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

Un enfoque exacto para el problema de camino mínimo elemental con restricciones de recursos y tiempos de viaje variables

Tesis presentada para optar al título de
Licenciado en Ciencias de la Computación

Gonzalo Lera Romero

Director: Dr. Juan José Miranda Bront

Buenos Aires, 2017

UN ENFOQUE EXACTO PARA EL PROBLEMA DE CAMINO MÍNIMO ELEMENTAL CON RESTRICCIONES DE RECURSOS Y TIEMPOS DE VIAJE VARIABLES

La congestión en las grandes ciudades y en áreas sobrepobladas es uno de los mayores desafíos en la logística urbana, y hoy en día se ha convertido en uno de los inconvenientes principales en el planeamiento urbano y la denominada *distribución de la última milla* debido a su impacto económico, social y ambiental. La mayor parte de la investigación dedicada a los Problemas de Ruteo de Vehículos (VRP, por su sigla en inglés) considera que el tiempo de viaje entre dos sitios cualesquiera es constante a lo largo del horizonte de planeamiento. En los últimos años, ha habido una tendencia a enriquecer estos modelos mediante la incorporación de funciones de tiempo de viaje más complejas que capturen el efecto de la congestión, conocidos como VRP Dependientes del Tiempo (TDVRP, ver Gendreau et al. [11])). La incorporación explícita de la congestión a nivel táctico es un aspecto fundamental de los Sistemas de Soporte a la Decisión (DSS) modernos a fin de obtener planificaciones y recorridos que sean representativos de las operaciones habituales (Huang et al. [15]). Sin embargo, los algoritmos presentes actualmente en la literatura requieren mejoras, como se muestra en Dabia et al. [6] y Montero et al. [26], para poder transformarse en herramientas automatizadas que puedan ser utilizadas en la práctica.

Los enfoques exactos generales para las variantes mono-vehículo suelen basarse en modelos de Programación Lineal Entera (PLEM) y algoritmos Branch&Cut (B&C), como puede verse en los trabajos de Cordeau et al. [5] para el Problema del Viajante de Comercio Dependiente del Tiempo (TDTSP) y Arigliano et al. [3] para el TDTSP-TW, es decir, el TDTSP con ventanas de tiempo. Los algoritmos exactos para problemas multi-vehículo suelen recurrir a modelos de PLEM combinados con técnicas de descomposición, dando lugar a los denominados algoritmos Branch&Price (B&P). Uno de los ingredientes principales de estos algoritmos radica en la resolución del subproblema de generación de columnas, que puede ser formulado como un problema de Camino Mínimo Elemental Dependiente del Tiempo con Restricciones de Recursos y Beneficios (TD-PESPPRC). Dabia et al. [6] siguen este enfoque y proponen un algoritmo B&P para el TDVRP con Ventanas de Tiempo (TDVRP-TW), que es luego adaptado por Sun et al. [29] para considerar también restricciones de precedencia. En ambos artículos, en sus problemas correspondientes es abordado mediante algoritmos de *labeling* y Programación Dinámica (DP). La generación de columnas aparece como un problema difícil y desafiante computacionalmente, especialmente cuando no se posee un recurso limitante (por ejemplo, la capacidad o las ventanas de tiempo) que permita reducir de forma significativa el espacio de búsqueda.

En esta tesis abordamos el TD-PESPPRC con restricciones de capacidad pero, a diferencia de los enfoques anteriores, no consideramos la presencia de ventanas de tiempo en los clientes. Desde el punto de vista práctico, con esta configuración los algoritmos de labeling propuestos previamente encuentran fuertes limitaciones dado que el espacio de búsqueda se vuelve muy grande, dando lugar también a la construcción de rutas con muchos clientes como parte de las soluciones al problema. Luego, la importancia de estudiar esta variante tiene dos razones principales. La primera de ellas es que las versiones mono-vehículo, co-

mo el TDTSP y el TDTSP-TW, pueden ser fácilmente adaptadas introduciendo pequeñas modificaciones al modelo. En segundo lugar, la obtención de un algoritmo eficiente para el TD-PESPPRC abriría la posibilidad a integrarlo en los esquemas B&P en los casos donde los enfoques tradicionales no son satisfactorios. Con esto en mente, proponemos un modelo PLEM mejorado basado en el propuesto por Sun et al. [29]. Realizamos un estudio teórico enfocado en derivar nuestras familias de desigualdades válidas que exploten la estructura particular introducida por la red subyacente dependiente del tiempo, y que permita mejorar la calidad de las cotas inferiores provistas por la relajación lineal del modelo. Estas familias son embebidas en un algoritmo de tipo Branch&Cut, que es evaluado experimentalmente en función del tiempo de cómputo requerido y la calidad de las cotas provistas. Los resultados son muy prometedores y muestran que el algoritmo es capaz de resolver instancias de 25 vértices consistentemente dentro de los límites establecidos, que los resultados teóricos pueden ser extendidos a otras variantes de problemas dependientes del tiempo y que el enfoque en su totalidad tiene potencial de ser aplicado en la práctica.

Palabras claves: Programación Lineal Entera, Ruteo de Vehículos, Tráfico y Congestión, Optimización Combinatoria

AN EXACT APPROACH FOR THE TIME-DEPENDENT ELEMENTARY SHORTEST PATH PROBLEM WITH RESOURCE CONSTRAINTS

Congestion in large cities and populated areas is one of the major challenges in urban logistics, becoming one of the key issues in city planning and *last-mile logistics* due to its economic, social and environmental impact. Most of the research devoted to the Vehicle Routing Problem (VRP) considers that travel times between any two locations are fixed along the time horizon. In the last few years, there has been a trend to enrich these models by incorporating more complex travel time functions to capture the effect of congestion, known as Time-Dependent VRPs (TDVRP, see, e.g., Gendreau et al. [11], for an updated survey). Incorporating the congestion explicitly at a tactical level is a key aspect of modern Decision Support Systems (DSS) in order to obtain distribution plans that are representative of the real-life operations (Huang et al. [15]). However, state-of-the-art algorithms require further developments, as shown in Dabia et al. [6] and Montero et al. [26], in order to derive them into automated tools that can be implemented in practice.

Standard exact approaches for single-vehicle problems are generally addressed by means of Integer Linear Programming (ILP) and Branch&Cut (B&C) algorithms, as shown in Cordeau et al. [5] for the Time Dependent Traveling Salesman Problem (TDTSP) and by Arigliano et al. [3] for the TDTSP with Time Windows (TDTSP-TW). Exact approaches for multi-vehicle variants generally resort to ILP combined with decomposition techniques, resulting in *Branch&Price* (B&P) algorithms. One of the key ingredients of the latter rely on the pricing problem within the column generation algorithm, which can be formulated as a Time-Dependent Elementary Shortest Path Problem with Resource Constraints and Profits (TD-PESPPRC). Dabia et al. [6] follow this approach and propose a B&P algorithm for the capacitated TDVRP with time windows (TDVRP-TW), which is later adapted by Sun et al. [29] to further consider the TD-PESPPRC with precedence constraints. In both papers, the corresponding problem is tackled by labeling and Dynamic Programming (DP) algorithms. The pricing step appears as a very difficult and challenging problem that requires further developments, specially under the absence of a limiting resource (for instance, time windows) that may allow to reduce the search space.

In this thesis, we tackle the TD-PESPPRC with capacity constraints but, opposed to the previously mentioned cases, we do not consider time windows. From a practical standpoint, under these settings the standard labeling algorithms from previous approaches find some limitations since the search space becomes too large to explore, giving place to constructing large routes as part of the overall solution. Therefore, the importance of studying this variant has two main reasons. Firstly, single-vehicle variants, such as the TDTSP and the TDTSP-TW, are easily recovered with some minor modifications to the formulation. Secondly, an efficient algorithm for the TD-PESPPRC could be integrated within a B&P scheme where the standard approaches are not satisfactory. For this purpose, we propose an enhanced ILP formulation based on the one proposed in Sun et al. [29]. We perform a theoretical study focused on deriving three new families of valid inequalities that exploit the particular structure introduced by the time-dependent underlying network aiming to

improve the quality of the lower bounds provided by the LP relaxation. These families are embedded in a Branch&Cut algorithm, which is experimentally evaluated in terms of computing times and quality of the lower bounds. The results are very promising, showing that our algorithm is able to solve consistently instances with 25 vertices in reasonable computing times, that the results can be extended to other time-dependent problems and that the overall approach has potential to be applied in practice.

Keywords: Integer Lineal Programming, Vehicle Routing Problem, Congestion, Combinatorial Optimization

AGRADECIMIENTOS

Esta etapa de mi vida fue extensa y por lo tanto quiero escribir unas palabras de agradecimiento para aquellos que me acompañaron en este camino.

En primer lugar, quiero agradecer a Paula Zabala y Javier Marenco quienes aceptaron ser jurados de mi tesis y dedicaron tiempo a leer y evaluar el trabajo realizado.

Hubiese sido imposible disfrutar la carrera tanto como lo hice sin la aparición de ciertos personajes que hicieron que los días se vuelvan más divertidos. Gracias Rama, Agus, Charly, Fede "Fid" Canay, Fran, Lau y Tincho por compartir discusiones, clases, y mates. Gracias FNL, Partu, Lao, Titi y Sacha por los asados llenos de debate y exageración, los finales rendidos juntos y por ser tan excelentes personas, de las mejores que conocí en estos últimos años. Quiero también hacer un agradecimiento destacado para el "Dream-team" del cuál pude formar parte desde el día cero de la carrera y cuyos apodos van a quedar siempre en mi mente: Gabri, Chile y Ruso. Sepan que la mitad de esta carrera es de ustedes, que son grandes entre los grandes.

Además, por dos años y medio tuve la suerte de poder aprender el arte de la docencia de la mano de grandes personas como Alejandro, Javier y Guido. Sin dudas pude crecer mucho y aprender de ellos.

Un párrafo aparte para los que estuvieron siempre para entender mis formas de pensar, ayudarme con mis miedos y festejar mis triunfos. Nico, Urka y Adro, Chapeau! Gracias por bancarme, no se puede tener un mejor grupo de amigos que ustedes. También extiendo esto para Kanel, Ari, Ale, Champi, Jony, Isra y Luqui, que con sus anécdotas e historias logran cambiar malos días en buenos y estos en mejores.

Juanjo, quiero decir que fuiste el mejor director que me pudo haber tocado, sin lugar a dudas tuve suerte y encontré al director óptimo en el nodo raíz. Muchas gracias por tu generosidad, por todo el tiempo que dedicaste, por los consejos, el mate y la ayuda. No hace falta decir la excelente persona que sos porque todo aquel que te conoce ya lo sabe. Hacer la tesis con vos fue uno de los aciertos más grandes de la carrera. Mil gracias. Extiendo el agradecimiento también a Agus Montero que fue una parte importante en el comienzo de esta tesis. Gracias!

Por último, y creo que más importante, le agradezco a mi familia Lean, Papi, Mami y Vicky que me acompañaron a lo largo de todos estos años. Sin la ayuda y el apoyo de ustedes no hubiese llegado tan lejos. Los quiero mucho mucho.

En fin, muchas gracias.

Índice general

1. INTRODUCCIÓN

Hoy en día, la logística es una parte fundamental de una importante cantidad de empresas, ya sea porque deben distribuir su mercadería a todos sus clientes, o porque deben abastecer sus fábricas de materia prima. Más aún, la forma en que esta tarea es llevada a cabo por un gran porcentaje de organizaciones es mediante el uso de vehículos terrestres, camiones o camionetas, ya que otros medios como el ferroviario o el aéreo suelen ser más utilizados para mediana y larga distancia.

En muchos casos, la manera en que se planifica este conjunto de tareas, ya sea envíos o recolección de mercaderías, puede generar un cambio significativo en los gastos de una compañía. Es por esto que cada vez se vuelve más valioso el aporte de analizar este tipo de problemas.

A su vez, esta necesidad de aplicar a casos reales los modelos estudiados exige que estos sean cada vez más cercanos a la realidad. Por lo tanto, una primera aproximación casi evidente es introducir tráfico y congestión en los modelos de manera tal de poder simular la congestión que ocurre en las grandes ciudades.

Los Problemas de Ruteo de Vehículos (VRP, por su sigla en inglés) son una amplia familia de problemas que abarca la planificación y decisión acerca de diversas situaciones donde se dispone de una flota de vehículos, que debe llevar a cabo ciertas entregas (o pedidos), cumpliendo ciertas restricciones operacionales y buscando encontrar la *mejor* manera de hacerlo.

El criterio para decidir qué significa mejor varía según cada caso, aunque entre los más comunes se encuentran: menor tiempo de viaje, menor distancia recorrida, menor utilización de camiones (menos conductores) y menor tiempo de espera para los clientes, entre otras. A veces, incluso, es necesario balancear varios de los criterios mencionados aunque resulten opuestos.

Existen diferentes variantes del problema, que se encuentran explicadas en detalle en Toth y Vigo [31]. En este trabajo nos vamos a enfocar en la variante que tiene una flota limitada de camiones que debe visitar a un conjunto de clientes pasando exactamente una vez por cada uno de ellos. Más aún, existe un único depósito del cuál parten y al cuál arriban los camiones. Por ejemplo, esto permite modelar un supermercado que lleva en sus camiones los pedidos a todos sus clientes que optaron por una entrega a domicilio.

Una variante en particular es el VRP con Restricciones de Recursos (o VRPRC, por su nombre en inglés) que modelan algunos tipos particulares de restricciones operacionales. Esta variante también se explica en detalle en Toth y Vigo [31]. Hay varios tipos de recursos que suelen utilizarse para VRP, e.g., capacidades, ventanas de tiempo, precedencias, etc. En este trabajo vamos a considerar los primeros dos.

La primera, supone que los productos que se entregan a cada cliente son homogéneos

y que cada cliente indica cuántas unidades de cada uno quiere. Además, especifica cuánto es la capacidad de cada camión, o sea, la máxima cantidad de unidades que puede cargar (todos tienen la misma capacidad). Evidentemente, esto impone una restricción sobre las posibles rutas en una planificación y hace infactible aquellas que cargan más unidades de las posibles en un mismo camión. Siguiendo el ejemplo del supermercado, cada camión de entrega tiene una capacidad, por ejemplo el peso que puede cargar que está limitado por una máxima cantidad de kilogramos establecida por el fabricante, y cada cliente hace un pedido que en total, una vez embolsado, tiene un cierto peso.

El segundo recurso, las ventanas de tiempo, impone una nueva restricción a las soluciones estableciendo que cada cliente solicita que su pedido sea entregado en un rango horario que está comprendido entre un tiempo de inicio (o release) y uno de fin (o deadline). Esto nuevamente hace que ciertas soluciones sean infactibles, por ejemplo, en el supermercado, si un cliente quiere recibir su pedido entre las 8am y las 12am, es infactible visitarlo a las 3pm. Más aún, si otro cliente establece un pedido para recibirlo entre las 4pm y las 6pm, entonces no puede ser que un camión visite al segundo antes que al primero. En este segundo caso se puede establecer una precedencia entre clientes causada por sus ventanas de tiempo.

Otro aspecto del VRP que se puede refinar son los tiempos de viaje. En la versión más elemental, se define por cada par de ubicaciones (clientes o depósitos) la distancia entre ambos, y se asume que la velocidad es constante e independiente del momento en que se está viajando. Esto es una aproximación poco realista en muchos contextos dado que en las grandes ciudades la velocidad de viaje varía mucho según el momento del día.

Por ejemplo, un comportamiento típico en grandes ciudades son los picos de congestión por la mañana y la tarde-noche, donde las personas se movilizan hacia y desde sus trabajos, causando mayor tráfico y por lo tanto velocidades de viaje más reducidas mientras que en el resto del día las calles están más despejadas y se pueden transitar de manera más veloz. Es por esto que cada vez es más necesario tenerlo en cuenta a la hora de modelar los problemas.

Para acercar la planificación que se consigue a la realidad, es importante tener en cuenta que los tiempos de viaje no son independientes del momento del día, y es por esto que existe una rama llamada Problemas de Ruteo de Vehículos Dependiente del Tiempo (o TD-VRP).

En esta variante se fija un horizonte de tiempo, es decir, un intervalo en el tiempo bajo el cuál la planificación se ejecuta, por ejemplo un día, y se particiona en los denominados *Speed Zones*. Dentro de cada Speed Zone se pueden considerar tiempos/velocidades de viaje distintas, buscando capturar la variabilidad y la dependencia temporal.

Por ejemplo, si el horizonte de tiempo es la duración de un día laboral, de 8am a 8pm, entonces se puede partir en 3 Speed Zones, de 8am a 11am, de 11am a 6pm, y de 6pm a 8pm. El primero y tercer Speed Zone corresponden a horas pico, en donde la gente va y sale del trabajo, y por lo tanto las calles están más congestionadas. Sin embargo, el del medio corresponde a horarios donde no hay tanto movimiento vehicular, llevando a

velocidades más altas.

