `CF` y `CC`.

`CF` y `CC` resuelven la misma cantidad de instancias pero difieren en los grupos *Medianas 25* y *Chicas 50*. Dado que ambos exploran el conjunto de rutas de una forma parecida, sus convergencias también sos parecidas, y terminan llamando al labeling exacto igual veces en promedio. Incluso el gap de mejora en la relajación entre llamados al exacto es igual. Lo único que cambia, es que al ser `CC` más rápido que `CF`, hace el primer llamado al exacto antes en el tiempo. En ambos casos, la instancia adicional hace que el tiempo promedio en el tipo de instancia respectivo, aumenten. Sin embargo, esto no genera que la situación se revierta en el grupo *Chicas 50*. Cabe destacar que si bien `CC` no logra resolver la instancia 2 de *Medianas 25*, si encuentra una solución factible con el valor óptimo de la relajación lineal y el límite de tiempo lo encuentra corriendo nuevamente del labeling exacto, por lo que esperamos que agrandando el límite de tiempo muy poco, esta instancia sea resuelta

también por *CC*. Por esto último y porque consideramos que el grupo *Chicas 50* es más difícil de resolver, concluimos que *CC* es la mejor elección.

5.2.2 Eligiendo una secuencia de configuraciones

En [7], la meta configuración de LDS es igual a la utilizada por `CF` en el experimento anterior. La diferencia es que en lugar de ejecutar un labeling bidireccional asimétrico, corren una última configuración LDS con $\text{DISC}_{\max}$, que como dijimos en la sección 4.1.7 es equivalente a ejecutar un labeling mono-direccional estándar. Dado que `CC` se desempeña mejor en estas instancias, la utilizamos en lugar de `CF`, y la llamamos en este caso *incremento constante* o `CONS`.

Con el objetivo de elegir una meta configuración de LDS, comparamos esta con la propuesta en [1], que consta de fijar DISC e incrementar GN. Específicamente usamos la secuencia de valores $(1, 2, 4, 8, 14)$ para GN y un valor de DISC de 2. Experimentos preliminares arrojaron que la misma configuración, cambiando el valor DISC a 1, obtiene mejores resultados para nuestras instancias, por lo que utilizamos este valor. A esta configuración, la llamamos `TILK`. Como explicamos en el experimento anterior, al incrementar GN nos vemos forzados a utilizar la alternativa `CF` con cada configuración de LDS de la secuencia. Por lo que esta configuración no se ve favorecida por el resultado del experimento anterior.

Tomamos al igual que antes, las instancias resueltas por ambas configuraciones y mostramos la distribución de tiempos con un boxplot. El resultado se puede ver en la figura 5.3. Si bien en las instancias *Chicas 25* y *Medianas 25* `CONS` obtiene mejores tiempos, en las instancias más difíciles de resolver, *Chicas 50*, el resultado es el opuesto.

El experimento anterior muestra que `CC` es consistentemente mejor que `CF`. Para descartar que la razón sea el hecho de tener una secuencia de configuraciones LDS más larga, se hizo un experimento preliminar con secuencias de configuraciones cuyos valores del parámetro DISC crecían exponencialmente. Estos mostraron que el beneficio de crecer incrementando de a 1 es mayor al overhead de una cadena larga de configuraciones. Es más, pocas veces `CONS` requiere resolver configuraciones LDS con valores de DISC mayores a 6, por lo que saltear valores de DISC intermedios, resolviendo por ejemplo configuraciones con DISC 7 u 8 cuando DISC 5 encuentra rutas, es un gasto de tiempo innecesario.

Si bien no es reportado explícitamente, analizando la cantidad de labels extendidos podemos notar que el conjunto de labels factibles crece de manera escalonada e inmediata con el aumento de DISC. En cuanto a `TILK`, vemos un corrimiento en el crecimiento exponencial de los labels. A partir de esto, tomamos las dos instancias que son resueltas por `TILK` y `CONS` del grupo *Chicas 50*, agrupamos cada una de las configuraciones LDS ejecutadas por igual parámetros GN y DISC, las promediamos y segmentamos en base a las fases del labeling (extensión, encolamiento, dominación, etc.), y graficamos en la figura 5.4. Como era de esperar, el crecimiento rápido del conjunto de labels factibles al aumentar DISC se traduce en un crecimiento rápido del tiempo. También, como pocas veces `TILK` necesita resolver configuraciones con GN mayor a 4, el tamaño del conjunto de labels le juega a favor.

`TILK` no solo extiende muchos menos labels que `CONS`, sino que con esa cantidad logra encontrar rutas con costo reducido negativo mientras `CONS` falla en hacerlo. Si bien exploran un espacio de labels factibles distintos, no logramos explicar cual es el beneficio que goza `TILK` o la razón de estos resultados. Seguramente sea necesario plantear experimentos adicionales para mejor la comprensión del comportamiento de cada configuración de secuencias, que quedará como trabajo a futuro.

Tipo	Exp	Resueltos	Tiempo total	Tiempo heurística	Tiempo exacto
C 25	TILK	10	65.4	43.51	21.49
C 25	CONS	10	33.29	12.74	20.29
M 25	TILK	6	380.79	273.36	106.94
M 25	CONS	6	159.14	38.95	119.89
C 50	TILK	2	1175.98	694.54	478.7
C 50	CONS	2	1321.32	1122.11	196.61

Tab. 5.2: Resultados por tipo de instancia para las configuraciones de secuencia de LDS `TILK` y `CONS`, promediando valores sobre las instancias resueltas en cada caso.

Los resultados agrupados por tipo de instancia se encuentran en la tabla 5.2. Ambas configuraciones resuelven las mismas instancias. Concluimos que `TILK` es la mejor meta configuración de LDS considerada hasta ahora en este trabajo.