Si bien particionar el día en 3 Speed Zones es más real que considerar una única velocidad independiente del tiempo, cuanto más granularidad se utilice para la representación a lo largo del día, más cercano a la realidad resulta el modelo. Sin embargo, esto también causa que el modelo sea cada vez más complejo.

Camino Mínimo Una variante más restringida del VRP es lo que se conoce como Problemas de Camino Mínimo (SPP). En estos problemas, lo que se intenta es encontrar el camino más corto entre dos puntos A y B. Al igual que con el VRP, existen diversas variantes que se pueden ir agregando al problema llevando a que tome una dificultad superior.

En primer lugar, decimos que un camino es *elemental* cuando no repite vértices, por lo tanto, la variante Camino mínimo Elemental (ESPP) es aquella que busca el camino más corto sin repetir nodos. Luego, al igual que en VRP, se puede agregar recursos con sus respectivas restricciones (ESPPRC), por ejemplo, ventanas de tiempo y capacidad, tomando el mismo significado que el explicado anteriormente. Más aún, se define el TD-ESPPRC como el ESPPRC donde cada eje se atraviesa con una velocidad que depende del instante del tiempo en que se pasa. Estas velocidades están definidas al igual que en el VRP, es decir, dividiendo el horizonte de tiempo en Speed Zones.

Por último, una nueva variante para este tipo de problemas es lo que se conoce en la literatura como Profitable TD-ESPPRC (TD-PESPPRC). En esta variante, cada nodo tiene un beneficio (o profit) que se recibe por visitarlo. Retomando el ejemplo del supermercado, se puede asociar a cada cliente un beneficio que es el monto total de su compra, y definir que los clientes a los que no se les entrega el pedido se les devuelve el dinero. Entonces, ahora el objetivo puede ser encontrar una ruta que equilibre el dinero que se gasta en combustible con el monto que se gana por cada entrega. En este caso algunos clientes puedan no ser visitados si esto resulta conveniente, por ejemplo, un cliente que vive en otra provincia implicaría un camión viajando por días y esto es más perjudicial que la plata que se cobra.

1.1. Revisión de la literatura

Gran parte de la investigación relacionada con el VRP considera que el tiempo de viaje entre dos puntos está fijo a lo largo del horizonte de tiempo. Como se mencionó anteriormente, una descripción de las variantes y métodos del VRP se encuentra en Toth y Vigo [31]. Últimamente hay una inclinación por acercar estos modelos a la realidad, incorporando funciones de tiempo de viaje más complejas, que tienen en cuenta la congestión que se produce en las rutas. Esto es particularmente importante para la logística urbana, donde el tráfico varía significativamente a lo largo del día.

Un resumen reciente de las variantes de TDVRP se encuentra disponible en Gendreau et al. [11], el cuál tiene en cuenta tanto algoritmos exactos como también heurísticos. Las aplicaciones comerciales son, por lo que conocemos, bastante acotados en la práctica.

Una de las variantes que recibió más atención en los últimos años es la conocida como Time-Dependent Traveling Salesman Problem (TDTSP, por sus siglas en inglés), la cuál considera un solo vehículo y no tiene en cuenta su capacidad. Este problema, por lo tanto, es equivalente a encontrar un circuito Hamiltoniano de mínimo costo el cuál depende de alguna función de tiempo de viaje en particular. En este contexto, el mismo nombre TDTSP fue utilizado al considerarse distintas funciones de tiempo de viaje. A continuación describimos algunas de ellas.

Lo más simple es el modelo considerado por Picard y Queyranne [27], que tiene aplicaciones en el problema de Scheduling y que generaliza el problema conocido como Traveling Deliveryman Problem (ver, e.g. [9, 19, 23]). La mejora que presenta es la consideración de que el costo de viajar entre dos ciudades no solo depende de la distancia sino también de la posición de esas ciudades en el circuito. Algoritmos exactos para este problema se encuentran en [1, 12, 13, 25].

Malandraki y Daskin [20] proponen un modelo distinto, donde el tiempo de viaje entre dos ciudades depende del momento en el horizonte de tiempo en el que el arco es atravesado. Para lograr esto, se divide el horizonte en distintos intervalos llamados *time periods* y el tiempo de viaje se define entonces como una función constante a trozos sobre estos períodos. Esto permite capturar el efecto de la congestión en distintos momentos del día. Algoritmos exactos que consideran este modelo se encuentran en [2,21,22,24,28]. Respecto a las aplicaciones Furini et al. [10] formulan un modelo TDTSP para el *Aircraft Sequencing Problem*.

Una de las críticas más fuertes al modelo anterior es que los tiempos de viaje no necesariamente satisfacen la propiedad FIFO (First In, First Out), que indica que no se puede llegar al destino antes habiendo salido más tarde. Esta condición generalmente es deseada en una red de transportes de vehículos. Para combatir esta dificultad Ichoua et al. [16] desarrolla un modelo sobre el propuesto por Hill y Benton [14] que tiene la misma base que el de Malandraki y Daskin [20] pero en lugar de indicar el tiempo de viaje en cada time period se especifica la velocidad promedio. Los tiempos de viaje se pueden computar a partir del instante de partida desde un cliente en particular, recorriendo la distancia a la velocidad promedio del período y ajustando esta velocidad al cruzar las fronteras hacia otro período.

El modelo propuesto por Ichoua et al. [16] atrajo la atención de varios investigadores. Por ejmplo, Cordeau et al. [5] aborda el TDTSP utilizando como función objetivo minimizar el makespan (lo más temprano que se puede terminar el recorrido). También, estudian algunas propiedades de la función de tiempos de viaje, como por ejemplo una cota inferior que se consigue a partir de resolver un problema auxiliar TSP con tiempos de viaje constantes. Además, proponen un algoritmo Branch&Cut con el cuál logran resolver instancias de hasta 40 clientes. Otros trabajos presentados donde se aborda la variante de TDTSP con ventanas de tiempo (TDTSP-TW) se encuentran en [3, 26]. Allí se aborda el problema utilizando Programación Lineal Entera y aprovechando que las ventanas de tiempo permiten hacer un preprocesamiento sobre la red de transportes logrando reducir el tamaño del modelo.

Versiones multivehículo de TDVRP también son resueltas con algoritmos exactos que toman el modelo propuesto por Ichoua et al. [16]. Por ejemplo, Dabia et al. [6] aborda el TDVRP con ventanas de tiempo y el objetivo de minimización de duración total del viaje en lugar del makespan. Ellos proponen un modelo *set-partitioning* que es desarrollado mediante un algoritmo Branch&Price donde el problema de generación de columnas se resuelve por medio de un algoritmo de *labeling*. Los autores logran resolver consistentemente instancias de 25 clientes, varias de 50 y solamente algunas con 100, pareciendo ser este el límite del algoritmo. Parecido a esto se encuentra el trabajo de Sun et al. [29], que propone una variante de TSP con profits, ventanas de tiempo, y un tipo particular de restricciones de precedencia llamadas *pickup y delivery*. Para resolver esta variante proponen un modelo de Programación Lineal Entera en el cuál está basado este trabajo. Allí no lo analizan en detalle por su performance y debido a que al tener restricciones de ventanas de tiempo pueden resolverlo de manera más eficiente mediante un algoritmo personalizado de labeling que es una extensión del propuesto en Dabia et al. [6].

1.2. Objetivos y contribuciones

El objetivo principal de esta tesis es explorar mediante técnicas de Programación Lineal Entera la factibilidad práctica de desarrollar algoritmos exactos para el TD-PESPPRC que, como se menciona anteriormente, busca incorporar el efecto de la congestión a los modelos para acercar la planificación táctica a la ejecución de las operaciones. Mejoras en los algoritmos se traducen en soluciones más eficientes que reducen costos, congestión e incrementan el nivel de calidad de servicio.

La elección del problema se basa en dos motivos principales. El primero de ellos es que el TD-PESPPRC puede ser usado, con pequeñas modificaciones, para abordar otras variantes de problemas con tiempos de viaje variables, como por ejemplo el TDTSP, TDTSP-TW, Time-Dependent Orienteering Problem, entre otros. Luego, la mayoría de los desarrollos y contribuciones en esta tesis pueden ser consideradas también para estos problemas. La segunda razón radica en que el TD-PESPPRC aparece naturalmente como subproblema de generación de columnas en el contexto de algoritmos de tipo Branch&Price, y por lo tanto las mejoras obtenidas pueden tener un impacto indirecto en la resolución de variantes multi-vehículo, problemas en los cuáles este tipo de enfoque es estándar.

Si bien el TDVRP ha recibido una creciente atención en los últimos años, la variedad de enfoques y algoritmos es relativamente escasa hoy en día. Para el TDTSP y TDTSP-TW, algunos de los enfoques exactos de tipo Branch&Cut se centran alrededor de un modelo particular, que explota el cálculo de cotas inferiores de buena calidad, que en la práctica es capaz de resolver instancias de hasta 25 nodos consistentemente (ver [3, 5]). Recientemente, se han obtenidos buenos resultados para el TDTSP-TW (ver Montero et al. [26]) utilizando un algoritmo Branch&Cut sobre un modelo alternativo. Sin embargo, el algoritmo utiliza principalmente resultados generales extraídos del TSP-TW con tiempos de viaje constante, sin explotar las particularidades de la red de transporte más allá de algunas reglas de preprocesamiento para las ventanas de tiempo.

Por otro lado, el TD-PESPPRC y algunas variantes han sido abordadas principalmente

como parte del desarrollo de algoritmos de tipo Branch&Price. En estos en casos, en general las instancias suelen tener algún recurso restrictivo (ventanas de tiempo, capacidad, etc.) que en la práctica acota el tamaño de las rutas factibles. Los trabajos abordan este tipo de problemas usando algoritmos de labeling combinados con reglas de dominancia que permiten achicar significativamente el espacio de búsqueda explotando la presencia de estos recursos mencionados anteriormente. Sin embargo, este enfoque pierde efectividad ante la falta de algún recurso limitante. En los últimos años, algunos trabajos exploran la posibilidad de considerar modelos de Programación Lineal Entera en el contexto de algoritmos de generación de columnas para problemas de ruteo de vehículos, justamente haciendo hincapié en estos casos y buscando evitar la utilización de relajaciones que resulten en cotas inferiores de peor calidad.

Para ello, en este trabajo nos proponemos considerar un enfoque unificador a los mencionados previamente. Tomando como punto de partida la formulación propuesta en Sun et al. [29], desarrollamos una algoritmo exacto de tipo Branch&Cut que considere la dependencia del tiempo, con flexibilidad para modelar la presencia de recursos (limitantes o no). En esta dirección, las contribuciones específicas de la tesis son:

- Se estudia una variante de Problemas de Ruteo de Vehículos con tiempos de viaje variables muy relevante en términos prácticos.
- Se considera el modelo propuesto por Sun et al. [29], también considerado en Montero et al. [26] que obtiene muy buenos resultados para el TDTSP-TW. En particular, analizamos un conjunto particular de las restricciones del mismo y obtenemos una formulación más ajustada.
- Se realiza un estudio poliedral que explota la estructura particular de la red de transporte obtenida a partir del modelo de velocidades variables propuesto en Ichoua et al. [16]. Específicamente, se derivan tres familias de desigualdades que aprovechan la información provista por los tiempos de viaje variables. Si bien se aplican en el caso sin ventanas de tiempo, las mismas pueden ser fácilmente adaptadas también en este contexto. A su vez, se estudian los problemas de separación asociados y se proponen algoritmos que luego son evaluados en la práctica. Hasta donde llega nuestro conocimiento, no existe aún en la literatura un estudio de estas características para esta variante del TDVRP.
- Se propone una heurística inicial que puede ser utilizada para buscar soluciones factibles rápidamente y así poder obtener una cota superior inicial.
- Se implementa un algoritmo de tipo Branch&Cut que incorpora los desarrollos anteriores, además de algunas desigualdades válidas obtenidas de problemas relacionados sin tiempos de viaje variables.
- Los desarrollos propuestos son evaluados experimentalmente en un conjunto de 30 instancias que consideran un entorno distinto a los habituales: instancias sin ventanas de tiempo. De esta forma podemos analizar el comportamiento de la formulación propuesta cuando, por ejemplo, las técnica de preprocesamiento no pueden aplicarse con el fin de reducir el tamaño del problema y, por lo tanto, reducir el tamaño del modelo resultante.

1.3. Estructura de la tesis

En el capítulo 2 se explican las técnicas que se utilizan en la tesis, así como también se ofrece referencias para profundizar más sobre esos temas.

En el capítulo 3 se define formalmente el problema con el que se trabaja y toda la notación relacionada con el mismo. Además se hace un análisis detallado del impacto de agregar dependencia del tiempo en las velocidades a un problema de ruteo. Adicionalmente, se agregan nuevas definiciones que son necesarias para las demostraciones de capítulos posteriores.

Más tarde, en el capítulo 4 se presenta el modelo de Programación Lineal Entera que se utiliza para atacar el problema y un estudio de familias de desigualdades válidas para este modelo. A su vez, se proponen nuevas familias de desigualdades, se demuestra su validez y se proponen algoritmos de separación.

Luego, en el capítulo 5 se especifica cómo es la experimentación que se lleva a cabo para poner a prueba el modelo y las desigualdades presentadas en el capítulo 4. Entre otras cosas se define cómo se configura CPLEX, en qué computadora son ejecutados los experimentos, y cómo se lleva a cabo este proceso.

En el capítulo 6 se exhiben los resultados obtenidos en la experimentación y se hace un breve análisis de ellos. Por último, en el capítulo 7 se ofrecen conclusiones generales del trabajo y posibles direcciones por dónde puede ser continuado.

2. PRELIMINARES

2.1. Programación Lineal

La Programación Lineal (PL) permite expresar modelos para encontrar una valuación en $\mathbb{R}_+$ para un conjunto de variables x no negativas, de manera tal que se maximice (o minimice) una función objetivo $z : \mathbb{R}^n \to \mathbb{R}$ y que se cumpla un conjunto de desigualdades lineales $Ax \leq b$ ($A \in \mathbb{R}_+^{m \times n}, b \in \mathbb{R}_+^m$). Todo problema de PL puede escribirse de la siguiente forma:

$$\begin{array}{ll} \textbf{min} & z(x) = cx \\ \textbf{sujeto a} & \\ & Ax \leq b \\ & x_i \geq 0 \quad \forall i = 1 \dots n \end{array}$$

A su vez, esto tiene una interpretación geométrica que se basa en entender al grupo de restricciones $Ax \leq b$ como rectas que definen a un poliedro en $\mathbb{R}_+^n$. Por ejemplo, en la Figura 2.1 se grafica el siguiente Modelo PL:

$$\begin{array}{ll} \textbf{min} & z(x,y) = x + y \\ \textbf{sujeto a} & \\ & \frac{1}{3}x - y \leq -2 \\ & 2x - y \leq 5{,}5 \\ & \frac{1}{2}x + y \geq 2 \\ & 2x - y \geq 0{,}5 \\ & x, y \in \mathbb{R}_+ \end{array} \tag{2.1}$$

Existen algoritmos que encuentran una solución óptima para cualquier Modelo PL en tiempo polinomial (e.g. ver Karmarkar [18]). Además, existen otros algoritmos que si bien no son polinomiales funcionan bien en la práctica como es el caso de Simplex, algoritmo propuesto por Dantzig [4]. Hoy en día existe una variedad de productos que implementan estos algoritmos de manera eficiente tales como CPLEX, AMPL, Gurobi, etc.

2.2. Programación Lineal Entera Mixta

Si bien PL permite modelar una gran variedad de problemas, hay muchos que no pueden ser resueltos utilizando solamente variables continuas y que necesitan restringir el dominio de las variables a los enteros ($\mathbb{Z}$). Estos problemas se pueden modelar mediante los llamados Modelos de Programación Lineal Entera (PLEM). Un PLEM tiene la siguiente estructura:

$$\begin{array}{ll} \textbf{min} & z(x) = cx \\ \textbf{sujeto a} & \\ & Ax \leq b \\ & x_i \geq 0 \quad \forall i \in C \cup I \\ & x_i \in \mathbb{N} \quad \forall i \in I \end{array} \tag{2.2}$$

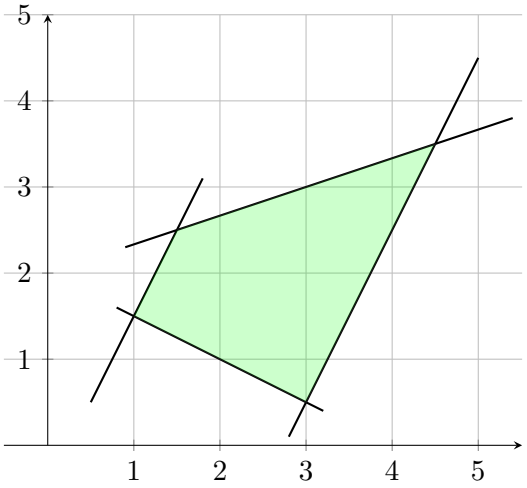


Fig. 2.1: Ejemplo de poliedro descripto por sistema de ecuaciones PL 2.1.