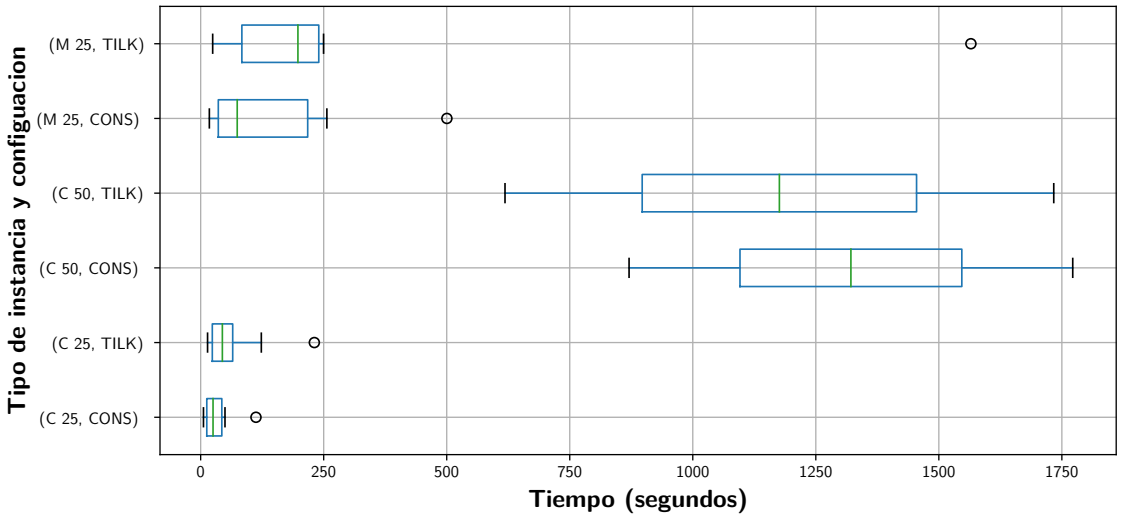

Fig. 5.3: Tiempos de ejecución por tipo de instancia para configuraciones de secuencia de LDS: `CONS` y `TILK`. Considerando solo instancias resueltas por ambas.

5.3 La ventaja de utilizar LDS

Nuestras configuraciones hasta el momento constan de la ejecución de una secuencia de heurísticas LDS seguido de un labeling exacto. Hay una observación de los resultados anteriores que es consistente con lo que mencionan en [1]: la mayor parte del tiempo del labeling se va en la dominación. Es por esto que cualquier mejora en la dominación tiene un gran impacto, y las heurísticas que relajan estas condiciones consiguen tiempos muy bajos. En este experimento vamos a agregar a cada configuración las dos heurísticas de relajación detalladas en la sección 4.1.6. Estas no solo son rápidas sino que inspeccionan un conjunto de rutas distinto al de LDS, ya que no se guían por la discrepancia.

Si bien le dedicamos una gran porción de este informe a LDS y cómo buscar una buena configuración de esta, no hemos medido hasta ahora cual es el impacto en la solución al

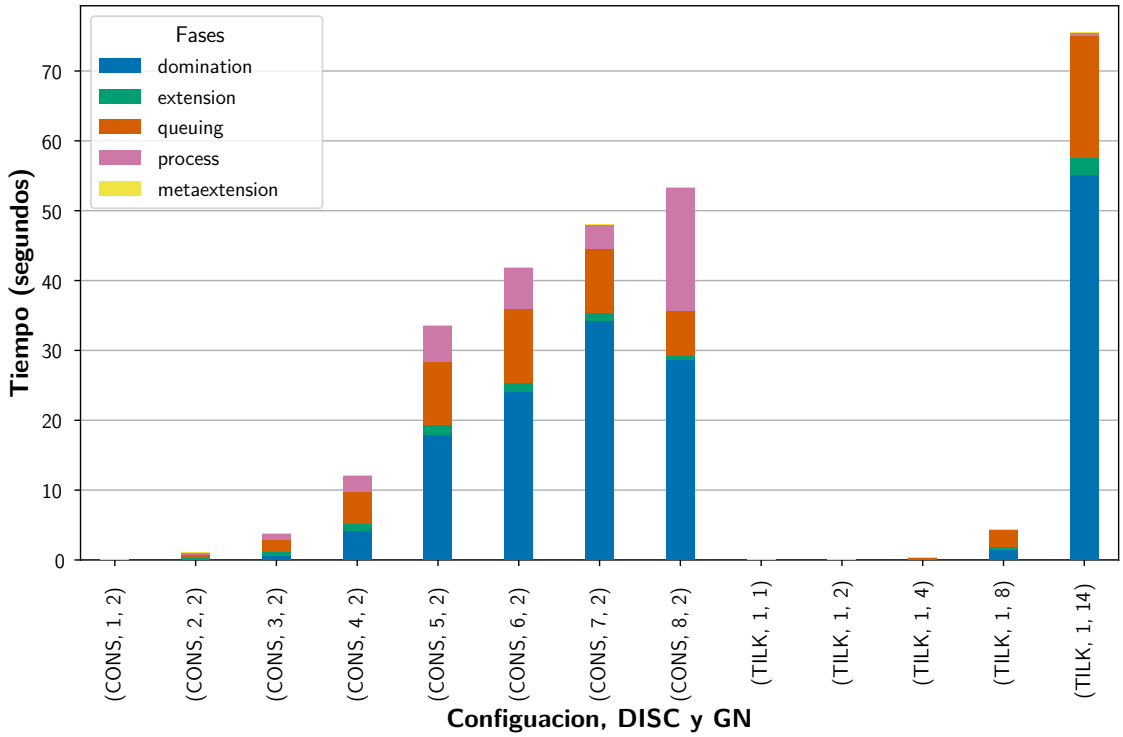

Fig. 5.4: Tiempos de ejecución promedio agrupado por igual configuración de parámetros de LDS (DISC, GN), para las instancias resueltas de *Chicas 50*, segmentado por fase de labeling, para configuraciones de secuencia de LDS: `CONS` y `TILK`.

agregar estas heurísticas. Más aún, si basta con utilizar heurísticas que relajen condiciones de dominación. Partiendo de la meta configuración ganadora del experimento anterior, agregamos las dos heurísticas de relajación previo a ejecutar la secuencia, y llamamos a esto `REL+TILK`. El orden se debe a que son las más rápidas. Por otro lado, tenemos otra meta configuración sin las heurísticas LDS, que llamamos `NO_LDS`, que usa solamente las heurísticas de relajación.

En ambas meta configuraciones las heurísticas de relajación se agregan con un límite de 1 ruta como máximo para reportar.

Si nos restringimos a las instancias resueltas por ambas configuraciones como hemos hecho antes, podemos ver que, contrario a lo que esperábamos, `NO_LDS` obtiene mejores tiempos en promedio que `REL+TILK`, en 2 de los 3 grupos de instancias, y más importante, en el más difícil. En la figura 5.5 se puede ver incluso que la relación de tiempos en las instancias más difíciles es la misma. Esto tiene que ver con que la mayoría de las iteraciones de la generación de columnas en ambos casos se resuelve con las heurísticas de relajación, resultando ser mucho mejor de lo que esperábamos.

Sin embargo, cuando analizamos los datos por grupo de instancia en la tabla 5.3, podemos ver que `REL+TILK` logra resolver dos nuevas instancias del grupo más difícil, resolviendo así más instancias que `NO_LDS`. Incluimos más datos esta vez en la tabla para justificar esta diferencia. Se puede ver que el tiempo que `NO_LDS` no utiliza ejecutando heurísticas, se contrarresta con un mayor tiempo de ejecución del exacto. Una razón obvia de esto tiene que ver con que al fallar las 2 heurísticas no tiene otra opción que llamar al exacto, mientras `REL+TILK` resuelve más iteraciones sin ejecutar el exacto, utilizando LDS. La otra razón, no tan obvia, es que al llamar tan pronto al exacto, la diferencia con el óptimo de la relajación lineal es más grande que en `REL+TILK`. Las sucesivas ráfagas de iteraciones de la generación de columnas resueltas con las heurísticas de relajación terminan más rápido y mejoran más la solución en comparación con las mejoras que brinda `REL+TILK`. Sin embargo, estas mejoras no llegan a ser suficiente como para contrarrestar la lejanía del comienzo, y como resultado se necesita ejecutar más veces el labeling exacto.

Concluimos que es mejor utilizar ambos grupos de heurísticas, y utilizamos `REL+TILK` para los siguientes experimentos.

5.4 Ventanas dinámicas y la granularidad de la información

Podemos pensar a la ventana de tiempo de un SDL como un segmento en la escala real, y a los retiros de pedidos como puntos en el segmento. En la práctica hay muchos factores que inciden en la calidad y forma de este dato. Por ejemplo, en el caso de estar prediciéndolo, podría ser que la información histórica ya se encuentre agregada o tenga una unidad mínima de intervalo de tiempo con el que se miden los cambios de capacidad. Este intervalo también puede ser elegido en función de otras características, como la performance del modelo predictor o la cantidad de bytes necesarios para representar el dato.

Hasta el momento hemos dedicado nuestro esfuerzo en dar con una configuración de la solución que resuelva la relajación lineal de la mayor cantidad de instancias y lo más rápido posible. Ahora nos enfocaremos en analizar el comportamiento de la solución frente a la variación del input del problema, específicamente en cómo impacta la cantidad de intervalos de capacidad reportados en los tiempos de cómputo de la solución. Pero más importante aún, cúal es el beneficio que obtenemos al considerar las capacidades variables.

Las instancias que generamos y utilizamos en los experimentos anteriores, tienen retiros

Tipo	Exp	Resueltos	Tiempo total	Tiempo heurística	Tiempo exacto	Llamados al exacto	Tiempo al 1er llamado al exacto	gap en 1er llamado al exacto
C 25	REL+TILK	10	59.57	45.68	10.21	2.4	41.97	0.13
C 25	NO_LDS	10	63.96	5.08	58.06	8.5	4.61	13.041
M 25	REL+TILK	5	154.02	89.81	59.31	2.6	75.91	0.09
M 25	NO_LDS	5	265.79	7.29	256.34	7.4	6.98	10.852
C 50	REL+TILK	4	1369.42	872.8	451.31	3.0	743.28	0.105
C 50	NO_LDS	2	1104.91	120.72	973.55	8.0	114.86	4.71

Tab. 5.3: Resultados agregados por grupo de instancia comparando para las configuraciones del labeling `REL+TILK` y `CONS`. Los valores se promedian considerando las instancias resueltas por cada uno.

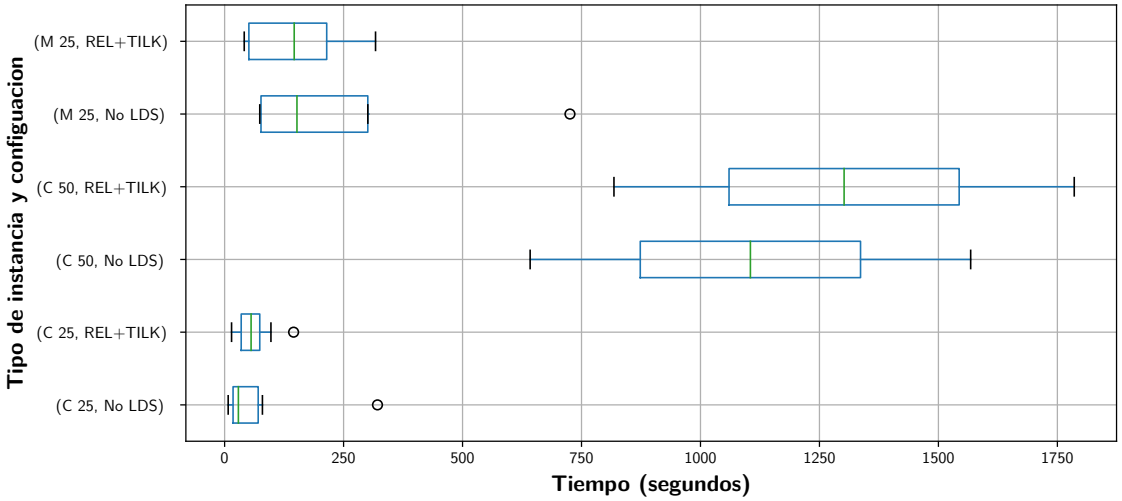

Fig. 5.5: Tiempos de ejecución por tipo de instancia para las configuraciones del labeling `REL+TILK` y `CONS`. Los valores se promedian considerando solo las instancias resueltas por ambos.

de pedidos en cualquier minuto. El dato resultante de la capacidad es igual a si pudiéramos medir en el futuro la capacidad del SDL de a intervalos de 1 minuto.

En la figura 5.6 podemos observar un ejemplo de la recta de retiro de pedidos y el dato de retiros obtenido utilizando intervalos de medición t, de 1 minuto. Como estamos modelando los retiros en la realidad con una discretrización de un minuto, siempre van a reflejar la realidad modelada. Este dato podemos posprocesarlo para generar instancias que se hubieran obtenido si el intervalo de medición fuese otro. En la figura podemos ver 3 ejemplos de esto, con un t de 5, 7 y 10 minutos.

Cabe aclarar que la solución óptima calculada a partir de datos posprocesado, tanto del problema como de la relajación lineal, pueden no ser óptimas respecto a la misma instancia con datos originales (sin posprocesar). Esto se debe a que al posprocesar los datos de capacidad de recepción, el espacio de rutas factibles puede reducirse. Por la misma razón, las soluciones obtenidas a partir de datos originales pueden no ser factibles cuando los datos están posprocesados. Por ejemplo, cualquier ruta que use $s + 4$ unidades de capacidad en la figura 5.6 es irrealizable cuando el intervalo de medición es de 5 minutos o más. Esto no pasa necesariamente entre dos instancias con datos posprocesados cualquiera. Por ejemplo, las rutas factibles con un intervalo de 5 minutos no incluyen a las rutas factibles usando un intervalo de 7. En estos casos no hay una relación directa entre las relajaciones lineales o las soluciones óptimas enteras, como si hay cuando se trata de intervalos de 1, 5 y 10 minutos, como $\Omega_{\text{intervalos},1} \subseteq \Omega_{\text{intervalos},5} \subseteq \Omega_{\text{intervalos},10}$, donde $\Omega_{\text{intervalos},t}$ es el conjunto de rutas factibles cuando el intervalo de medición es de t minutos.