Donde $x = I \cup C$, siendo I las variables que solo pueden tomar valores enteros, y C aquellas que pueden tomar cualquier valor continuo. Si llamamos $P = \{x \in \mathbb{R}^n \mid Ax \leq b, x \geq 0\}$ al poliedro que describe el modelo, el conjunto de soluciones factibles es $S = \left\{x \in P \mid I \in \mathbb{N}^{|I|}\right\}$. Además, decimos que $\bar{x} = \text{mín}\{z(x) \mid x \in P\}$ es la relajación lineal de P. Y como $S \subseteq P$, entonces se puede afirmar que la solución óptima x^* de S cumple que $z(x^*) \geq z(\bar{x})$, es decir, la relajación del PLEM es una cota inferior de la solución óptima dado que estamos considerando un problema de minimización.

Desde el punto de vista geométrico, se llama $conv(S)$ a la cápsula convexa de S, o sea, el poliedro más chico que contiene a todos los puntos de S. Si $conv(S)$ se puede representar con un conjunto polinomial de restricciones, entonces se puede hallar la solución óptima utilizando cualquier algoritmo que resuelva PL. En la Figura 2.2 se puede ver la representación gráfica del PLEM de ejemplo que se presenta a continuación, donde P está de color verde y $conv(S)$ de color azul:

$$
\begin{array}{ll}
\textbf{min} & z(x,y) = x + y \\
\textbf{sujeto a} & \\
 & \frac{1}{3}x - y \leq -2 \\
 & 2x - y \leq 5{,}5 \\
 & \frac{1}{2}x + y \geq 2 \\
 & 2x - y \geq 0{,}5 \\
 & x, y \in \mathbb{N}
\end{array}
$$

El problema de resolver un PLEM general pertenece a la clase NP-Hard, ya que varios problemas que también están en NP-Hard se pueden modelar como PLEM, por ejemplo, el VRP mencionado anteriormente. Como mencionamos anteriormente, se puede probar que si $P = conv(S)$ y se tiene una cantidad polinomial de restricciones para representar a P, entonces se puede utilizar cualquier algoritmo de PL para encontrar la solución óptima de S. Sin embargo, no se sabe si encontrar la cápsula convexa de un conjunto cualquiera S es polinomial, o si la cantidad de restricciones para representarla lo es. A continuación, se presenta las técnicas más frecuentes para resolver PLEM.

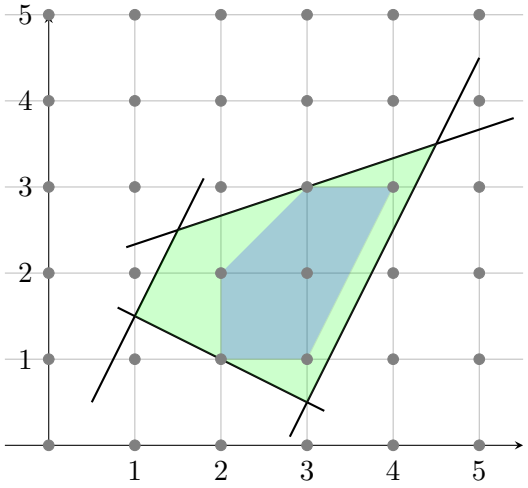


Fig. 2.2: Ejemplo de poliedro descripto por sistema de ecuaciones PLEM 2.2.

2.3. Algoritmos para PLEMs

2.3.1. Branch&Bound (B&B)

La primera estrategia para resolver PLEM se basa en una utilización de los algoritmos de PL. Cuando se aplica uno de estos algoritmos sobre un PLEM, la estrategia más común es relajar las restricciones que restringen el dominio de las variables a los enteros. Es por esto que la solución óptima que arroja el algoritmo puede contener alguna variable entera tomando un valor fraccionario, lo cual, no es factible en el problema original. En estos casos lo que se hace es aplicar la táctica Divide & Conquer y se "Branchea" o divide el problema en dos subproblemas, de manera tal que la solución de la relajación no sea solución de ninguno de ellos pero a su vez no se pierda ninguna solución entera.

Comúnmente, se utiliza el denominado Branching por Variable, que consiste en tomar una variable binaria que tenga un valor fraccionario en la solución de la relajación y crear dos subproblemas: uno asignando la variable al valor 1, y la otra al 0. El criterio para elegir cual variable x tomar puede variar según cómo se va generando el árbol de ejecución, entre los más comunes están: x más cercana a 0.5, x más cercana a 0, y x más cercana a 1.

Por otro lado, para evitar explorar ramas del árbol que no lleven a una solución óptima, se aplica el denominado "Bounding", que consiste en aplicar podas sobre el árbol basadas en cotas del funcional. Generalmente, durante la ejecución del algoritmo se mantiene una solución x_{UB} (x_{LB}) que es la mejor encontrada hasta el momento, y la cuál sirve de cota superior (inferior), si el problema es de minimización (maximización), la cual permite estando en un nodo dejar de explorarlo si la relajación vale $z(x^*) > z(x_{UB})$ ($z(x^*) < z(x_{LB})$). Esta cota también suele llamarse "Cota Primal".

Otro de los parámetros que se puede ajustar es cómo se recorre el árbol de nodos que se va generando. Este puede ser recorrido de las maneras más convencionales como Depth First Search (DFS), lo cuál es particularmente conveniente para conseguir Cotas Primales de manera veloz y para no utilizar tanta memoria, o Best Bound First (BBF) que utiliza

más memoria pero pretende minimizar el número de nodos explorados al elegir siempre el nodo con mejor "Cota Dual", o sea, valor de la relajación.

2.3.2. Cutting-Planes

Otra forma de abordar problemas con variables enteras o binarias, es mediante el agregado de *planos de corte*. Un plano de corte es una restricción que se cumple para toda solución en S, pero no se cumple para la solución óptima fraccionaria de la relajación del problema. Un esquema general que representa a los algoritmos de planos de corte se define en los siguientes pasos:

1. Resolver la relajación. Sea x^* el óptimo.
2. Si $x^* \in S$, terminar.
3. Encontrar $a'x \leq b'$ una restricción válida para todo $x \in S$ pero violada por x^* y agregarla a la formulación.
4. Volver al Paso 1.

Mediante esta técnica se busca ir acercándose a representar con restricciones lineales a $conv(S)$ ya que como se mencionó anteriormente, si se obtiene un PL que describa $conv(S)$ entonces va a tener una solución óptima que pertenece a S.

Una de las desventajas de esta técnica es la dificultad de encontrar planos de corte que se ajusten a la cápsula convexa de S, ya sea porque es computacionalmente ineficiente o por la dificultad teórica que esto implica. Es por esto que esta técnica no suele utilizarse por sí misma sino en conjunción con Branch&Bound.

2.3.3. Branch&Cut (B&C)

La técnica Branch&Cut es una combinación de las mencionadas previamente B&B y Cutting-Planes. Este algoritmo surge a partir de la necesidad de reducir la enumeración de nodos al utilizar B&B. Para lograr esto, la idea es aplicar una cierta cantidad de iteraciones de planos de corte en los nodos del árbol para así lograr ajustar su cota dual y por lo tanto poder aplicar Bounding de manera más frecuente.

Esta técnica es particularmente interesante en la etapa de experimentación porque su eficiencia depende de un compromiso entre la cantidad de iteraciones de planos de corte y la cantidad de nodos que se enumeran. Esto debe ser ajustado ya que las iteraciones de plano de corte pueden ser muy costosas y a partir de cierto número la enumeración resulta más eficiente.

Uno de los parámetros a ajustar es la cantidad de iteraciones de planos de corte que se llevan a cabo en cada uno de ellos y otro es la cantidad total de desigualdades se agregan al modelo, por ejemplo. Una estrategia que a veces se utiliza es realizar más iteraciones de plano de corte cuanto más cerca del nodo raíz se está. Esto en general ofrece un buen compromiso entre tiempo invertido en la separación de desigualdades y enumeración de nodos. En particular, cuando solamente se utilizan iteraciones de planos de corte en el nodo raíz, el algoritmo se conoce como Cut&Branch.

2.3.4. Generación de Columnas (GC)

En algunas formulaciones PL sucede que la cantidad de variables de decisión es exponencial en función del tamaño del problema. La formulación completa puede ser muy ineficiente de resolver o imposible en la práctica. Para atacar este problema se utiliza una técnica llamada Generación de Columnas.

Como se tiene un número muy grande de variables la idea es comenzar con un conjunto de columnas acotado y ejecutar Simplex, pero con la salvedad de ir generando nuevas columnas dinámicamente cuándo no se encuentre ninguna entre las existentes que pueda entrar a la base. De este modo, se espera poder llegar al óptimo sin la necesidad de considerar todo el conjunto de variables.

Sea M un PL con un número grande de variables, sea x el conjunto de todas sus variables. Para resolver M mediante GC hay que formular un modelo M' similar a M, que difiere únicamente en que su conjunto de variables $x' \subseteq x$ tiene solamente un subconjunto del total. A M se lo llama Problema Maestro (PM) y a M' Problema Maestro Restringido (PMR).

Luego, se resuelve M' mediante cualquier método de resolución de PL. Como hay variables que no están incluidas en M', entonces no se puede afirmar que la solución encontrada es óptima. Sin embargo, se puede probar que una variable $x_i \notin x'$ puede cambiar la solución óptima únicamente si su costo reducido $cr_{x_i} = c_{x_i} - y_i^* x_i < 0$ donde y_i^* es el costo de x_i en la función objetivo del problema dual asociado. Teniendo esto en cuenta, GC se puede definir con los siguientes pasos:

1. Formular M' PMR de M con $x' \subseteq x$.
2. Resolver M'.
3. Buscar alguna variable x_i tal que $cr_{x_i} < 0$. Si no existe ninguna, terminar.
4. Agregar x_i a x' y repetir el Paso 1.

Al tercer paso, es decir, al problema de buscar alguna variable con costo reducido negativo se lo conoce con el nombre de Problema Esclavo o Problema de Pricing (PP). Este problema puede ser resuelto con otra técnica distinta que el PMR, siendo usual el uso de Programación Dinámica, Labeling, o PLEM en los casos dónde la complejidad del PMR pertenezca a la clase NP-Hard.

2.3.5. Branch&Price (B&P)

Cuando un PLEM tiene un número exponencial de variables, no se puede resolver la relajación debido a que esta se vuelve muy costosa, y por lo tanto no se puede usar el algoritmo de Branch&Bound. Sin embargo, combinando esta técnica con GC, se obtiene la técnica llamada Branch&Price.

En cada nodo en lugar de resolver la relajación con un algoritmo para PL, se utiliza GC. Esto hace considerablemente más pesada la enumeración, pero suele tener mejores cotas que modelos alternativos, más compactos, que tienen un número polinomial de variables,

haciendo que sea interesante abordar los problemas mediante el uso de esta técnica. Una observación es que cuándo la técnica de GC se combina con un algoritmo B&C, se lo denomina Branch&Price (B&P).

2.4. Heurística

En ciertos problemas donde la dificultad de resolverlos de manera exacta es muy elevada, a veces se puede hacer un compromiso y buscar encontrar una solución "buena", aunque no sea la mejor, a cambio de tener un tiempo de computo razonable. Este es exactamente el propósito de un algoritmo heurístico. Una observación válida es que los algoritmos heurísticos pueden no encontrar ninguna solución.

2.4.1. Goloso

Hay distintos tipos de estrategias para hallar soluciones. Una de las más utilizadas por su simpleza y velocidad es la técnica llamada Goloso (o Greedy). Básicamente, se comienza con una solución vacía o trivial y se busca ir extendiéndola de a un paso a la vez buscando maximizar algún criterio con decisiones locales que finalmente pueden no ser óptimas. Por ejemplo, en VRP se puede ir construyendo un camino partiendo desde el depósito e ir agregando nodos (un paso) al final de este si minimizan la duración parcial (criterio).

2.4.2. Búsqueda Local

Otra técnica un poco más compleja es la Búsqueda Local. La idea es plantear todo el conjunto de soluciones como un gran grafo donde las soluciones son los nodos y están conectadas entre sí si son parecidas, con algún criterio establecido. Una vez definido este grafo, se parte de una solución inicial ya conocida y se lo navega, con algún criterio, yendo a través de sus ejes hasta llegar a una solución satisfactoria.

Lo primero que se debe hacer es definir un criterio de Vecindad que indica si dos soluciones son adyacentes o no en el grafo. Un ejemplo de este criterio: si las soluciones fuesen cadenas de caracteres, es crear un eje entre ellas si difieren en un solo carácter. De este modo la solución "casa" y "cama" son adyacentes, pero "casa" y "pala" no.

Una de las cosas interesantes a la hora de definir la Vecindad es tener en cuenta su tamaño. Si es muy grande, el grafo va a ser más denso y el camino hacia una solución óptima va a ser más corto, logrando que se necesiten pocos movimientos para hallar una buena solución. Sin embargo, el costo de computar tal Vecindad puede ser muy grande, incluso no polinomial. Es por esto, que se busca en general equilibrar estas dos variables.

Otro criterio que se debe definir es cómo recorrer el grafo. Dos maneras muy populares son las llamadas First-accept y Best-accept.

- *First-accept:* Al recorrer el vecindario, si se encuentra una solución mejor que la actual se mueve inmediatamente hacia ella.
- *Best-accept:* Se mueve hacia la mejor solución vecina de la actual, si esta es mejor.

Por último, se debe definir los movimientos en sí mediante lo que se conoce como operadores. Los operadores son pasos a seguir para encontrar soluciones vecinas. Un operador, por ejemplo, en el caso de las cadenas de caracteres nombradas anteriormente es sustituir una letra por otra, de esta manera sustituyendo la letra 's' por la 'm' de la palabra "casa" se llega a su solución vecina "cama".

El esquema general de los algoritmos de Búsqueda Local se muestra en el Algoritmo 1.

Algorithm 1 Esquema general de Búsqueda Local First-Accept y Best-Accept

```
1: function BusquedaLocal-FirstAccept(S_0)
2:     S* ← S_0
3:     while ¬CriterioParada(S*) do
4:         for S ∈ V(S*) do
5:             if c(S) < c(S*) then
6:                 S* ← S
7:                 break
8:     return S*
1: function BusquedaLocal-BestAccept(S_0)
2:     S* ← S_0
3:     while ¬CriterioParada(S*) do
4:         S' ← S*
5:         for S ∈ V(S*) do
6:             if c(S) < c(S') then
7:                 S' ← S
8:         S* ← S'
9:     return S*
```

3. DEFINICIÓN DEL PROBLEMA

A continuación se presentan los problemas y definiciones con los que se trabaja en este documento. Se introduce además la red de transporte, el modelo de velocidades dependientes del tiempo utilizado, cómo se traduce en tiempos de viaje y se presentan definiciones y propiedades básicas sobre el mismo que serán utilizadas en los siguientes capítulos.

3.1. TD-VRPRC

El TD-VRPRC considera la siguiente red en la cual los vehículos realizan sus movimientos. Sea $G = \langle V, E \rangle$ un digrafo con n vértices, con $V = \{v_s, v_e, 1, 2, \ldots, n-2\}$, donde v_s representa al depósito de salida, v_e al de llegada y cada vértice i representa al cliente i. Además, para cada eje $(i, j) \in E$ se define su distancia d_{ij}. Por otra parte, K indica la máxima cantidad de vehículos que se puede utilizar para la planificación.

Se define también lo necesario para representar las restricciones de recursos: Q indica la máxima capacidad de cada camión, q_i indica la demanda de cada cliente, $[r_i, d_i]$ su ventana de tiempo y s_i su tiempo de servicio. Se define también $q_{v_s} = q_{v_e} = s_{v_s} = s_{v_e} = 0$.

Además, el tráfico y la congestión se modela de la siguiente manera. Siguiendo el modelo propuesto por Ichoua et al. [16], definimos el horizonte de tiempo $H = \left[\underline{H}, \bar{H}\right]$ en el cuál se ejecutan las operaciones. Definimos $H_1, H_2, \ldots, H_h$ una partición de H, donde llamamos a cada H_k un *Speed Zone*, que son aquellos intervalos de tiempo dónde cambia la velocidad promedio de viaje. Además se utiliza la notación $\underline{H}_k, \bar{H}_k$ para indicar el extremo inferior y superior respectivamente de cada intervalo H_k.

Sea C una partición de los ejes, llamamos cluster a cada conjunto de ella. Para cada cluster c e intervalo H_k, v_{ck} representa la velocidad promedio a la que viaja un vehículo por cualquier eje de c durante el intervalo H_k. En este trabajo asumimos $v_{ck} > 0 \forall c, k$ por una cuestión de simplicidad en las cuentas, pero puede ser extendido fácilmente para $v_{ck} = 0$ también. Adicionalmente, se denota v_{ij}^k como la velocidad del cluster del eje (i, j) en el Speed Zone k. Estas velocidades son utilizadas para calcular los tiempos de viaje, lo cuál es explicado en detalle en la Sección 3.2.

El TD-VRPRC consiste en determinar un conjunto de a lo sumo K rutas, visitando a cada cliente exactamente una vez de forma tal de minimizar la duración total de las mismas.