Como la ventana mínima de un SDL para las instancias que generamos es de 240 minutos, y queremos comparar los óptimos de cada una, vamos a considerar los intervalos de $1, 15, 30, 60, 120$ y 240 minutos. El conjunto de instancias que utilizamos en los experimentos anteriores, utiliza intervalos de 1 minuto. Para los intervalos mayores a 1, se posprocesa el conjunto de instancias de 1, utilizando el intervalo que se desea. Para replicar estos experimentos, hay que tener en cuenta escalar estos valores al igual que se

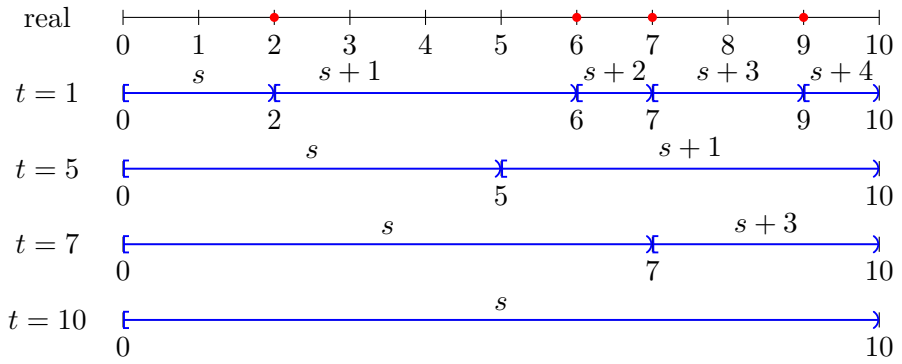

Fig. 5.6: Ejemplo de retiros de pedidos en un punto SDL durante el período $[0, 10)$. El SDL comienza con una capacidad de recepción de s. Los retiros ocurren en los minutos 2, 6, 7 y 9, marcados en rojo. El dato de entrada es representado con un intervalo de medición de un minuto, y luego posprocesado para intervalos de 5, 7 y 10 minutos.

	# Instancias resueltas						# Relajaciones resueltas					
Intervalos / Instancias	1	15	30	60	120	240	1	15	30	60	120	240
C 25	6	6	7	7	9	10	10	10	10	10	10	10
C 50	0	0	0	0	0	1	4	3	3	3	3	7
M 25	1	1	1	1	1	3	5	6	6	6	7	8
M 50	0	0	0	0	0	0	0	0	0	0	0	0

Tab. 5.4: Cantidad de instancias resueltas de manera óptima y solo la relajación lineal en nodo raíz por tipo de instancia, para los intervalo de medición de SDL $1, 15, 30, 60, 120$ y 240.

hace con los tiempos de viaje.

Ejecutamos la solución con la configuración ganadora del experimento anterior, `REL+TILK`, pero esta vez buscando el óptimo entero, utilizando el esquema de *branching* definido en la sección 4.2.4. El segundo nivel del esquema no se precisó para resolver a optimalidad ninguna de estas instancias ni las compartidas en [1], por lo que no lo implementamos. Cada instancia se ejecutó con tiempo límite de 30 minutos.

En los experimentos anteriores, nos encontramos con varias instancias a las que no pudimos resolverles su relajación lineal. Como era de esperarse, menos instancias aún son resolubles de manera óptima. Este primer resultado podemos verlo reflejado en la tabla 5.4. El grupo de instancias más fáciles, *Chicas 25*, es el que más cantidad de instancias resueltas de manera óptima tiene, vamos a analizar los aspectos de interés para estas.

La primera relajación lineal, también referida como relajación del nodo raíz, es la que resolvemos previo a cualquier decisión de *branching*. En la figura 5.7 se puede ver una reducción de este valor entre los intervalos de medición de 240 y 120 minutos, del 18% en promedio. Esto es pasando de capacidades diarias a reportar la capacidad por cada mitad de la ventana de tiempo, de haber algún retiro durante la primera mitad de la ventana. Como era de esperarse, el valor de la relajación se reduce a medida que achicamos el intervalo de medición de la capacidad, aunque el cambio relativo es en promedio más chico que en el primer caso:

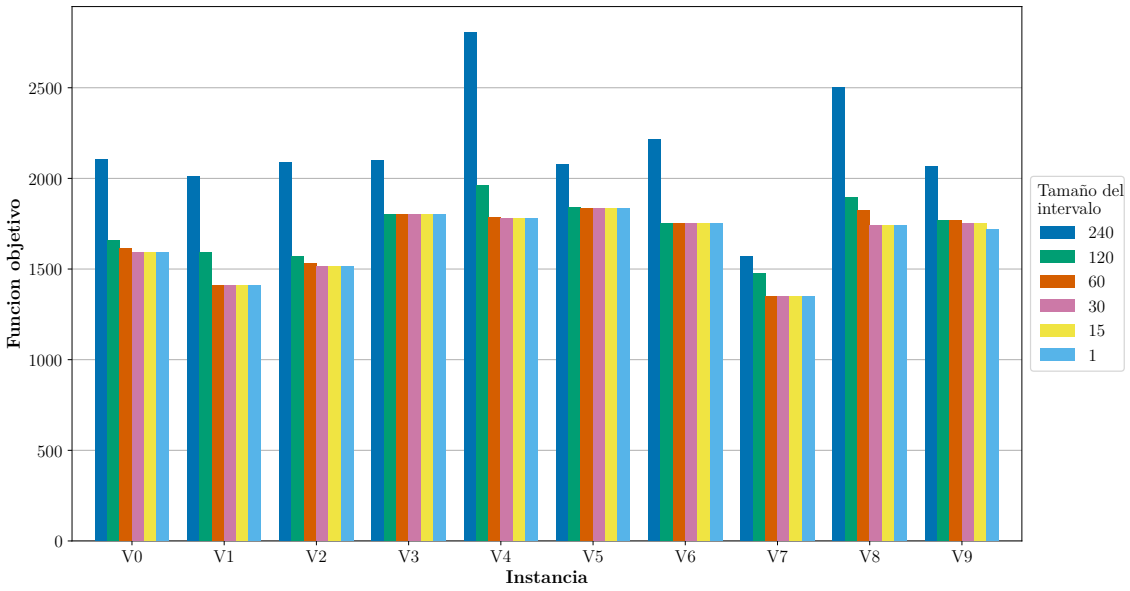

Fig. 5.7: Valor de la relajación lineal variando el intervalo de medición de la capacidad de los SDL, agrupando por una misma instancia original, para las 10 instancias del grupo *Chicas 25*.

- $120 - 60$: reducción del 3.8% en promedio.
- $60 - 30$: reducción del 0.8% en promedio.
- $30 - 15$: reducción nula en promedio.
- $15 - 1$: reducción del 0.19% en promedio.

De la misma forma podemos analizar la reducción del valor de la mejor solución entera a medida que achicamos el intervalo. No podemos hablar de la solución óptima, porque algunas instancias no se pudieron resolver de manera óptima en los 30 minutos. Haremos el análisis en base a la mejor solución encontrada dentro del tiempo de ejecución. En la figura 5.8 se puede ver el mismo comportamiento que observamos con las relajaciones lineales. Los resultados muestran que modelar las capacidades variables tiene sentido desde un punto de vista de negocio, ofreciendo una reducción del costo logístico que va del 5% al 37%, y es del 22% en promedio. Cabe destacar que resultó útil incluso en los escenarios en los que no se obtuvo la solución entera óptima.

Si bien por la dificultad de las instancias, este análisis no es cuantitativo, el porcentaje de reducción de costos resulta comparable y en algunos casos mejor que el obtenido por Dror y Trudeau [5] al sugerir que *split deliveries* podía impactar positivamente en los costos.

En cuanto a los tiempos de ejecución de la solución para resolver la relajación lineal, se puede ver en la figura 5.9 que todas las instancias *Chicas 25* se resuelven en menos de 200 segundos. Esto era esperable porque todas las configuraciones de los experimentos anteriores pudieron resolver la relajación lineal en las instancias de este grupo con intervalo de medición de 1 minuto, que esperábamos fueran las más difíciles de resolver.

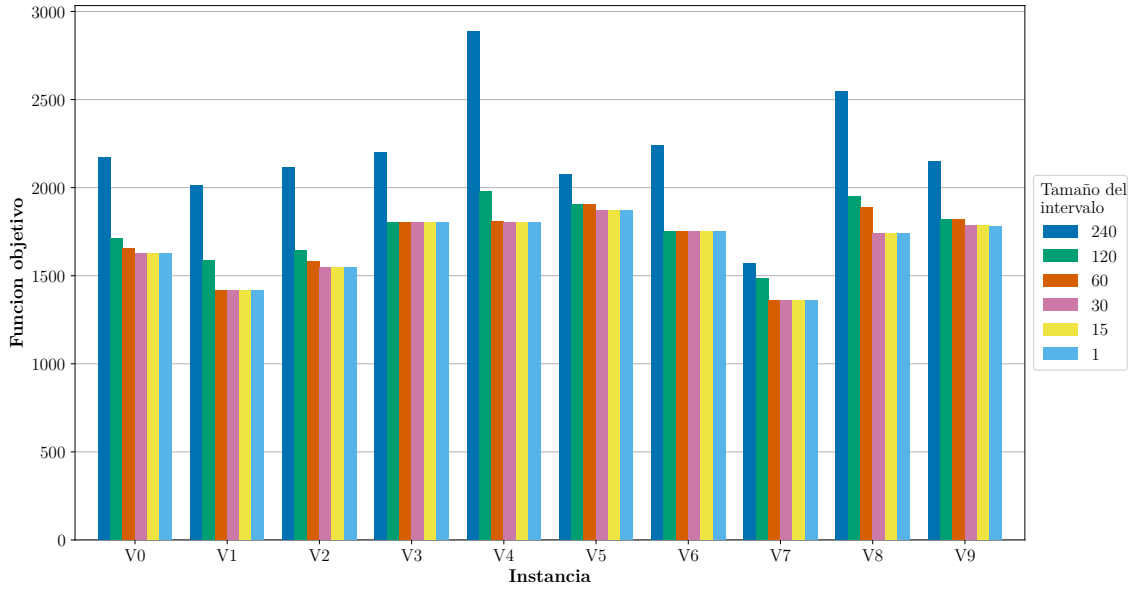

Fig. 5.8: Valor de la mejor solución entera variando el intervalo de medición de la capacidad de los SDL, agrupando por una misma instancia original, para las 10 instancias del grupo *Chicas 25*.

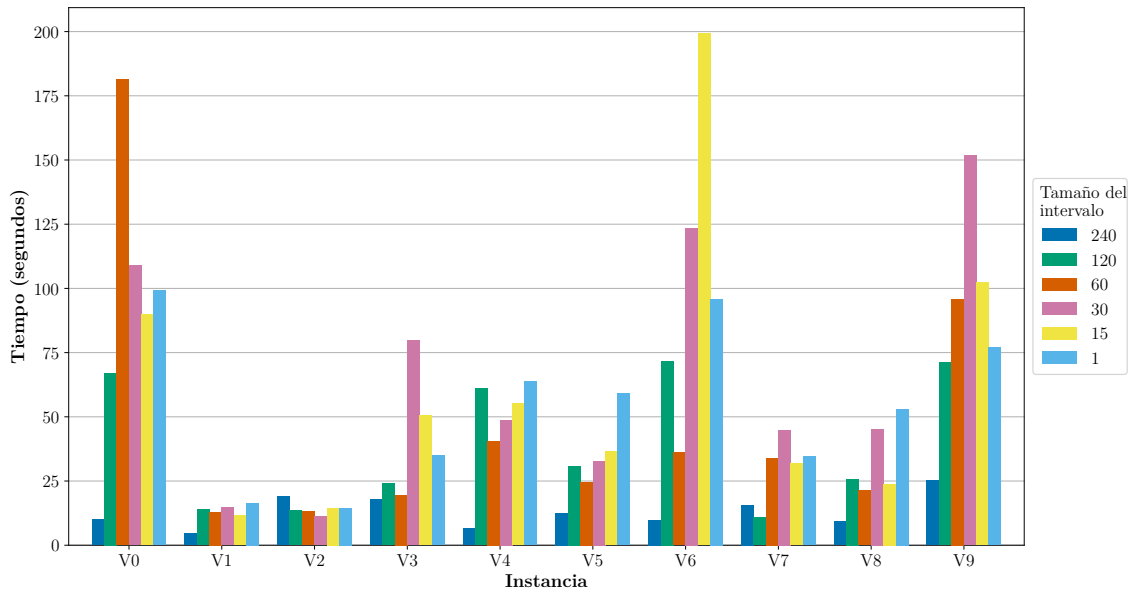

Fig. 5.9: Tiempo total de corrida de relajación lineal del nodo raíz variando el intervalo de medición de la capacidad de los SDL, agrupando por una misma instancia original, para las 10 instancias del grupo *Chicas 25*.