3.2. Análisis de la dependencia del tiempo

Para lograr modelar problemas dependientes del tiempo se necesita definir dos funciones, analizar cómo computarlas y sus propiedades. Estas dos funciones son τ y ρ, la función de tiempo de viaje y tiempo de fin de recorrido (o Ready Time Function) respectivamente. La primera indica dado un eje (i, j) y un tiempo t en qué instante se llega a j si se parte de i en t y se atraviesa el eje (i, j). Por otro lado, ρ indica dado un camino

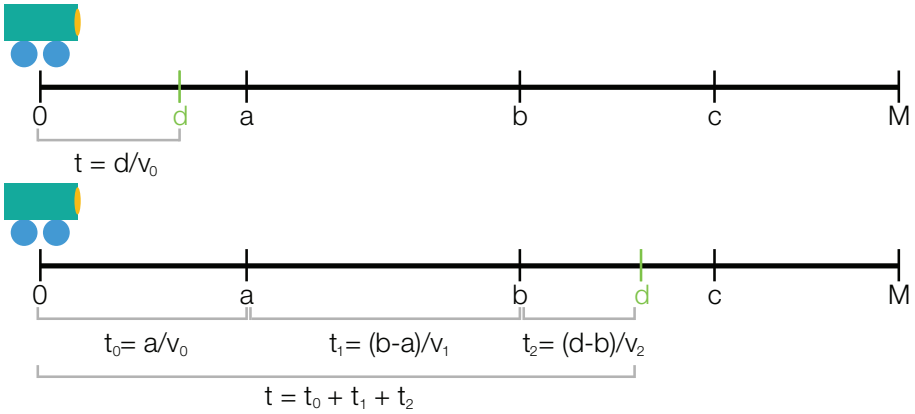


Fig. 3.1: Ejemplos del cálculo de tiempo de viaje.

P y un instante t, en qué momento se termina de recorrer todo el camino (tiempos de servicio incluidos) partiendo en t. A continuación se desarrolla cada una de las funciones, se da una caracterización de las mismas y se propone el pseudocódigo de un algoritmo para computarlas.

3.2.1. Tiempos de viaje

Calcular los tiempos de viaje en un modelo independiente del tiempo es simple dado que únicamente depende de la distancia. Sin embargo, en el modelo presentado en esta sección también se considera el momento en el cuál se atraviesa el eje, lo cuál determina la velocidad en que se lo transita y por otra parte, el modelo también permite reajustar esa velocidad a lo largo del viaje en caso de ser necesario.

Cómputo de tiempo de viaje Para computar esta función se utiliza el algoritmo propuesto por Ichoua et al. [16], que se detalla en el Algoritmo 2. Si se tiene que recorrer el eje (i, j) cuya distancia total es d_{ij} partiendo en t_0, la idea es calcular qué distancia se viaja en cada intervalo H_k y luego utilizando las fórmulas físicas de Movimiento Rectilíneo Uniforme (MRU) deducir cuál es el tiempo que esto lleva. En la Figura 3.1 se muestran dos ejemplos donde hay 4 Speed Zones: en el primero se requiere uno solo para recorrer el eje y en el segundo se debe pasar por 3 distintos cambiando de velocidad al cruzar las fronteras de cada uno de ellos.

Suponiendo que el vehículo comienza en el Speed Zone 1, la letra a indica la distancia que se debe atravesar para pasar al Speed Zone 2, la b para pasar al Speed Zone 3, y así sucesivamente. Esta distancia se puede calcular fácilmente utilizando la fórmula $d = v * t$ dado que se sabe para cada Speed Zone su velocidad y se sabe cuánto tiempo dura. Luego, hay que determinar por cuales Speed Zone hay que atravesar para recorrer d, y por último calcular el tiempo de viaje en cada uno de ellos.

Habiendo visto esto, se presenta el Algoritmo 2 que implementa la idea mencionada anteriormente. Su complejidad es lineal en la cantidad de Speed Zones.

Algorithm 2 Calculo de tiempo de viaje en instancia time-dependent.
Complejidad: $O(h)$

```
1:  function TravelTime((i, j), t_0)
2:      if t_0 ∉ H then                                   ▷ O(1)
3:          return INFACTIBLE
4:      d ← d_ij                                          ▷ O(1)
5:      t ← t_0                                           ▷ O(1)
6:      for m = 1 ... h do                                ▷ O(h)
7:          if d = 0 then
8:              break
9:          if t ∈ H_m then                               ▷ O(1)
10:             d' ← mín(d, v_ij^m (H̄_m − t))             ▷ O(1)
11:             t' ← d'/v_ij^m                            ▷ O(1)
12:             d ← d − d'                                ▷ O(1)
13:             t ← t + t'                                ▷ O(1)
14:     if d > 0 then                                     ▷ O(1)
15:         return INFACTIBLE
16:     return t
```

Cómputo de tiempo de partida El tiempo de viaje dice cuánto nos tarda ir de i a j partiendo en t_0. Sin embargo, a veces es necesario transitar el camino inverso, es decir, si queremos ir de i a j llegando en t_f, ¿Cuándo debemos salir?

A esta función la llamamos tiempo de partida $\widetilde{\tau}_{ij}(t)$, y la manera de computarla es análoga a la función de tiempo de viaje, pero en lugar de ir desde el Speed Zone 1 hacia el h, se recorre el horizonte de tiempo de manera inversa. En el Algoritmo 3 se detalla una posible implementación de esta función cuya complejidad es lineal en función de la cantidad de Speed Zones.

Cómputo de Travel Time Breakpoints Se puede probar que la función τ_{ij} es una Función Lineal a Trozos (FLT) para cada eje (i, j). Aún más, se puede definir exactamente en qué puntos la función cambia de pendiente. A estos puntos los llamamos Travel Time Breakpoints (TTB) y los escribimos como w_{ij}^m, y a cada intervalo formado por pares de breakpoints consecutivos se los llama $T_{ij}^m = \left[w_{ij}^m, w_{ij}^{m+1}\right]$. Además se llama W_{ij} al conjunto de todos los w_{ij}^m para el arco (i, j) agregados según el m.

El conjunto de todos los breakpoints está compuesto por aquellos momentos $\bar{H}_k$ en dónde cambia la velocidad, y sus respectivos $\widetilde{\tau}_{ij}(\bar{H}_k)$ cuando estén definidos, ya que salir después de estos puntos implica un nuevo cambio de velocidad durante el recorrido. Formalmente,

$$W_{ij} = \{H_s\} \cup \{H_e\} \cup \bigcup_{k=1}^{h} \left\{\bar{H}_k, \widetilde{\tau}_{ij}(\bar{H}_k)\right\} \tag{3.1}$$

Un ejemplo de ambas funciones se describe en las Figuras 3.2a y 3.2b donde se grafican las funciones v_{ij} y τ_{ij} respectivamente. Para este ejemplo se considera que la distancia del eje (i, j) es $d_{ij} = 10$ y el horizonte de tiempo está determinado por la siguiente tabla:

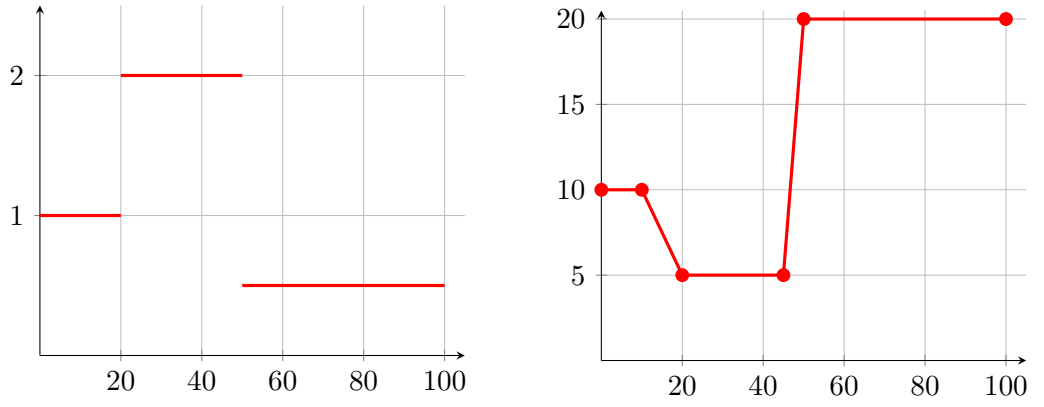
Algorithm 3 Cálculo del momento de partida para terminar de recorrer un eje en un instante t_f.
Complejidad: $\mathcal{O}(h)$

```
1:  function DepartingTime((i, j), t_f)
2:      if t_f ∉ H then                                   ▷ O(1)
3:          return INFACTIBLE
4:      d ← d_ij                                          ▷ O(1)
5:      t ← t_f                                           ▷ O(1)
6:      for m = h ... 1 do                                ▷ O(|h|)
7:          if d = 0 then
8:              break
9:          if t ∈ H_m then                               ▷ O(1)
10:             d' ← mín(d, v_ij^m (t − H_m))             ▷ O(1)
11:             t' ← d'/v_ij^m                            ▷ O(1)
12:             d ← d − d'                                ▷ O(1)
13:             t ← t − t'                                ▷ O(1)
14:     if d > 0 then                                     ▷ O(1)
15:         return INFACTIBLE
16:     return t
```

H	[0, 20]	[20, 50]	[50,100]
v_{ij}	1	2	0.5

En este ejemplo, los puntos donde cambia la velocidad son $a = 20, b = 50$, luego $\widetilde{\tau}_{ij}(a) = 10, \widetilde{\tau}_{ij}(b) = 45$, y por lo tanto los breakpoints son $W_{ij} = [0, 10, 20, 45, 50, 100]$.

Para computar los TTB dado un eje (i, j) se utiliza el Algoritmo 4. Una observación importante es que para algún momento $\bar{H}_k$ donde cambia la velocidad puede no estar definido su correspondiente $\widetilde{\tau}_{ij}(\bar{H}_k)$ si este ocurre antes del comienzo del horizonte y en dicho caso no es necesario agregarlo.



(a) Ejemplo de función de velocidades

(b) Ejemplo de función de tiempo de viaje

Fig. 3.2: Funciones de viaje en problemas con velocidades dependientes del tiempo

Luego de este análisis podemos reescribir las funciones τ_{ij} llamando θ_{ij}^m y η_{ij}^m a la pendiente y ordenada al origen de la función lineal que se forma en τ_{ij} en el intervalo T_{ij}^m, como se describe a continuación.

$$\tau_{ij}(t) = \begin{cases} \theta_{ij}^0 t + \eta_{ij}^0 & \text{si } t \in T_{ij}^0 \\ \theta_{ij}^1 t + \eta_{ij}^1 & \text{si } t \in T_{ij}^1 \\ \dots & \\ \theta_{ij}^{|T_{ij}|-1} t + \eta_{ij}^{|T_{ij}|-1} & \text{si } t \in T_{ij}^{|T_{ij}|-1} \end{cases} \tag{3.2}$$

Algorithm 4 Calculo de Travel Time Breakpoints de un eje (i, j).
Complejidad: $\mathcal{O}\left(h^2\right)$

```
1:  function TravelTimeBreakpoints((i, j))
2:      S ← [H_s]                                   ▷ O(1)
3:      S' ← []                                     ▷ O(1)
4:      for m = 1 ... h do                          ▷ O(h^2)
5:          x ← H̄_m                                 ▷ O(1)
6:          x' ← τ̃_ij(x)                            ▷ O(h)
7:          S ← S · x                               ▷ O(1)
8:          if x' ≠ INFACTIBLE then                 ▷ O(1)
9:              S' ← S' · x'                        ▷ O(1)
10:     return MergeSinDuplicados(S, S')            ▷ O(h)
```

3.2.2. Ready Time Function

Otra función importante es la denominada Ready Time Function (RTF) que la escribimos como $\rho_P(t)$. Esta función indica dado un camino y un tiempo de partida cuando se termina de recorrer todo el camino incluyendo tiempos de servicio.

Definición Sea $P = (v_1, v_2, \dots, v_k)$ un camino que empieza en v_1 y termina en v_k y $\hat{P}$ el camino P sin su último nodo, v_k. La función $\rho_P(t)$ se puede definir recursivamente considerando cuánto tarda en recorrerse todo el camino menos el último vértice, y a eso sumarle el tiempo de transitar el último eje y servir a ese último cliente. La definición formal se muestra a continuación.

$$\rho_P(t) = \begin{cases} \max\left(r_{v_k}, \rho_{\hat{P}}(t)\right) + \tau_{v_{k-1}v_k}(\max(r_{v_k}, \rho_{\hat{P}}(t))) + s_{v_k} & \text{si } |P| > 1 \\ r_{v_k} + s_{v_k} & \text{si } |P| = 1 \end{cases} \tag{3.3}$$

Es importante notar que esta función se encuentra definida para un dominio que consiste en aquellos instantes de partida t tales que recorrer P es factible. Este dominio se define de la siguiente manera:

$$dom(\rho_P) = \begin{cases} \left[\underline{H}, \max\limits_{t \in dom(\rho_{\hat{P}})} \left\{\rho_{\hat{P}}(t) + \tau_{v_{k-1}v_k}(\rho_{\hat{P}}(t)) \leq d_v\right\}\right] & \text{si } |P| > 1 \\ [\underline{H}, d_v] & \text{si } |P| = 1 \end{cases} \tag{3.4}$$

Cómputo de la función ρ La función ρ_P es sencilla de computar y una posible implementación se exhibe en el Algoritmo 5 con una complejidad de $\mathcal{O}(h|P|)$. La idea es ir simulando el viaje que transita el vehículo y tomar el tiempo final como resultado.

Algorithm 5 Calculo de Ready Time.
Complejidad: $\mathcal{O}(h|P|)$

```
1:  function ReadyTime(P, t_0)
2:      t' ← máx(r_{P_1}, t_0) + s_{P_1}                    ▷ O(1)
3:      for k ← 1 ... |P| − 1 do                            ▷ O(h|P|)
4:          i ← P_k, j ← P_{k+1}                            ▷ O(1)
5:          tt ← τ_ij(t')                                   ▷ O(h)
6:          if tt = INFACTIBLE then                         ▷ O(1)
7:              return INFACTIBLE                           ▷ O(1)
8:          t' ← máx(r_j, t' + tt)                          ▷ O(1)
9:          if t' > d_j then                                ▷ O(1)
10:             return INFACTIBLE                           ▷ O(1)
11:         t' ← t' + s_j                                   ▷ O(1)
12:     if t' ∉ H then                                      ▷ O(1)
13:         return INFACTIBLE                               ▷ O(1)
14:     return t'                                           ▷ O(1)
```

3.3. Notación adicional

3.3.1. Intervalo de llegada

En primer lugar, se define $\widetilde{w}_{ij}^m$ como el momento en que se termina de servir al cliente j si se parte de i en w_{ij}^m y se atraviesa el eje (i, j). Formalmente:

$$\widetilde{w}_{ij}^m = \text{máx}(r_j, w_{ij}^m + \tau_{ij}(w_{ij}^m)) + s_j \tag{3.5}$$

Usando esta definición, por ejemplo, se puede especificar que sabiendo que se parte de un cliente i en algún momento de $\left[w_{ij}^m, w_{ij}^{m+1}\right]$, seguro se termina de satisfacer a el siguiente cliente j en algún instante del intervalo $\left[\widetilde{w}_{ij}^m, \widetilde{w}_{ij}^{m+1}\right]$.

3.3.2. Recorrido *no-wait*

Llamamos *Recorrido* a una manera de atravesar un camino P en el tiempo. Por ejemplo, el camino $P = (1, 2, 3)$ puede tener distintos recorridos. Entre ellos R_1 =((1, 2, 100.0), (2, 3, 150.50)), es decir, el recorrido que atraviesa el eje $(1, 2)$ partiendo en el instante 100.0 y que pasa por el eje $(2, 3)$ partiendo en el instante 150.50.

De manera similar, se define *Camino Time-Dependent* que es una manera más relajada de indicar un Recorrido, especificando en qué intervalo se atraviesa cada eje en lugar del instante exacto. Por ejemplo, $((1, 2, m_1), (2, 3, m_2))$ es un recorrido de P, donde se sabe que se parte del eje $(1, 2)$ en $T_{12}^{m_1}$, y de $(2, 3)$ en $T_{23}^{m_2}$.

Se llama $\Delta(R)$ a la duración del recorrido R, es decir, a la diferencia entre el momento en que se termina y el momento en que se parte. Se llama $\Delta(P)$ a la duración del camino P, o sea, a la duración mínima entre todos sus recorridos.

Uno de los conceptos que es interesante de analizar es lo que denominamos la propiedad *no-wait* que puede tener un recorrido. Decimos que un recorrido de P es *no-wait* si en ningún momento se detiene el vehículo salvo por el tiempo de servicio o la espera en un vértice debido a que se arribó antes de su ventana de tiempo. Por ejemplo, un camino donde se termina de servir al nodo i en t y se parte en $t+\epsilon$ no cumple con la propiedad *no-wait*.