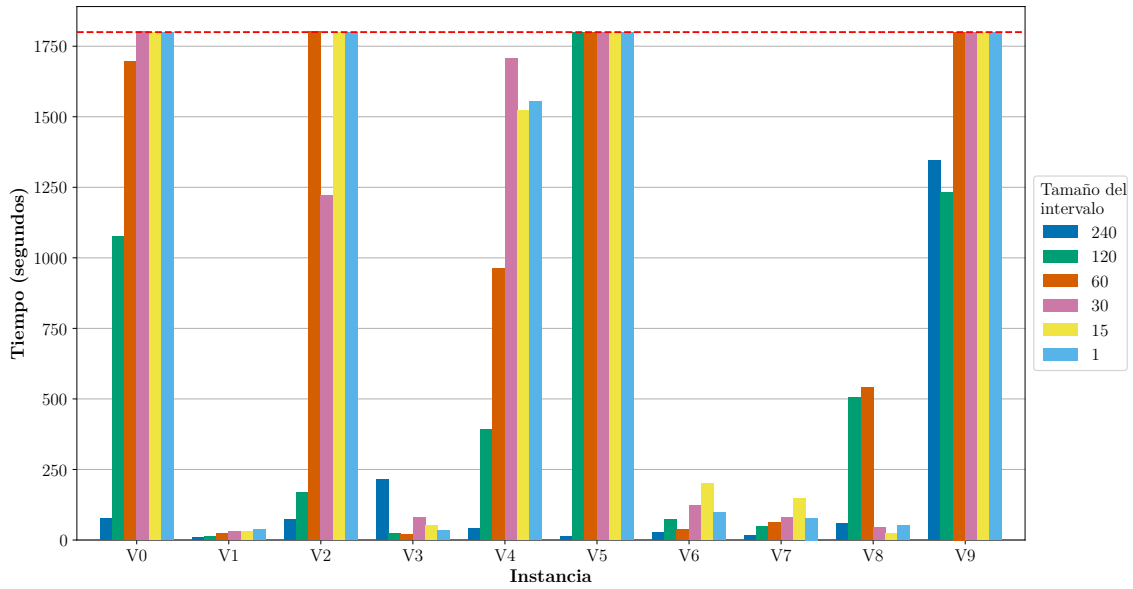

Fig. 5.10: Tiempo total de corrida de la solución variando el intervalo de medición de la capacidad de los SDL, agrupando por una misma instancia original, para las 10 instancias del grupo *Chicas 25*.

Se puede ver que en general los tiempos crecen a medida que achicamos el intervalo de medición. En el caso de que los tiempos bajen, esperábamos que esté relacionado al hecho de que uno o más lockers consiguen un valor de capacidad de recepción suficientemente grande como para poder recibir todos los pedidos asociados a estos, con lo que achicar más el intervalo de medición no agrega más capacidad que pueda ser utilizada. Así sucede en los caso de *V0* y *V3* en parte. Nos sorprendió encontrar que en el caso de *V6*, la diferencia es que una mayor discretización de los valores intermedios dentro de la ventana de tiempo, permite observar ventanas de mayor duración para cada valor, a pesar de no modificar la máxima recepción en la planificación. Interpretamos esta mejora en base a las restricciones de tiempo: no hay que esperar hasta los últimos k minutos de la ventana del SDL para atender los pedidos que faltaban, sino que más temprano hay más lugares disponibles y se combina más fácil con otros puntos, pudiendo armar rutas más largas más rápido.

En cuanto al tiempo para resolver las instancias de manera óptima, se ve en la figura 5.10 como claramente achicar el intervalo hace que crezcan los tiempos de ejecución del algoritmo, y rápidamente 4 de 10 instancias no se puedan resolver en el tiempo límite. En los casos de *V3*, *V6* y *V8*, a partir de cierto tamaño de intervalo se encuentra el óptimo en la raíz y dejan de necesitar hacer branching.

Algo que puede ser útil si tenemos un poder de decisión sobre la generación de los datos de entrada, o nos planteamos preprocesar esta para mejorar los tiempos de cómputo de la solución, es elegir un intervalo de medición con el que trabajar. La forma de elegir este depende de muchos factores, como:

- Cuánto tiempo estamos dispuestos a esperar para obtener una solución.
- Si estamos esperando una solución óptima o si hay un gap para considerar una

Tamaño del intervalo	% mejora respecto capacidad diaria	% mejora respecto capacidad diaria (solo resueltas por todos)	Cantidad de instancias resueltas (sobre 10)
120	20.05	20.21	9
60	24.33	24.39	7
30	25.60	25.36	7
15	25.36	25.36	6
1	25.36	25.36	6

Tab. 5.5: Porcentaje de mejora respecto a intervalos de 240 minutos solo considerando las instancias resueltas de manera óptima por cada uno, las instancias resueltas de manera óptima por todos, y porcentaje de instancias resueltas de manera óptima.

solución entera como suficientemente buena.

- Si estamos dispuestos a dejar de resolver cierto X% de instancias mientras la mejora sea de más de Y%.

La instancia *V5* solo fue resuelta de manera óptima con capacidades diarias. Por lo que con 30 minutos, si queremos maximizar cantidad de instancias resueltas primero, concluimos en mejor utilizar capacidades diarias. Sin embargo, con tamaños de intervalos de longitud menor a 240 se consiguen soluciones enteras con menor costo logístico en el tiempo límite, que presentan una mejora de al menos el 6% respecto a la obtenida utilizando capacidades diarias. Como podemos observar en la tabla 5.5, si nos permitimos no resolver de manera óptima entre un $10\% - 29\%$ de las instancias, podemos mejorar un 20.05% utilizando intervalos de 120 minutos. Si nos permitimos resolver un 30% menos de instancias, esta mejora sube a su máximo de 25.6%.

En la tabla, como era de esperarse, se puede ver que si nos limitamos a considerar las instancias que fueron resueltas de manera óptima con todos los tamaños intervalos, la mejora se incrementa a medida que achicamos el tamaño. Esto se corresponde con los resultados obtenidos en la figura 5.8. Contrario a esto, si consideramos para cada tamaño de intervalo el total de instancias resueltas de manera óptima, utilizando un intervalo de 30 minutos, obtenemos una mejora un poco más alta que con 15 y 1 minuto. Como vemos en la figura 5.10, utilizando un intervalo de 30 minutos se pudo resolver una instancia más que con las otras dos, la `V2`, y esta obtiene una reducción sustancial en los costos, como se ve en figura 5.8. Cabe destacar a pesar de esto que con los tres tamaños de intervalos se obtienen soluciones del mismo costo, pero no podemos asegurar para 15 y 1 que sean óptimas.

Es claro que una mejora en los costos viene en detrimento de una menor cantidad de instancias resueltas de manera óptima. Creemos que una buena estrategia es combinar intervalos de 120 minutos con capacidades diarias en los casos en que esta no obtiene la solución óptima.

Por último, en la tabla 5.6 mostramos el gap promedio en el nodo raíz y el gap final. El primero se calcula como $\frac{|\lambda^* - Z^*|}{\lambda^*} * 100$ y el segundo como $\frac{|\overline{\lambda}^* - Z^*|}{\overline{\lambda}^*} * 100$, donde Z^* es el valor de la mejor solución entera encontrada, λ^* es el valor óptimo de la relajación lineal en el nodo raíz, y $\overline{\lambda}^*$ es el valor óptimo más grande entre todas las relajaciones lineales de los nodos abiertos del árbol de branch & bound. El gap raíz obtenido es ajustado y consistente

Tamaño del intervalo	Gap raíz	Gap final	Nodos cerrados (solo resueltos)	Nodos cerrados (solo no resueltos)
240	1.91	-	42.20	0.00
120	1.65	0.09	49.88	793.00
60	1.23	1.35	28.71	200.67
30	0.67	0.38	177.00	522.33
15	0.42	2.08	15.00	447.75
1	0.42	2.56	11.33	390.00

Tab. 5.6: Por un lado, gap de integralidad en el nodo raíz y cantidad de nodos cerrados del árbol branch & bound promediados entre las instancias resueltas. Por otro, gap de integralidad final y cantidad de nodos cerrados promediados entre las instancias no resueltas.

con los resultados obtenidos en soluciones branch & price para otros problemas con este mismo tipo de formulación (por ejemplo [3]). Para el caso de las instancias resueltas, un gap raíz menor significó generalmente una menor cantidad de nodos cerrados o explorados. En el caso de las instancias que no se resolvieron de manera óptima, el gap final es en promedio chico. También, se puede ver como a medida que se exploraron más nodos, más se achica el gap final.

Los resultados de la ejecución sobre nuestras instancias generadas, es decir, para el caso de intervalos de medición de 1 minuto, pueden encontrarse con mayor detalle en la tabla 7.1 del Apéndice.

6. CONCLUSIONES Y TRABAJOS A FUTURO

En esta tesis abordamos el problema de ruteo de vehículos con ventanas de tiempo, opciones de entrega y capacidades dinámicas. Hasta donde sabemos, no hay estudios que consideren el aspecto de las capacidades dinámicas dentro de la familia de VRPDO, por lo que esta tesis es también una presentación del problema.

Tomamos como punto de partida el trabajo realizado en Tilk et al sobre VRPDO [1], donde las capacidades de los puntos de entrega compartidos se asumen constantes a lo largo del horizonte de planificación. Continuamos con su propuesta de modelado de la red logística a través de un grafo a medida que agrupe las opciones de entrega que se encuentran en un mismo punto geográfico. También, incluimos en nuestra solución una de las heurísticas que ellos utilizan, Limited Discrepancy Search. Experimentamos con implementaciones y configuraciones alternativas de esta, con el objetivo de entender mejor aspectos de performance del algoritmo, así como también mejorar la solución existente propuesta por Tilk.

Planteamos una solución exacta utilizando la técnica branch and price, con heurísticas para acelerar la resolución del problema de pricing de variables. Observamos que gran parte del tiempo de ejecución del algoritmo se incurre en la validación de las condiciones de dominación, y para acelerar esta fase, propusimos utilizar heurísticas que relajen las condiciones de dominación y una estructura de datos para representar el conjunto de labels procesados que utiliza características de nuestros labels, como el uso de los puntos de entrega compartidos. También, proponemos un esquema de branching específico para nuestro problema, y probamos que este conduce a una solución óptima.

Creamos y compartimos un nuevo conjunto de instancias con capacidades dinámicas en los puntos de entrega compartidos, con el que realizamos la experimentación presente en este documento. Si bien no es en el mismo conjunto de instancias, ni el mismo problema, los resultados de la experimentación muestran que nuestras propuestas performaron peor que la configuración basada en lo que propone Tilk et al.

Utilizando nuestro conjunto de instancias, y proponiendo una metodología para agregar la información de las capacidades, mostramos que considerando las capacidades como dinámicas se puede obtener hasta una reducción de los costos logísticos que va del 5% al 37%. Esto ocurre en detrimento del tiempo del algoritmo, lo que nos lleva a resolver a optimalidad el 60% de las instancias que podemos resolver considerando las capacidades como estáticas. Los resultados muestran que reportando al menos 2 veces al día la capacidad de los puntos SDL, se puede obtener una reducción de costos del 20.05% en promedio, resolviendo un 90% de las instancias respecto a capacidades diarias.

Este trabajo deja abiertas varias líneas de investigación:

- Nos es evidente que nuestra solución no obtuvo la performance esperada, salvo con unos pocos casos, no pudimos resolver instancias de 50 clientes. Si bien una mejora puede obtenerse a través de una mejor implementación del mismo algoritmo propuesto, adaptando nuestra solución al problema específico de Tilk et al, notamos que en las instancias en las que ellos realizan cortes, sus tiempos son mucho mejores que los nuestros. Proponemos entonces que se extienda esta solución con cortes, que pueden ser tanto de familias conocidas como específicos para este problema.

- Si bien modelamos las capacidades como dinámicas, no entró en el alcance de este trabajo las devoluciones de pedidos en los puntos de entrega. Un cliente puede hacer la devolución de un pedido en los puntos de entrega compartidos, que debe luego ser retirado por los vehículos para ser transportado al depósito. No solo se agregan nuevas condiciones que una solución debe satisfacer, sino que también las capacidades ya no puede modelarse constante a trozos, y estrictamente creciente de a trozos. Creemos que gran parte de la dificultad hoy está en aproximar los modelos a la realidad, y con esto agregar una característica de esta índole, ya presente en la industria y la vida diaria, toma gran importancia.

- El tamaño en bytes que requiere un label puede impactar en la performance dado el gran número de labels factibles que se exploran. En este trabajo utilizamos una estructura llamada *bitset* para representar el conjunto S de vértices inalcanzables, pero al momento de generar nuestro grafo extendido G_{ext}, la cantidad de vértices que representan opciones de entrega en un punto SDL se multiplican por la cantidad de intervalos de capacidad de este. Este número hizo que tengamos que fijar el tamaño de nuestro bitset en 500 para toda la experimentación. Existen representaciones alternativas de S que requieren menos espacio y no se ven afectadas por la cantidad de intervalos, como por ejemplo, considerando pedidos en lugar de vértices, y marcando pedidos como inalcanzables cuando todos los vértices que representan opciones de ese pedido son inalcanzables. El problema que tuvimos con este tipo de representaciones, y que hizo que las descartemos, es que obtienen en la práctica una dominación más débil. Sin embargo, un uso más eficiente de la memoria para representar labels nos permite que más memoria pueda ser utilizada en otras partes del algoritmo, como en la estructura de dominación. Proponemos entonces que se estudien nuevas formas de representar este conjunto.