Una observación es que dado cualquier recorrido R que no cumple *no-wait*, existe otro R' que sí lo cumple y vale que $\Delta(R) \geq \Delta(R')$. Esto se cumple porque el modelo de tiempos de viaje cumple con la propiedad FIFO a nivel eje, entonces esperar nunca puede reducir la duración del camino. Sin embargo, sí puede reducir el tiempo total de viaje, por lo cuál dependiendo de la función objetivo puede tener sentido restringirse únicamente a soluciones que cumplan la propiedad *no-wait*.

3.3.3. Ejes dependientes del tiempo

Como se describe anteriormente, una manera de representar recorridos con menor granularidad es indicar en qué time zone se recorre cada eje. Esto se hace utilizando triplas (i, j, m) donde $(i, j) \in E$ es el eje y $m \in [0 \dots |T_{ij}|)$ el intervalo en el cuál se comienza a recorrer. Para simplificar nomenclatura se los llamará simplemente ejes y la dependencia del tiempo se asumirá implícita en el contexto.

Dado una instancia con velocidades dependientes del tiempo, se llama $\mathbb{X}$ al conjunto de todos sus ejes dependientes del tiempo. Formalmente:

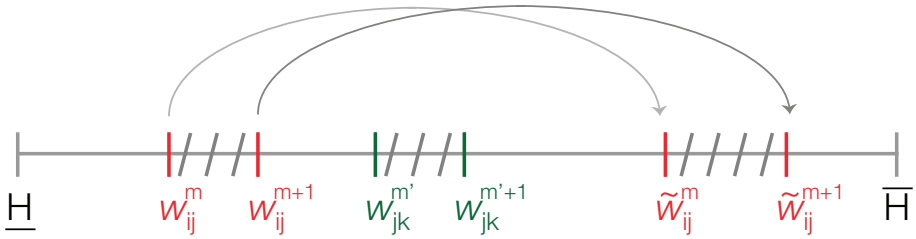
$$\mathbb{X} = \{(i, j, m) \mid (i, j) \in E, m \in [0 \dots |T_{ij}|)\} \tag{3.6}$$

Habiendo definido esto, se introduce la noción de eje sucesor. Decimos que (j, k, m') es sucesor de (i, j, m) si se puede comenzar a recorrer (i, j) en T_{ij}^{m}, de manera tal de luego poder partir de j en $T_{jk}^{m'}$. Formalmente, decimos que (j, k, m') es sucesor de (i, j, m) si $\widetilde{w}_{ij}^{m} \leq w_{jk}^{m'+1}$.

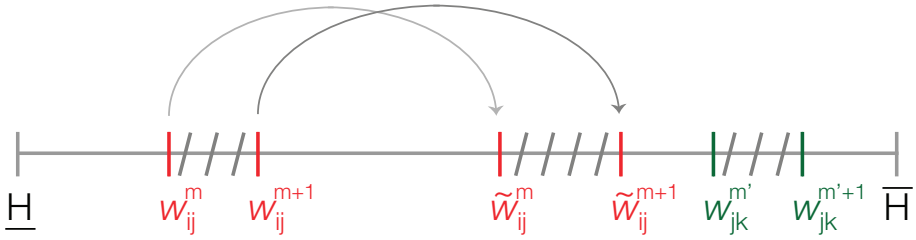
Por otro lado, se da una definición más estricta de sucesión llamada sucesor *no-wait*. Se dice que (j, k, m') es sucesor *no-wait* de (i, j, m) si es sucesor y además ambos pueden aparecer en un recorrido *no-wait*. Formalizando, (j, k, m') es sucesor *no-wait* de (i, j, m) si $\left[\widetilde{w}_{ij}^{m}, \widetilde{w}_{ij}^{m+1}\right] \cap \left[w_{jk}^{m'}, w_{jk}^{m'+1}\right] \neq \emptyset$.

Formalmente, se define dado un conjunto de ejes $X \subseteq \mathbb{X}$ el conjunto $A^+(X)$ de ejes sucesores, es decir, aquellos ejes que son sucesores de alguno de los ejes de X pero no pertenecen a ese conjunto. Además, se define $\widetilde{A}^+(X)$ del mismo modo pero solamente considerando sucesores *no-wait*. Se permite además hacer abuso de notación y utilizar la función para referirse a los sucesores de un solo eje ($A^+((i, j, m)) = A^+(\{(i, j, m)\})$, y $\widetilde{A}^+((i, j, m)) = \widetilde{A}^+(\{(i, j, m)\})$

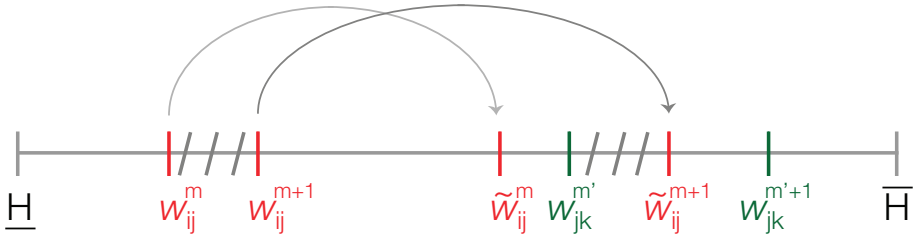
$$A^+(X) = \left\{(j, k, m') \in \mathbb{X} - X \mid \exists (i, j, m) \in X, \widetilde{w}_{ij}^{m} \leq w_{jk}^{m'+1}\right\} \tag{3.7}$$



(a) Ejemplo de eje no sucesor.



(b) Ejemplo de eje sucesor con tiempo de espera.



(c) Ejemplo de eje sucesor *no-wait*.

Fig. 3.3: Ejemplos del concepto de eje sucesor y sucesor *no-wait*.

$$\widetilde{A}^{+}(X) = \left\{(j,k,m') \in \mathbb{X} - X \mid \exists (i,j,m) \in X, \left[\widetilde{w}_{ij}^{m}, \widetilde{w}_{ij}^{m+1}\right] \cap \left[w_{jk}^{m'}, w_{jk}^{m'+1}\right] \neq \emptyset\right\} \quad (3.8)$$

En las Figuras 3.3 se pueden ver tres ejemplos de comparación de ejes. En la Figura 3.3a (j,k,m') no es sucesor de (i,j,m) porque tomar el eje (i,j,m) implica terminar de servir al cliente j al menos en el instante $\widetilde{w}_{ij}^{m}$ que es mayor que el último momento en que se puede tomar el eje (j,k,m').
En la Figura 3.3b se exhibe un ejemplo donde sí es sucesor, pero no cumple *no-wait*, dado que se debe esperar luego de servir a j para poder salir en m'. En la Figura 3.3c se muestra un caso dónde sí sucede que un eje es sucesor *no-wait* de otro eje.

3.4. TD-PESPPRC

Un problema que está relacionado con el TD-VRPRC es el denominado TD-PESPPRC. A continuación se define el TD-PESPPRC y luego se presenta cuál es la relación entre ambos problemas.

3.4.1. Definición

La definición del problema TD-PESPPRC es similar a TD-VRPRC pero con algunas modificaciones. La primera es que v_s y v_e en el contexto de camino mínimo representan

al nodo origen y destino. Además, dado que se considera un único vehículo el parámetro K no es necesario. Por otra parte, se cuenta con p_i que indica para cada cliente cuál es el beneficio de visitarlo. Dado que el camino debe visitar obligatoriamente a los vértices v_s y v_e, se define $p_{v_s} = p_{v_e} = 0$.

Además, al tener beneficios por cada cliente visitado, la función objetivo varía levemente. En este caso, lo que se intenta es lograr un compromiso entre el beneficio obtenido por el recorrido y su duración. Formalmente, se quiere encontrar el recorrido r que minimice $\Delta(r) - \sum_{i \in r} p_i$.

En el caso de las restricciones de recursos y el modelo de tiempos de viaje se utiliza las mismas definiciones que en el caso de TD-VRPRC.

3.4.2. Relación entre TD-PESPPRC y TD-VRPRC

Una manera de resolver instancias de TD-VRP es utilizar PLEM. En particular, un tipo de formulación que se suele usar para modelar este problema es conocida como *Set-Partitioning*. Para más detalles de esta formulación ver Toth y Vigo [31].

Dada una instancia de TD-VRPRC se toma el conjunto Ω de todos los caminos que empiezan y terminan en el depósito y que son factibles, es decir, que cumplen con las restricciones operacionales establecidas por los recursos, que visitan cada vértice a lo sumo una vez y que se pueden realizar en el horizonte de planificación. Además para cada vértice i y ruta r se define a_{ir} que indica si se visita el vértice i en la ruta r. Por último se definen las variables λ_r que indican si se utiliza la ruta r en la solución. A continuación se presenta el modelo.

$$\textbf{min} \sum_{r \in \Omega} c_r \lambda_r \tag{3.9}$$

sujeto a

$$\sum_{r \in \Omega} a_{ir} \lambda_r = 1 \quad \forall i \in V \setminus \{v_s, v_e\} \tag{3.10}$$

$$\lambda_r \in \{0, 1\} \quad \forall r \in \Omega \tag{3.11}$$

La función objetivo (3.9) minimiza la suma de los costos de cada ruta c_r, donde el costo puede ser la duración, el tiempo de viaje, etc. Las restricciones (3.10) establecen que cada cliente debe ser visitado exactamente una vez. Además (3.11) son las restricciones de integralidad para las variables de decisión.

Debido a que el número de caminos en Ω suele ser exponencial, este problema se aborda utilizando la técnica de Branch&Price, y por lo tanto cada relajación lineal se resuelve utilizando Generación de Columnas.

Sea $p_i, \forall i \in V \setminus \{v_s, v_e\}$ las variables duales asociadas a las restricciones (3.10). El costo reducido de una variable λ_r está definido como

$$\bar{c}_r = c_r - \sum_{i \in V \setminus \{v_s, v_e\}} a_{ir} p_i = \Delta(r) - \sum_{i \in V \setminus \{v_s, v_e\}} a_{ir} p_i \tag{3.12}$$

En una iteración de Simplex, una variable λ_r entra a la base si su costo reducido $\bar{c}_r$ es negativo, por lo tanto encontrando cualquier recorrido que cumpla esa condición se puede seguir iterando. En particular, para definir si existe alguna variable que pueda entrar a la base se puede encontrar aquella con menor costo reducido y verificar si este es negativo. En caso afirmativo se agrega a la base y en caso negativo se puede afirmar que se alcanzó una solución óptima de la relajación lineal.

Como minimizar $\bar{c}_r$ es exactamente la función objetivo del problema PESPP entonces se puede usar de problema esclavo. La única salvedad es que buscamos caminos r que sean factibles, y por lo tanto se debe agregar estas restricciones de factibilidad, haciendo que el problema que hay que resolver sea finalmente el TD-PESPPRC donde los recursos a considerar son exactamente aquellos que se consideran en el VRPRC.

Para mayor detalles sobre esta formulación o la relación entre estos problemas se puede consultar [6, 31].

4. FORMULACIÓN PARA TD-PESPPRC

En esta sección se presenta un PLEM para resolver el problema TD-PESPPRC. Luego, se estudia el poliedro P_{TD} de la formulación y se identifican desigualdades válidas que finalmente son incorporadas a un algoritmo de plano de corte.

Este trabajo junto con sus desarrollos se pueden mostrar válidos para instancias con recursos tanto de capacidad como ventanas de tiempo, sin embargo de aquí en adelante la experimentación solamente tiene en cuenta los recursos del primer tipo. Las ventanas de tiempo se dejan fuera de la experimentación debido a que existen varios algoritmos basados en otro tipo de técnicas que tienen un buen funcionamiento en instancias con este tipo de restricciones (ver e.g. Dabia et al. [6]) pero no sin ellas.

4.1. Modelo

El modelo que se propone a continuación es una adaptación del propuesto en Sun et al. [29]. Se utilizan 5 conjuntos de variables distintas. La variable x_{ij}^m es binaria, y está en 1 únicamente si el vehículo recorre el eje (i, j) partiendo en algún instante del time zone T_{ij}^m. A su vez, x_{ij} está en 1 únicamente si se recorre el eje (i, j) independientemente del momento en el que se realice el viaje. La variable y_i toma el valor 1 sí y solo sí el nodo i se visita en el camino. Estas últimas dos variables, x_{ij} e y_i, sintetizan la información de x_{ij}^m. Formalmente:

$$x_{ij} = \sum_{m=0}^{|T_{ij}|-1} x_{ij}^m$$

$$y_i = \sum_{(i,j)\in\delta^+(i)} x_{ij}$$

Por otro lado, las variables t_i, t_{ij}^m modelan los tiempos de salida de los nodos y por lo tanto pertenecen al dominio de los reales. Esto es importante incorporarlo al modelo en problemas con tiempos de viaje variable porque conocer los tiempos exactos de salida permite calcular, de manera precisa, el tiempo de viaje. La variable t_i indica el tiempo en que se terminó de servir al cliente i en caso de ser visitado ($y_i = 1$), caso contrario está en 0. La variable t_{ij}^m toma el valor de t_i si el eje (i, j) fue visitado durante el time zone m (o sea, $x_{ij}^m = 1$), y toma el valor 0 en caso contrario.

$$t_{ij}^m = \begin{cases} t_i & \text{si } x_{ij}^m = 1 \\ 0 & \text{caso contrario} \end{cases} \tag{4.1}$$

Tal como se explica en la Sección 3.2.1, las funciones τ_{ij} son lineales a trozos, y en particular son lineales en cada T_{ij}^m, permitiendo escribirlas en función de sus pendientes θ y ordenada al origen η como se detalla en la ecuación (3.2).

Esta observación, combinada con la información que representan las variables x_{ij}^m, es decir, en qué time zone se comienza a recorrer cada eje, permiten escribir linealmente las restricciones que representan el tiempo de viaje en nuestro modelo.

$$\text{mín } t_{v_e} - t_{v_s} - \sum_{i \in V \setminus \{v_s, v_e\}} p_i y_i \tag{4.2}$$

sujeto a

$$x_{ij} = \sum_{m=0}^{|T_{ij}^m|} x_{ij}^m \qquad \forall (i,j) \in E \tag{4.3}$$

$$\sum_{i \in \delta^-(j)} x_{ij} = y_j \qquad \forall j \in V \setminus \{v_s\} \tag{4.4}$$

$$\sum_{j \in \delta^+(v_s)} x_{v_s,j} = 1 \tag{4.5}$$

$$\sum_{i \in \delta^-(v_e)} x_{i,v_e} = 1 \tag{4.6}$$

$$\sum_{i \in \delta^-(k)} x_{ik} - \sum_{j \in \delta^+(k)} x_{kj} = 0 \qquad \forall k \in V \setminus \{v_s, v_e\} \tag{4.7}$$

$$t_j = \sum_{i \in \delta^-(j)} \sum_{m=0}^{|T_{ij}|-1} (1 + \theta_{ij}^m) t_{ij}^m + \eta_{ij}^m x_{ij}^m + s_j x_{ij}^m \qquad \forall j \in V \setminus \{v_s\} \tag{4.8}$$

$$t_i = \sum_{j \in \delta^+(i)} \sum_{m=0}^{|T_{ij}|-1} t_{ij}^m \qquad \forall i \in V \setminus \{v_e\} \tag{4.9}$$

$$w_{ij}^m x_{ij}^m \leq t_{ij}^m \leq w_{ij}^{m+1} x_{ij}^m \qquad \forall (i,j) \in E, 0 \leq m < |T_{ij}| \tag{4.10}$$

$$\sum_{i \in V \setminus \{v_s, v_e\}} q_i y_i \leq Q \tag{4.11}$$

$$t_{ij}^m, t_i \geq 0 \qquad \forall (i,j) \in E, 0 \leq m < |T_{ij}| \tag{4.12}$$

$$x_{ij}^m, x_{ij}, y_i \in \{0,1\} \qquad \forall (i,j) \in E, 0 \leq m < |T_{ij}| \tag{4.13}$$

La función objetivo (4.2) busca balancear la duración del camino contra el beneficio obtenido por visitar a los clientes del mismo. Para esto resta el beneficio a la duración del camino, logrando así que al aumentar el beneficio o reducir la duración la función objetivo decrezca. Las restricciones (4.3)-(4.4) relacionan las variables x_{ij}, y_i tal como se explicó previamente. Su función además es garantizar que a lo sumo una x_{ij}^m tome el valor 1 para cada j. Las restricciones (4.5)-(4.6) establecen que se sale del depósito de inicio y se llega al de fin. Las desigualdades (4.7) modelan la conservación de flujo, que establece que la cantidad de veces que se entra a un nodo es igual al número de veces que se sale de él, para todos los nodos que no son depósito. En particular, como solamente se puede entrar y salir una vez de cada nodo, estas desigualdades nos dicen que si se entra al nodo se debe salir, y si no se entra no se puede salir.
Las restricciones (4.8) son las que ajustan el ready time de cada nodo y la forma en que lo hacen es considerando lo explicado en la ecuación (3.2). Notar que esta restricción es una igualdad y debería ser cambiada por un $\geq$ en caso de tener ventanas de tiempo para mo-

delar correctamente los tiempos de espera previos al procesamiento del cliente. Además, para asegurarse que las t_{ij}^m sean consistentes con los x_{ij}^m y los t_i, se utilizan las restricciones (4.9)-(4.10). Por otra parte, la restricción (4.11) es necesaria para asegurarse que la capacidad no se exceda. En último lugar, la restricción (4.12) establece que los ready time no deben ser negativos, porque el horizonte de tiempo no es negativo, y además las desigualdades (4.13) exigen que las variables de decisión sean binarias.

Las restricciones (4.8) son una versión reforzada de las propuestas por Sun et al. [29] dado que allí se agrega una restricción por cada (i, j, m) mientras que en esta formulación se agrega la suma de todas aquellas restricciones que se refieran a un mismo vértice de llegada j. Esto se puede hacer dado que por cada j, cuándo la solución es entera, existe a lo sumo un único x_{ij}^m y a lo sumo un único t_{ij}^m con valor positivo. En experimentos informales notamos una mejora considerable en la calidad de la relajación con un impacto positivo en términos de performance del algoritmo B&C.

4.2. Heurística inicial

Para poder partir desde una solución inicial y brindarle una cota superior al algoritmo de Branch&Bound se diseña una heurística combinando una fase constructiva golosa con una posterior búsqueda local.

En primer lugar se obtiene una solución golosa detallada en el Algoritmo 6 donde se parte del camino $S = [v_s, v_e]$ y se agrega de a un vértice en la ante última posición de manera tal de ir minimizando el costo de la solución parcial.

Por ejemplo, si $V = \{v_s, v_e, 1, 2, 3\}$, en la primera iteración se evalúa $S_1 = [v_s, 1, v_e], S_2 = [v_s, 2, v_e], S_3 = [v_s, 3, v_e]$ y se selecciona el camino que sea factible y que minimice el costo. Suponiendo que ese camino es S_1, entonces ahora $S = S_1$ y luego se elige entre $S_1' = [v_s, 1, 2, v_e]$ y $S_2' = [v_s, 1, 3, v_e]$, y así sucesivamente hasta no poder mejorar el costo.

Algorithm 6 Algoritmo Goloso heurístico. Complejidad: $\mathcal{O}\left(|V|^2(F+T)\right)$ con F el costo computacional de verificar si un camino es factible y T el de calcular el costo reducido de un camino.

```
1:  function HeuristicaGoloso
2:      S* ← [v_s, v_e]                                          ▷ O(1)
3:      while S* mejora do                                       ▷ O(|V|^2(F+T))
4:          S ← S*                                               ▷ O(1)
5:          for i ∈ V \ S do                                     ▷ O(|V|(F+T))
6:              S' ← S_{1...|S|-1} · [i, S_{|S|}]                ▷ O(1)
7:              if S' es factible y cr(S') < cr(S) then          ▷ O(F+T)
8:                  S ← S'                                       ▷ O(1)
9:          S* ← S                                               ▷ O(1)
10:     return S*                                                ▷ O(1)
```

Luego utilizando el mejor camino encontrado S de la heurística, se utilizan dos vecin-

darios distintos para ir moviéndose en el espacio de soluciones utilizando Búsqueda Local, tal como se especifica en el Algoritmo 7.

- *Intercambio de nodos:* Dos soluciones son vecinas si solamente tienen un par de nodos intercambiados de posición.
  Ejemplo: $[1, 2, 3, 4, 5] -> [1, \mathbf{4}, 3, \mathbf{2}, 5]$
- *Entra/Sale:* Dos soluciones son vecinas si tienen la misma cantidad de nodos y solamente tienen una posición con un nodo distinto que no está en la otra solución.
  Ejemplo: $[1, 2, 3, 4, 5] -> [1, \mathbf{6}, 3, 4, 5]$

Los operadores se aplican usando la política Best-accept, es decir buscando la mejor solución entre todas las del vecindario.

Algorithm 7 Algoritmo Búsqueda Local. Complejidad: $\mathcal{O}\left(U(|V|^2(F+T))\right)$ con U la cantidad de iteraciones de aplicación de operadores, F el costo computacional de verificar si un camino es factible y T el de calcular el costo reducido de un camino.

1: **function** HeuristicaBusquedaLocal(S_0)
2: $S^* \leftarrow S_0$ ▷ $\mathcal{O}(1)$
3: **while** S mejora **do** ▷ $\mathcal{O}\left(U(|V|^2(F+T))\right)$
4: $S \leftarrow S^*$ ▷ $\mathcal{O}(1)$
5: **for** $i, j \in [2, \ldots, |S|-1], i \neq j$ **do** ▷ $\mathcal{O}\left(|V|^2(F+T)\right)$
6: $S' \leftarrow \text{Swap}(S_i, S_j)$ ▷ $\mathcal{O}(1)$
7: **if** S' es factible y $cr(S') < cr(S)$ **then** ▷ $\mathcal{O}(F+T)$
8: $S \leftarrow S'$ ▷ $\mathcal{O}(1)$
9: **for** $i \in [2, \ldots, |S|-1], v \in V \setminus S$ **do** ▷ $\mathcal{O}\left(|V|^2(F+T)\right)$
10: $S' \leftarrow S$ ▷ $\mathcal{O}(1)$
11: $S'_i \leftarrow v$ ▷ $\mathcal{O}(1)$
12: **if** S' es factible y $cr(S') < cr(S)$ **then** ▷ $\mathcal{O}(F+T)$
13: $S \leftarrow S'$ ▷ $\mathcal{O}(1)$
14: $S^* \leftarrow S$ ▷ $\mathcal{O}(1)$
15: **return** S^* ▷ $\mathcal{O}(1)$

4.3. Planos de corte

A continuación, se presentan los planos de corte que se consideran en este trabajo.

Primero se presentan 2 planos de corte ya conocidos en la literatura y se muestra su aplicabilidad al problema. Después, se introducen 3 nuevas familias de desigualdades especiales para modelos con tiempos de viaje variable que contemplen las variables x_{ij}^m descriptas en la sección anterior y que apuntan a explotar las características particulares introducidas por la dependencia temporal en los tiempos de viaje. La idea principal es cortar posibles soluciones fraccionarias. Además, para las mismas, se demuestra validez y se las relaciona con desigualdades ya existentes en la literatura.

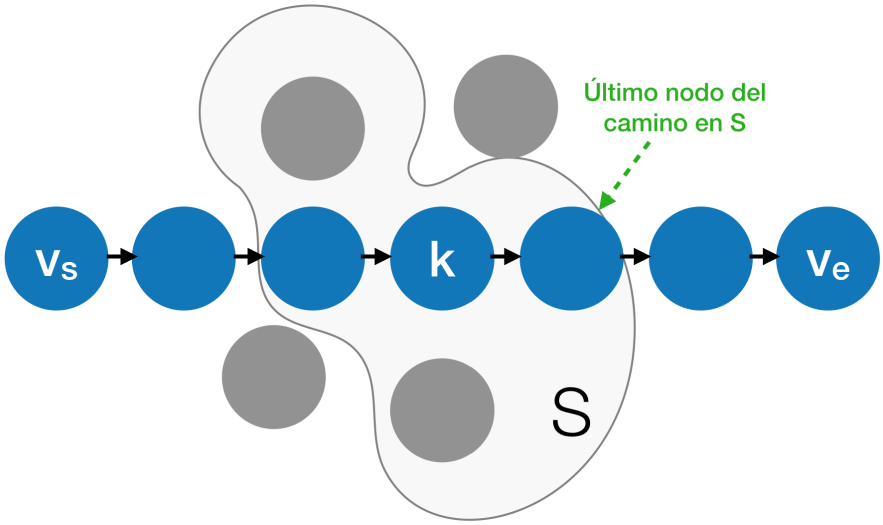


Fig. 4.1: Idea intuitiva de las desigualdades GCS cuando $y_k = 1$.

4.3.1. Generalized Cut Set (GCS)

La primera familia de desigualdades que se presenta es conocida con el nombre *Generalized Cut Set* (GCS) (ver e.g. Taccari [30]) y es una generalización de las Subtour Elimination Constraints (SEC) para el caso en que no necesariamente se visitan todos los vértices.

Dado un conjunto de nodos $S \subset V \setminus \{v_s, v_e\}$, $|S| \geq 2$ y un nodo $k \in S$.

$$\sum_{(i,j)\in\delta^+(S)} x_{ij} \geq y_k \tag{4.14}$$

Para entender qué significan las desigualdades es conveniente analizar dos casos. Cuando $y_k = 0$ la desigualdad es trivialmente válida porque el lado izquierdo vale al menos 0 dado que las variables x_{ij} no toman valores negativos.

Cuando $y_k = 1$ el camino visita el vértice k como se puede apreciar en la Figura 4.1 y las desigualdades son equivalentes a las SEC en su versión de conectividad. Por lo tanto, la solución parte desde v_s, pasa por k, y termina en v_e. Así es que tomando cualquier conjunto de nodos que no incluya a v_e, siempre vamos a tener alguno de ellos que sea el último de la solución que esté incluido, y por lo tanto, el siguiente no va a estarlo. De este modo se puede afirmar que siempre debe haber un eje, que es utilizado, que sale del conjunto.

Esta familia de desigualdades es válida para este problema ya que es una generalización del Problema de Camino Mínimo Elemental (ESPP) y las GCS son válidas allí.

4.3.2. Generalized Large Multistar (GLM)

Las desigualdades *Generalized Large Multistar* (GLM) (ver e.g. Jepsen y Pisinger [17]) son una generalización de las **knapsack-cover**. La idea es asegurarse que cada subcon-

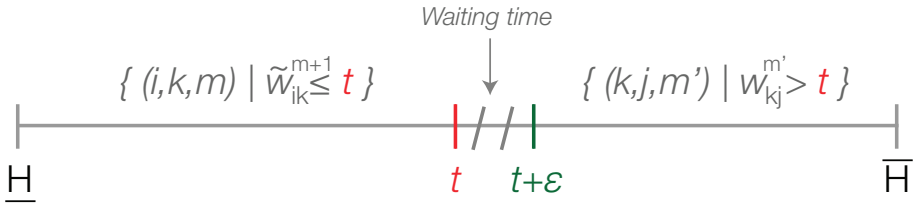


Fig. 4.2: Descripción gráfica de NWC.

junto $S \subseteq N$ sea visitado según la demanda de sus nodos.

$$\sum_{(i,j)\in\delta^+(S)} x_{ij} \geq \frac{1}{Q}\left(\sum_{i\in S} q_i y_i + \sum_{(i,j)\in\delta^+(S)} q_j(x_{ij}+x_{ji})\right) \tag{4.15}$$

Al igual que las GCS, son válidas para ESPP y por lo tanto también lo son para TD-PESPPRC ya que este problema es una generalización del primero.

4.3.3. No waiting cuts (NWC)

La siguiente desigualdad es nueva y explota la propiedad FIFO del modelo, es decir, que recorrer un camino P saliendo en un instante $t < t'$ garantiza terminarlo en $\rho_P(t) \leq \rho_P(t')$. Esto nos permite decir que existe una solución óptima, o sea, que minimiza la función objetivo del modelo (4.2), que recorre los nodos en orden $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ y que cumple la propiedad *no-wait*. Es decir, ni bien se termina de servir a un cliente, se sale hacia el próximo sin esperar.

Teniendo eso en mente, el primer corte que se introduce se llama *No waiting cuts* (NWC). El concepto de este corte es bastante intuitivo y se presenta en la Figura 4.2. La motivación es conseguir un conjunto de ejes que entren a un vértice k y otro que salgan de manera tal que los que entran y los que salen no tengan intersección alguna en el tiempo. Esto nos permite afirmar que no pueden ser usados un eje de cada grupo en la misma solución, dado que se tendría tiempo de espera y esto implica que existe otra solución mejor o igual en términos de duración simplemente ajustando ese tiempo, como se explicó en el capítulo anterior.

Es importante tener en cuenta que estos cortes tienen sentido si se intenta minimizar la duración total del viaje o su makespan, dado que si se intenta minimizar tiempos de viaje puede tener sentido esperar.

Proposición 4.3.1. *La familia de desigualdades NWC definida en (4.16) es válida para* P_{TD}.

$$\sum_{(i,k,m)\in\delta^-_{\leq t}(k)} x^m_{ik} + \sum_{(k,j,m)\in\delta^+_{>t}(k)} x^m_{kj} \leq 1, \quad \forall k \in V \setminus \{v_s, v_e\}, \forall t \in H \tag{4.16}$$

donde

$$\delta^-_{\leq t}(k) = \left\{(i,k,m) \in E \times \mathbb{N} \mid \widetilde{w}^{m+1}_{ik} \leq t\right\} \tag{4.17}$$

$$\delta^+_{>t}(k) = \left\{(k,j,m) \in E \times \mathbb{N} \mid w^m_{kj} > t\right\} \tag{4.18}$$

Como se puede observar estas desigualdades son infinitas ya que la cantidad de t distintos que se pueden elegir no está acotada. Sin embargo, se puede acotar a un conjunto finito de valores. La siguiente observación muestra cuál es este conjunto y demuestra por qué es válido solamente considerar dichas desigualadades.

Observación 4.3.2. *Sea* $\mathbb{T}_{NW}(k) = \left\{\widetilde{w}_{ik}^{m+1} \mid (i,k,m) \in E \times \mathbb{N}\right\}$ *un conjunto de instantes en el horizonte. La familia de desigualdades definida en la ecuación (4.19) domina a la familia definida en (4.16).*

$$\sum_{(i,k,m)\in\delta_{\leq t}^{-}(k)} x_{ik}^{m} + \sum_{(k,j,m)\in\delta_{>t}^{+}(k)} x_{kj}^{m} \leq 1, \quad \forall k \in V \setminus \{v_s, v_e\}, \forall t \in \mathbb{T}_{NW}(k) \tag{4.19}$$

Demostración. Sea un vértice $k \in V \setminus \{v_s, v_e\}$, y un instante de tiempo $t \in H$ dos valores cualquiera que definen una desigualdad de (4.16). Si encontramos un $t' \in \mathbb{T}_{NW}(k)$ tal que $\delta_{>t}^{+}(k) \subseteq \delta_{>t'}^{+}(k)$ y $\delta_{\leq t}^{-}(k) \subseteq \delta_{\leq t'}^{-}(k)$ entonces podemos afirmar que existe una desigualdad de (4.19) que domina a la anterior.

Tomamos $t' = \max\{t_0 \in \mathbb{T}_{NW}(k) \mid t_0 \leq t\}$. A continuación vemos que se cumplen las dos condiciones por separado.

Caso 1 ($\delta_{>t}^{+}(k) \subseteq \delta_{>t'}^{+}(k)$): Como $t' \leq t$ entonces cualquier eje (k,j,m) que cumple $w_{kj}^{m} > t$ también lo hace para $w_{kj}^{m} > t'$.

Caso 2 ($\delta_{\leq t}^{-}(k) \subseteq \delta_{\leq t'}^{-}(k)$): Supongamos que existe $(i,k,m) \in \delta_{\leq t}^{-}(k)$ que no pertenece a $\delta_{\leq t'}^{-}(k)$. Si eso sucede entonces quiere decir que $\widetilde{w}_{ik}^{m+1} \leq t$ pero $\widetilde{w}_{ik}^{m+1} > t'$, pero entonces por la Definición 4.17 $\widetilde{w}_{ik}^{m+1} \in \mathbb{T}_{NW}(k)$ y cumple que $\widetilde{w}_{ik}^{m+1} \leq t$, por lo tanto $t' = \widetilde{w}_{ik}^{m+1}$ lo cuál contradice la hipótesis de que no pertence a $\delta_{\leq t'}^{-}(k)$. Por lo tanto, queda demostrado que $\delta_{\leq t}^{-}(k) \subseteq \delta_{\leq t'}^{-}(k)$.

Como probamos que se cumplen $\delta_{>t}^{+}(k) \subseteq \delta_{>t'}^{+}(k)$ y $\delta_{\leq t}^{-}(k) \subseteq \delta_{\leq t'}^{-}(k)$ simultáneamente entonces podemos concluir que la observación es verdadera. □

Por otra parte, se puede reforzar las desigualdades cambiando el lado derecho por y_k ya que en los enteros esta toma valor 1 o 0. Se formaliza la desigualdad en la siguiente proposición y se demuestra su validez.

Proposición 4.3.3. *La familia de desigualdades (4.20) es válida para* P_{TD} *y domina a (4.19).*

$$\sum_{(i,k,m)\in\delta_{\leq t}^{-}(k)} x_{ik}^{m} + \sum_{(k,j,m)\in\delta_{>t}^{+}(k)} x_{kj}^{m} \leq y_k, \quad \forall k \in V \setminus \{v_s, v_e\}, \forall t \in \mathbb{T}_{NW}(k) \tag{4.20}$$

Demostración. El caso en donde $y_k = 0$ es válido trivialmente ya que ninguna variable de la izquierda puede estar prendida por las restricciones (4.3), (4.4) y (4.7). Veamos que vale también para el caso $y_k = 1$.