7. APÉNDICE

Tab. 7.1: Solución de instancias generadas descritas en

Instancia	Óptimo entero	Óptimo raíz	Tiempo total	Tiempo pricing	Tiempo LP	Tiempo branching	#Nodos cerrados	GAP raíz
V_0_small_25_5	-	1589.29	1800.12	1795.77	0.98	2.08	68	2.50
V_1_small_25_5	1414.00	1407.75	35.90	35.50	0.18	0.02	3	0.44
V_2_small_25_5	-	1511.95	1800.06	1446.40	58.66	230.79	1103	2.19
V_3_small_25_5	1802.00	1802.00	34.97	34.87	0.06	0.00	1	0.00
V_4_small_25_5	1802.00	1781.50	1552.71	1549.40	0.88	1.21	59	1.15
V_5_small_25_5	-	1831.59	1800.04	1733.05	15.54	34.66	318	1.99
V_6_small_25_5	1753.00	1753.00	95.92	95.75	0.10	0.00	1	0.00
V_7_small_25_5	1361.00	1348.33	78.00	77.65	0.17	0.03	3	0.94
V_8_small_25_5	1741.00	1741.00	53.10	52.95	0.08	0.00	1	0.00
V_9_small_25_5	-	1720.67	1800.26	1789.55	2.49	4.64	71	3.56
V_0_medium_25_5	-	1652.33	1853.06	1852.83	0.08	0.02	1	18.20
V_1_medium_25_5	-	1532.71	1800.49	1798.01	0.64	0.93	27	2.50
V_2_medium_25_5	-	-	1837.46	1837.25	0.12	0.00	0	-
V_3_medium_25_5	-	1253.44	1800.51	1800.29	0.10	0.03	1	6.27
V_4_medium_25_5	-	-	1806.43	1806.28	0.10	0.00	0	0.30
V_5_medium_25_5	1591.00	1586.20	881.66	880.55	0.34	0.30	19	-
V_6_medium_25_5	-	-	1802.98	1802.83	0.08	0.00	0	1.56
V_7_medium_25_5	-	1408.00	1801.69	1801.17	0.17	0.14	7	10.86
V_8_medium_25_5	-	1092.32	1800.06	1505.88	44.83	189.49	760	22.65
V_9_medium_25_5	-	1321.63	1800.18	1787.89	2.01	6.60	109	10.60
V_0_small_50_10	-	3091.44	1810.08	1808.72	0.44	0.09	1	13.54
V_1_small_50_10	-	2928.36	1850.08	1848.31	0.32	0.07	1	10.54
V_2_small_50_10	-	3162.75	1861.34	1860.06	0.27	0.06	1	15.80
V_3_small_50_10	-	3857.67	1872.74	1870.36	0.38	0.10	1	6.55
V_4_small_50_10	-	3067.13	1801.96	1800.30	0.53	0.19	2	0.22
V_5_small_50_10	-	2297.84	1806.11	1805.19	0.45	0.07	2	7.44
V_6_small_50_10	-	3150.61	1800.97	1799.62	0.40	0.11	1	6.14
V_7_small_50_10	-	2530.67	1800.56	1799.53	0.41	0.07	1	-
V_8_small_50_10	-	-	1812.91	1812.34	0.35	0.00	0	11.47
V_9_small_50_10	-	3274.52	1807.83	1806.97	0.27	0.06	1	-
V_0_medium_50_10	-	-	1809.46	1808.88	0.36	0.00	0	-
V_1_medium_50_10	-	-	1813.21	1812.76	0.27	0.00	0	-
V_2_medium_50_10	-	-	1812.84	1812.39	0.27	0.00	0	-
V_3_medium_50_10	-	-	1807.73	1806.95	0.46	0.00	0	-
V_4_medium_50_10	-	-	1824.24	1823.64	0.37	0.00	0	-
V_5_medium_50_10	-	-	1801.54	1800.80	0.44	0.00	0	-
V_6_medium_50_10	-	-	1820.50	1819.95	0.34	0.00	0	-
V_7_medium_50_10	-	-	1804.17	1803.54	0.39	0.00	0	-
V_8_medium_50_10	-	-	1812.44	1811.78	0.39	0.00	0	-
V_9_medium_50_10	-	-	1813.47	1812.99	0.30	0.00	0	-

BIBLIOGRAFÍA

[1] K. O. Christian Tilk and S. Irnich. The last-mile vehicle routing problem with delivery options. *Gutenberg School of Management and Economics and Research Unit "Interdisciplinary Public Policy"*, 2020.

[2] T. G. Christian Tilk, Ann-Kathrin Rothenbacher and S. Irnich. Asymmetry matters: Dynamic half-way points in bidirectional labeling for solving shortest path problems with resource constraints faster. *European Journal of Operational Research*, 2017.

[3] L. Costa, C. Contardo, and G. Desaulniers. Exact branch-price-and-cut algorithms for vehicle routing. *Transport. Sci.*, 53(4):946–985, 2019.

[4] Desrochers. La fabrication d'horaires de travail pour les conducteurs d'autobus par une méthode de génération de colonnes. *Ph.D Thesis, Centre de recherche sur les Transports*, 1986.

[5] Dror and Trudeau. Savings by split delivery routing. *Transportation Science*, 23(2):141–145, 1989.

[6] D. Feillet and Gendrau. The profitable arc tour problem: Solution with a branch and-price algorithm. *Transportation Science*, 2005.

[7] G. Feillet and Rousseau. New refinements for the solution of vehicle routing problems with branch and price. 2007.

[8] R. F. H. Jasmin Grabenschweiger, Karl F. Doerner and M. W. P. Savelsbergh. The vehicle routing problem with heterogeneous locker boxes. *Springer*, 2020.

[9] Mancini and Gangster. Vehicle routing with private and shared delivery locations. *Computers & Operations Research*, 2021.

[10] McKinsey. Parcel delivery: The future of last mile. Technical report, 2016.

[11] S. Rohmer and B. Gendron. A guide to parcel lockers in last mile distribution - highlighting challenges and opportunities from an or perspective. *Cirrelt*, 2020.

[12] P. Toth and D. Vigo, editors. *Vehicle Routing: Problems, Methods, and Applications.* 2nd ed. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2014.

[Referencias]