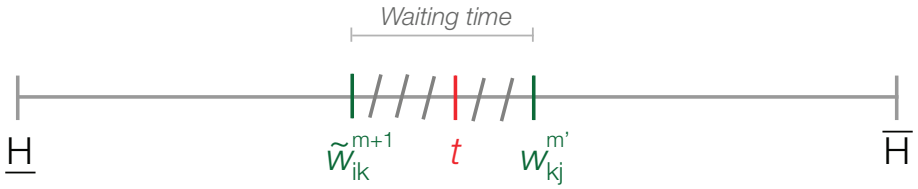


Fig. 4.3: Ejemplo para la demostración de la Proposición 4.3.3.

Sea $\bar{x}$ una solución factible que cumple la propiedad *no-wait* (ejemplificada en la Figura 4.3). Por absurdo, supongamos que existe un vértice $k \in V \setminus \{v_s, v_e\}$ y un $t \in \mathbb{T}_{NW}(k)$ tal que el lado izquierdo de (4.16) vale 2. Entonces existen (i,k,m) y (k,j,m') tales que $\widetilde{w}_{ik}^{m+1} \leq t$ y (k,j,m') y $w_{kj}^{m'} > t$, y además $\bar{x}_{ik}^{m} = 1$ y $\bar{x}_{kj}^{m'} = 1$.

Pero esto quiere decir que el momento en que se llega a k es a lo sumo t y el de partida es estrictamente mayor que t. Por lo tanto se debe esperar en k antes de salir hacia el siguiente vértice. Esto es absurdo, dado que $\bar{x}$ cumplía la propiedad *no-wait*. □

Observación 4.3.4. *El tamaño de la familia de desigualdades presentada en (4.20) es* $\mathcal{O}(|V||E|h)$.

4.3.4. No going back cuts (NGBC)

Los siguientes cortes que se presentan son llamados *No Going Back Cuts* (NGBC), y la idea es similar a los NWC: tomar un conjunto de ejes que entren y salgan de un vértice, de manera tal que a lo sumo uno de todos ellos pueda pertenecer a la solución. En este caso, la idea es usar los time zones para impedir que se salga de un vértice k en un instante estrictamente anterior al de llegada.

En la Figura 4.4 se ve de manera gráfica el corte propuesto, dónde dado un vértice k se elige un t que esté en el horizonte de tiempo y se eligen dos conjuntos de nodos. El primero, contiene a todos los ejes que *seguro* salen de k estrictamente antes que t. El segundo, contiene a todos los ejes que llegan en t o después. Al igual que en los cortes anteriores, si se eligiese un eje de cada conjunto para entrar y salir de k, esto implicaría que se arriba en un momento t' y se sale en un momento $t'' < t'$, lo cuál no puede suceder en una solución factible.

Proposición 4.3.5. *La familia de desigualdades NGBC definida en (4.21) es válida para* P_{TD}.

$$\sum_{(i,k,m)\in\delta^{-}_{\geq t}(k)} x_{ik}^{m} + \sum_{(k,j,m)\in\delta^{+}_{<t}(k)} x_{kj}^{m} \leq 1, \quad \forall k \in V \setminus \{v_s, v_e\}, \forall t \in H \tag{4.21}$$

donde

$$\delta^{-}_{\geq t}(k) = \{(i,k,m) \in A \times \mathbb{N} \mid \widetilde{w}_{ik}^{m} \geq t\} \tag{4.22}$$

$$\delta^{+}_{<t}(k) = \left\{(k,j,m) \in A \times \mathbb{N} \mid w_{kj}^{m+1} < t\right\} \tag{4.23}$$

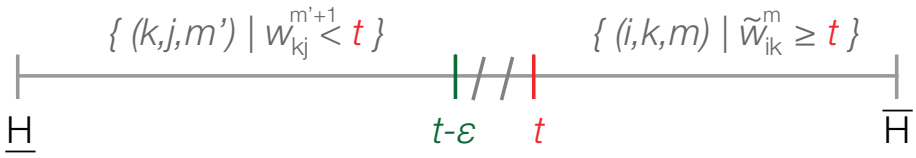


Fig. 4.4: Descripción gráfica de NGBC.

También, como en el caso de la familia NWC se puede reducir el dominio de la variable t, tomando solo en cuenta aquellos que se encuentran en el conjunto $\mathbb{T}_{NGB}(k)$ definido en la siguiente Proposición.

Observación 4.3.6. *Sea $\mathbb{T}_{NGB}(k) = \{\widetilde{w}^m_{ik} \mid (i,k,m) \in A \times \mathbb{N}\}$ un conjunto de instantes en el horizonte. La familia de desigualdades definida en la ecuación (4.24) domina a la familia definida en (4.21).*

$$\sum_{(i,k,m)\in\delta^-_{\geq t}(k)} x^m_{ik} + \sum_{(k,j,m)\in\delta^+_{<t}(k)} x^m_{kj} \leq 1, \quad \forall k \in V \setminus \{v_s, v_e\}, \forall t \in \mathbb{T}_{NGB}(k) \tag{4.24}$$

La demostración es análoga a la presentada para NWC. En este caso se debe tomar $t' = \min\{x \in \mathbb{T}_{NGB}(k) \mid x \geq t\}$ y se debe probar que $\delta^+_{<t}(k) \subseteq \delta^+_{<t'}(k)$ y $\delta^-_{\geq t}(k) \subseteq \delta^-_{\geq t'}(k)$.

Al igual que los NWC se puede reforzar las desigualdades modificando el lado derecho por y_k. Esto resulta en las desigualdades presentadas en la siguiente proposición.

Proposición 4.3.7. *La familia de desigualdades (4.25) es válida para P_{TD} y además domina a (4.24).*

$$\sum_{(i,k,m)\in\delta^-_{\geq t}(k)} x^m_{ik} + \sum_{(k,j,m)\in\delta^+_{<t}(k)} x^m_{kj} \leq y_k, \quad \forall k \in V \setminus \{v_s, v_e\}, \forall t \in \mathbb{T}_{NGB}(k) \tag{4.25}$$

Demostración. El caso cuándo $y_k = 0$ es válido trivialmente ya que ninguna de la izquierda puede estar prendida por las restricciones (4.3), (4.4) y (4.7). Veamos que vale también para el caso $y_k = 1$.

Sea $\bar{x}$ una solución factible. Por absurdo, supongamos que existe un vértice $k \in V \setminus \{v_s, v_e\}$ y un $t \in \mathbb{T}_{NGB}(k)$ tal que el lado izquierdo de (4.21) vale 2. Entonces existen (i,k,m) y (k,j,m') tales que $w^{m'+1}_{kj} < t \leq \widetilde{w}^m_{ik}$ y a su vez $\bar{x}^m_{ik} = 1$ y $\bar{x}^{m'}_{kj} = 1$.

Pero esto quiere decir que el momento en que se llega a k es como mínimo t y aquel en que se parte de k es estrictamente menor a t. Por lo tanto se sale antes de lo que se arriba. Esto es absurdo, porque no cumple con la definición de tiempo de viaje establecida por las restricciones (4.8) y (4.9). □

4.3.5. Time dependent - Edge Generalized Cut Set (TDEGCS)

A continuación presentamos una nueva familia de desigualdades. La idea intuitiva es bastante similar a las SEC: tomar un conjunto X de ejes (i,j,m) que no lleguen a v_e y

comprobar que su conjunto de ejes sucesores $A^+(X)$ conserve el flujo, es decir, que si hay algún eje dentro X que está seleccionado haya al menos uno dentro de $A^+(X)$ que también lo esté.

Sea $\mathbb{X}_j = \{(i,j,m) \in \mathbb{X}\}$ los ejes que entran a un nodo j. Dados $k \in V \setminus \{v_e\}$, $X \subset \mathbb{X}$ tal que $\mathbb{X}_{v_e} \cap X = \emptyset$, se propone la siguiente desigualdad para el modelo.

$$\sum_{(i,j,m)\in A^+(X)} x_{ij}^m \geq \sum_{(i,k,m)\in X} x_{ik}^m \tag{4.26}$$

Además, se puede reforzar la desigualdad (4.26) evitando sumar del lado izquierdo aquellos ejes que llegan a k. Intuitivamente, esto se puede hacer debido a que cuando el lado derecho vale 1, ningún otro eje que llega a k puede estar activo por lo tanto no suman nada al lado izquierdo, caso contrario la desigualdad es trivialmente verdadera. A continuación se presenta formalmente la familia de desigualdades TDEGCS.

Proposición 4.3.8. *Dado $X \subseteq \mathbb{X}, \mathbb{X}_{v_e} \cap X = \emptyset$ y $k \in V \setminus \{v_e\}$. La familia de desigualdades TDEGCS (4.27) es válida para P_{TD}.*

$$\sum_{(i,j,m)\in A^+(X)} x_{ij}^m - \sum_{(i,k,m)\in A^+(X)} x_{ik}^m \geq \sum_{(i,k,m)\in X} x_{ik}^m \tag{4.27}$$

Demostración. Sea $P = [(i_0,i_1,m_1),(i_1,i_2,m_2),\ldots,(i_{r-1},i_r,m_r)]$ un camino time-dependent factible (con $i_0 = v_s$, $i_r = v_e$) que cumple *no-wait*. Sea x^* una solución factible del modelo que lo representa, es decir, tiene $x_{i_0 i_1}^{*m_1} = 1, \ldots x_{i_r i_r}^{*m_r} = 1$.

La primera observación es que el lado derecho de la desigualdad siempre es menor o igual a 1, porque son todos ejes entrando a un mismo vértice k. Por otra parte, el lado izquierdo siempre es mayor o igual a 0 dado que el conjunto $\{(i,k,m) \in A^+(X)\}$ está incluido en $\{(i,j,m) \in A^+(X)\}$. Si el lado derecho es igual a 0, entonces la desigualdad es válida trivialmente. A partir de ahora entonces consideramos el caso $\sum_{(i,k,m)\in X} x_{ik}^m = 1$.

Una observación es que no existe un eje $(i,k,m) \in A^+(X)$ que tome el valor 1 ya que solo un eje entrante a k puede estar activado y ese es el causante de que el lado derecho sea igual a 1. Esto implica que $\sum_{(i,k,m)\in A^+(X)} x_{ik}^m = 0$.

Sabemos que $(i_{r-1},i_r,m_r) \notin X$, porque $i_r = v_e$. Además sabemos que existe un eje $e \in X$ en P, el que hace que la parte derecha de la desigualdad sea igual a 1. Por lo tanto, si tomamos el último índice l tal que $(i_l,j_l,m_l) \in X$, entonces $(i_{l+1},j_{l+1},m_{l+1}) \notin X \Rightarrow (i_{l+1},j_{l+1},m_{l+1}) \in A^+(X)$. Entonces la parte izquierda contiene al menos un eje que vale 1 y por lo tanto es mayor o igual a 1. □

Relación con las Generalized Subtour Constraints A continuación se relaciona las desigualdades TDEGCS con otras familias de desigualdades conocidas en la literatura.

Proposición 4.3.9. *La familia de desigualdades TDEGCS (4.27) son una generalización de las GCS (4.14).*

Demostración. Para probar que las TDEGCS generalizan a las GCS, vamos a encontrar por cada desigualdad de las segundas una equivalente en las primeras. Sea $k \in V \setminus v_s, v_e$ un vértice y S un conjunto de nodos que identifican a una desigualdad de las propuestas en la ecuación (4.14).

Tomamos $X = \{(i,j,m) \in \mathbb{X} \mid (i,j) \in \delta^+(S)\}$. Como todos los ejes $(i,k,m) \in X$ entonces $\sum_{(i,k,m)\in X} x_{ik}^m = y_k$, lo cuál es el lado derecho de (4.27).
Ahora veamos que el lado izquierdo también es equivalente. Primero, notemos que como todos los ejes $(i,k,m) \in X$, esto también implica que ningún eje que arribe a k va a pertenecer a $A^+(X)$, por lo cual el término que resta del lado izquierdo de las TDEGCS es igual a 0, ya que el conjunto de ejes sobre los que suma es vacío. Por lo tanto,

$$\sum_{(i,j,m)\in A^+(X)} x_{ij}^m - \sum_{(i,k,m)\in A^+(X)} x_{ik}^m = \sum_{(i,j,m)\in A^+(X)} x_{ij}^m = \sum_{(i,j)\in\delta^+(S)} \sum_{m=0}^{|T_{ij}|} x_{ij}^m = \sum_{(i,j)\in\delta^+(S)} x_{ij} \tag{4.28}$$

Entonces queda probado que para cada desigualdad GCS existe una TDEGCS equivalente. □

Relación con otras familias de desigualdades Antes de continuar con el desarrollo, incluimos un comentario respecto a la familia de desigualdades válidas TDEGCS. Notamos que las mismas comparten algunas de las características con las denominadas *Admissible Flow Constratins* propuestas por Abeledo et al. [1] para una variante del TSP donde el tiempo de viaje depende de la posición del arco en el circuito. Sin embargo, si bien ambos problemas son similares, la estructura de las redes subyacentes presentan algunas diferencias que no permiten establecer una relación directa entre las desigualdades, al menos de manera evidente. Por otro lado, las TDEGCS fueron derivadas tomando como punto de partida las *Bucket Subtour Elimination Constraints* propuestas en Dash et al. [7]. En este caso, se trabaja también sobre una red extendida para el TSP-TW pero con particiones del horizonte de tiempo en intervalos al nivel de los vértices, a diferencia de nuestro caso en donde los TTB dependen de cada eje en particular.

El estudio detallado de la (potencial) relación entre estas familias y los problemas antes mencionados quedan fuera del alcance de esta tesis. Sin embargo, destacamos que es un aspecto interesante a estudiar en una etapa futura, las similitudes y diferencias entre las mismas, tanto desde el punto de vista teórico, los algoritmos de separación, como así también el impacto desde el punto de vista práctico en cada uno de los problemas.

4.4. Algoritmos de separación

En esta sección se explica en detalle los algoritmos utilizados para separar las desigualdades presentadas en la sección anterior.

4.4.1. Generalized Cut Set (GCS)

La separación de los GCS se puede realizar en tiempo polinomial cómo se explica en [8]. Básicamente la idea es utilizar el grafo soporte, es decir el grafo de la instancia con peso

x^*_{ij} que es el valor que toma la variable x_{ij} en la relajación, y resolver flujo máximo.

Más precisamente, para cada $k \in V \setminus \{v_s, v_e\}$ se resuelve el flujo máximo con fuente k y sumidero v_e y se consigue el corte mínimo S, que son aquellos nodos alcanzables desde k en la red residual.

Luego, el valor del flujo máximo, que es igual que el corte de capacidad mínima, corresponde al menor valor que puede tomar el lado izquierdo de la desigualdad (4.14). De ser menor que y_k entonces la desigualdad está violada.

4.4.2. Generalized Large Multistar (GLM)

Los cortes GLM pueden ser separados en tiempo polinomial como se explica en [17]. Sin embargo, como es un corte que no influye mucho en la performance del modelo presentado en la sección anterior para el set de instancias considerado, se limita su implementación a resolver un PLEM que indique la desigualdad más violada.

Para resolver el MIP se utilizaron únicamente los ejes

$$E' = \left\{(i,j) \in E \mid x^*_{ij} + x^*_{ji} > 0\right\} \tag{4.29}$$

Las variables utilizadas tienen la siguiente semántica:

- s_i: Vale 1 si el vértice $i \in V$ pertenece a S, 0 si no.
- o_{ij}: Vale 1 si el eje (i,j) pertenece a $\delta^+(S)$, 0 si no.

$$\text{mín} \sum_{(i,j)\in E'} x_{ij}o_{ij} - \frac{1}{Q}\left(\sum_{i\in V} s_i q_i y_i - \sum_{(i,j)\in E'} o_{ij}q_j(x_{ij}+x_{ji})\right) \tag{4.30}$$

sujeto a

$$o_{ij} \leq \tfrac{1}{2}(s_i + (1 - s_j)) \qquad \forall (i,j) \in E' \tag{4.31}$$

$$o_{ij} \geq s_i - s_j \qquad \forall (i,j) \in E' \tag{4.32}$$

$$s_{v_s} = s_{v_e} = 0 \tag{4.33}$$

$$\sum_{i\in V\setminus\{v_s\}} s_i \geq 2 \tag{4.34}$$

$$o_{ij} \in \{0,1\} \qquad \forall (i,j) \in E' \tag{4.35}$$

$$s_i \in \{0,1\} \qquad \forall i \in V \tag{4.36}$$

4.4.3. No waiting cuts (NWC)

Se muestra a continuación el Algoritmo 8 para encontrar la desigualdad más violada de la familia NWC para cada vértice $k \in V \setminus \{v_s, v_e\}$ dada una solución de la relajación x^*, y^*.

La idea del algoritmo es iterar por cada vértice $k \in V \setminus \{v_s, v_e\}$ e ir moviéndose desde el comienzo del horizonte hacia el final por todos los puntos de $\mathbb{T}_{NW}$. El algoritmo puede hacerse de manera eficiente notando que si se ordenan todos los puntos de $\mathbb{T}_{NW}(k)$ en una secuencia T, vale que $\delta^-_{\leq T_1}(k) \subseteq \delta^-_{\leq T_2}(k) \subseteq \ldots \subseteq \delta^-_{\leq T_{|T|}}(k)$. Por otra parte también vale que $\delta^+_{>T_{|T|}}(k) \subseteq \delta^+_{>T_{|T|-1}}(k) \subseteq \ldots \subseteq \delta^+_{>T_1}(k)$.

Otro aspecto a considerar es que al comienzo del horizonte todos los ejes salientes $(k, j, m') \in \mathbb{X}$ pertenecen a $\delta^+_{>\underline{H}}(k)$. Pero ningún eje entrante $(i, k, m) \in \mathbb{X}$ pertenece a $\delta^-_{\leq \underline{H}}(k)$. Esto se puede comprobar siguiendo las definiciones de $\delta^+_{>t}, \delta^-_{\leq t}$. Como $\sum_{(k,j,m') \in \mathbb{X}} x^{m'}_{kj} = y_k$ por las restricciones (4.3) y (4.4), entonces con $t = \underline{H}$ el lado izquierdo de la desigualdad (4.20) vale y^*_k.

Luego, a medida que se avanza en el horizonte de tiempo hay ejes (k, j, m') que dejan de pertenecer a $\delta^+_{>\underline{H}}(k)$, como se mencionó antes, y ejes (i, k, m) que comienzan a pertenecer a $\delta^-_{\leq \underline{H}}(k)$. Estos se pueden encontrar rápidamente si se tiene el conjunto de ejes $(i, k, m) \in \delta^-_{\leq t}(k)$ ordenado por $\widetilde{w}^{m+1}_{ik}$ y el de ejes $(k, j, m) \in \delta^+_{>t}(k)$ ordenado por w^m_{kj}. Estos conjuntos pueden estar precomputados o se pueden computar en el momento. La ventaja de ordenarlos en el momento es que se puede utilizar únicamente los ejes del grafo soporte, lo cual puede reducir la cantidad de ejes a considerar. Del mismo modo se puede precomputar el conjunto $\mathbb{T}_{NW}(k)$ para cada k.

Al finalizar de recorrer el horizonte de tiempo se elige el $t \in \mathbb{T}_{NW}(k)$ que tuvo su lado izquierdo con mayor valor y si este es más grande que y_k entonces esa desigualdad está violada.

Este algoritmo de separación se puede computar con complejidad $\mathcal{O}(|V| + |E|h)$.

4.4.4. No going back cuts (NGBC)

Se muestra el Algoritmo 9 para encontrar la desigualdad más violada de la familia NGBC para cada vértice $k \in V \setminus \{v_s, v_e\}$ dada una solución de la relajación x^*, y^*.

Para separar la familia de desigualdades NGBC se utiliza una idea similar a NWC. Del mismo modo se itera por cada vértice $k \in V \setminus \{v_s, v_e\}$ y se va moviendo un t' a lo largo del horizonte del tiempo pasando por todos los puntos de $\mathbb{T}_{NW}$. En este caso la observación que permite hacer eficiente el algoritmo es que al ordenar $\mathbb{T}_{NGB}(k)$ en una secuencia T, vale que $\delta^+_{<T_1}(k) \subseteq \delta^+_{<T_2}(k) \subseteq \ldots \subseteq \delta^+_{<T_{|T|}}(k)$. Por otra parte también vale que $\delta^-_{\geq T_{|T|}}(k) \subseteq \delta^-_{\geq T_{|T|-1}}(k) \subseteq \ldots \subseteq \delta^-_{\geq T_1}(k)$.

En este caso al comienzo del horizonte de tiempo ningún eje saliente $(k, j, m') \in \mathbb{X}$ pertenece a $\delta^+_{<\underline{H}}(k)$. Pero todos los ejes entrantes $(i, k, m) \in \mathbb{X}$ pertenecen a $\delta^-_{\geq \underline{H}}(k)$. Al

Algorithm 8 Algoritmo de separación de NWC. Complejidad: $\mathcal{O}(|V| + |E|h)$

function SEPARATENWC($\mathbb{T}_{NW}, y^*, x^*$)
  for $k \in V \setminus \{v_s, v_e\}$ **do** ▷ $\mathcal{O}(|V| + |E|h)$
    if $y_k^* = 0$ **then**
      continue
    $\Delta_{best}, \Delta_{current} \leftarrow y_k^*$
    $T \leftarrow \mathbb{T}_{NW}(k)$
    $T_0 \leftarrow \underline{H}$
    for $i \in 1 \dots |T|$ **do** ▷ $\mathcal{O}(grado(k)h)$
      $X^- \leftarrow \left\{(i,k,m) \in (\delta^-_{\leq T_i}(k) \setminus \delta^-_{\leq T_{i-1}}(k))\right\}$
      $X^+ \leftarrow \left\{(k,j,m) \in (\delta^+_{>T_{i-1}}(k) \setminus \delta^+_{>T_i}(k))\right\}$
      $\Delta_{current} \leftarrow \Delta_{current} + \sum_{(i,k,m)\in X^-} x_{ik}^{*m} - \sum_{(k,j,m)\in X^+} x_{kj}^{*m}$
      $\Delta_{best} \leftarrow \max(\Delta_{best}, \Delta_{current})$
    if $\Delta_{best} > y_k$ **then**
      add cut with k **and** t **for** Δ_{best}

igual que en las NWC, se puede comprobar siguiendo las definiciones de $\delta^+_{<t}, \delta^-_{\geq t}$. Como $\sum_{(k,j,m')\in\mathbb{X}} x_{kj}^{m'} = y_k$ por las restricciones (4.3) y (4.4), entonces con $t = \underline{H}$ el lado izquierdo de la desigualdad (4.20) vale y_k^*.

Luego, a medida que se avanza en el horizonte de tiempo hay ejes (i,k,m) que dejan de pertenecer a $\delta^-_{\geq \underline{H}}(k)$, como se mencionó antes, y ejes (k,j,m') que comienzan a pertenecer a $\delta^+_{<\underline{H}}(k)$. En este caso es conveniente tener ordenado el conjunto de ejes $(i,k,m) \in \delta^-_{\geq t}(k)$ ordenado por $\widetilde{w}_{ik}^m$ y el de ejes $(k,j,m) \in \delta^+_{<t}(k)$ ordenado por w_{kj}^{m+1}.

Algorithm 9 Algoritmo de separación de NGBC. Complejidad: $\mathcal{O}(|V| + |E|h)$

function SEPARATENGBC($\mathbb{T}_{NGB}, y^*, x^*$)
  for $k \in V \setminus \{v_s, v_e\}$ **do** ▷ $\mathcal{O}(|V| + |E|h)$
    if $y_k^* = 0$ **then**
      continue
    $\Delta_{best}, \Delta_{current} \leftarrow y_k^*$
    $T \leftarrow \mathbb{T}_{NGB}(k)$
    $T_0 \leftarrow \underline{H}$
    for $i \in 1 \dots |T|$ **do** ▷ $\mathcal{O}(grado(k)h)$
      $X^- \leftarrow \left\{(i,k,m) \in (\delta^-_{\geq T_{i-1}}(k) \setminus \delta^-_{\geq T_i}(k))\right\}$
      $X^+ \leftarrow \left\{(k,j,m) \in (\delta^+_{<T_i}(k) \setminus \delta^+_{<T_{i-1}}(k))\right\}$
      $\Delta_{current} \leftarrow \Delta_{current} - \sum_{(i,k,m)\in X^-} x_{ik}^{*m} + \sum_{(k,j,m)\in X^+} x_{kj}^{*m}$
      $\Delta_{best} \leftarrow \max(\Delta_{best}, \Delta_{current})$
    if $\Delta_{best} > y_k$ **then**
      add cut with k **and** t **for** Δ_{best}

4.4.5. Time dependent - Edge Generalized Cut Set (TDEGCS)

En el caso de los TDEGCS la separación se resuelve de manera parecida a los GCS, es decir, creando una red y resolviendo flujo máximo para hallar el corte mínimo. A continuación se provee una intuición de como funciona este algoritmo de separación y luego se formaliza el mismo.

Esta separación considera una red $G(k)$ con un vértice por cada eje $(i, j, m) \in \mathbb{X}$ y un arco entre dos de ellos si el eje que representa el segundo es sucesor del primero. De este modo un camino de ejes dependientes del tiempo en el grafo original es un camino de vértices en esta red y viceversa.

Para encontrar el flujo máximo de la red hay que definir las capacidades. En este caso contamos con capacidades en los vértices dado que en cada eje dependiente del tiempo (i, j, m) hay una cota superior de flujo que puede pasar que esta dada por el valor x_{ij}^{*m} de la relajación. Como en todo problema de flujo donde la capacidad esta en los vértices se utiliza la técnica de dividir cada vértice u en dos u^- y u^+ unidos por un arco con su capacidad. Al primero de ellos llegan todos los arcos que arribaban al vértice original u y al segundo todos aquellos que salían de el. Por ultimo, se utiliza una capacidad más grande que todos los x_{ij}^{*m} en el resto de los arcos de la red para que no afecten el valor del flujo máximo.

Por último, para encontrar una desigualdad que esté violada es necesario fijar un vértice k y agregar una fuente y un sumidero a la red. La fuente debe tener un arco hacia aquellos vértices representando ejes que entran a k y todo vértice que representa un eje que entra al deposito final debe tener un arco hacia el sumidero. De este modo si existe una desigualdad violada quiere decir que existe un conjunto de ejes X tal que los ejes que salen de él no preservan el flujo que ingreso a k hasta el depósito de fin. A continuación se formaliza todo este análisis.

Dado $k \in V \setminus \{v_s, v_e\}$ un vértice fijo, se define la red $G(k) = (V', E')$ con los vértices

$$V' = \left(\bigcup_{(i,j,m)\in\mathbb{X}} \left\{ u_{ijm}^-, u_{ijm}^+ \right\} \right) \cup \{u_s, u_e\}, \tag{4.37}$$

los arcos

$$\begin{aligned} E' = &\left\{ (u_{ijm}^-, u_{ijm}^+) \mid (i, j, m) \in \mathbb{X} \right\} \\ &\cup \left\{ (u_s, u_{ikm}^-) \mid (i, k, m) \in \mathbb{X} \right\} \\ &\cup \left\{ (u_{i,v_e,m}^+, u_e) \mid (i, v_e, m) \in \mathbb{X} \right\} \\ &\cup \left\{ (u_{ijm}^+, u_{jlm'}^-) \mid (i, j, m) \in \mathbb{X}, (j, l, m') \in \widetilde{A}_{ij}^m \right\} \end{aligned} \tag{4.38}$$

y las capacidades de los arcos definidas por (suponiendo ϵ estrictamente positivo):

$$c(a) = \begin{cases} x_{ij}^{*m} & \text{si } a = (u_{ijm}^-, u_{ijm}^+) \\ 1 + \epsilon & \text{caso contrario} \end{cases} \tag{4.39}$$

Además se llama S^* a un corte mínimo de la red $G(k)$ creado a partir de incluir los vértices alcanzables desde la fuente en la red residual del flujo máximo. De aquí en adelante cuando nos referimos a un corte mínimo S^* es generado de este modo. A su vez, siendo S un corte cualquiera de $G(k)$ se define $\delta^+(S)$ como el conjunto de arcos que van desde un vértice en S hacia uno en $V' \setminus S$.

Por otra parte se define Δ_X^k como una medida de cuán violada está la desigualdad definida por k y X.

$$\Delta_X^k = \sum_{(i,j,m)\in A^+(X)} x_{ij}^{*m} - \sum_{(i,k,m)\in A^+(X)} x_{ik}^{*m} - \sum_{(i,k,m)\in X} x_{ik}^{*m} \tag{4.40}$$

Para probar que resolver flujo máximo sobre esta red es suficiente para encontrar la desigualdad más violada se va a utilizar una demostración que tiene los siguientes pasos.

1. Primero se define una transformación $\sigma(X)$ que convierte un conjunto de ejes dependientes del tiempo X en un corte de $G(X)$. Se prueba que para todo conjunto X, que cumple las hipótesis de la desigualdad, el corte $\sigma(X)$ tiene una capacidad $c(\sigma(X)) = \Delta_X^k + y_k^*$. De este modo sabemos que todo conjunto X está representado por un corte en la red.

2. En segundo lugar se prueba que un corte mínimo S^* de la red $G(k)$ representa a un X en particular, es decir, el corte generado por la transformación $\sigma(X)$ es exactamente S^*.

Los puntos 1 y 2 nos permiten afirmar que tomando el corte mínimo S^* de la red $G(k)$ y eligiendo el X correspondiente se puede encontrar la desigualdad más violada, si existe, para un vértice k. A continuación se define una serie de Lemas que serán útiles en la demostración formal de la correctitud del algoritmo.

Lema 4.4.1. *Sea S^* un corte mínimo de $G(k)$. Si (i,j,m) es sucesor de (l,i,m') y $u_{lim'}^+ \in S^*$ entonces $u_{ijm}^- \in S^*$.*

Demostración. Supongamos que no, entonces tomamos $S' = S^* \cup u_{ijm}^-$.

$$c(S') \leq c(S^*) - c(u_{lim'}^+, u_{ijm}^-) + c(u_{ijm}^-, u_{ijm}^+) \leq c(S^*) - (1+\epsilon) + 1 < c(S^*) \tag{4.41}$$

Absurdo, pues S^* era el corte de capacidad mínima. □

Lema 4.4.2. *Sea S^* un corte mínimo de la red $G(k)$. Si $u_{ijm}^+ \in S^*$ entonces $u_{ijm}^- \in S^*$.*

Demostración. Si u_{ijm}^+ pertenece a S^* es porque es alcanzable desde u_s en la red residual luego de resolver el flujo máximo. Como el único nodo predecesor de u_{ijm}^+ es u_{ijm}^-, entonces u_{ijm}^- es alcanzable y por lo tanto $u_{ijm}^- \in S^*$. □

Lema 4.4.3. *Sea S^* un corte mínimo de $G(k)$. Para todo $(i,k,m) \in \mathbb{X}$ vale que $u_{ikm}^- \in S^*$.*

Demostración. Por absurdo, suponemos que existen i, m tales que $u_{ikm}^- \notin S^*$. Pero esto implica que $(u_s, u_{ikm}^-) \in \delta^+(S^*)$, por lo tanto, tomando $S' = S^* \cup \{u_{ikm}^-\}$ tenemos que

$$c(S') \leq c(S^*) - c(u_s, u_{ikm}^-) + c(u_{ikm}^-, u_{ikm}^+) \leq c(S^*) - (1+\epsilon) + 1 < c(S) \tag{4.42}$$

Lo cual es absurdo ya que S^* era el corte de mínima capacidad. □

En la siguiente proposición se prueba el punto 1 de la demostración. Es decir, mostrar que cada conjunto de ejes X está representado en la red por un corte $\sigma(X)$. Primero, se define la transformación $\sigma(X)$ de ejes dependientes del tiempo de G en cortes de $G(k)$.

$$\sigma(X) = \{u_s\} \cup \left\{u^-_{ijm} \mid (i,j,m) \in A^+(X) \cup X \vee j = k\right\} \cup \left\{u^+_{ijm} \mid (i,j,m) \in X\right\} \quad (4.43)$$

Proposición 4.4.4. *Sea $k \in V \setminus \{v_s, v_e\}$ un vértice de G, sea $X \subseteq \mathbb{X}$ que cumple con las condiciones de (4.27). La capacidad del corte $\sigma(X)$ en la red $G(k)$ es igual a $\Delta^k_X + y^*_k$.*

Demostración. En primer lugar, vemos que todos los ejes que pertenecen a $\delta^+(\sigma(X))$ tienen la forma (u^-_{ijm}, u^+_{ijm}). Para esto probamos que no existe ningún otro tipo de eje.

1. $(u_s, u^-_{ikm}) \notin \delta^+(\sigma(X))$: Esto es verdadero porque $u^-_{ijm} \in \sigma(X)$ si $j = k$.
2. $(u^+_{i,v_e,m}, u_e) \notin \delta^+(\sigma(X))$: Esto vale ya que $(i, v_e, m) \notin X$ por las condiciones de la desigualdad (4.27), entonces $u^+_{i,v_e,m} \notin \sigma(X)$ para todo i, m.
3. $(u^+_{ijm}, u^-_{jlm'}) \notin \delta^+(\sigma(X))$: Para probar esto vamos a ver que si $u^+_{ijm} \in \sigma(X)$ entonces $u^-_{jlm'}$ también. Por la definición (4.43) sabemos que si $u^+_{ijm} \in \sigma(X)$ entonces $(i,j,m) \in X$. Por absurdo, supongamos que $u^-_{jlm'} \notin \sigma(X)$. Como el arco $(u^+_{ijm}, u^-_{jlm'})$ existe, eso quiere decir que (j,l,m') es sucesor de (i,j,m). Además, sabemos que $(j,l,m') \notin X$ porque $u^-_{jlm'} \notin \sigma(X)$ y esto implica que $(j,l,m') \in A^+(X)$. La definición (4.43) nos indica entonces que $u^-_{jlm'} \in \sigma(X)$ lo cual es absurdo. Por lo tanto concluimos que $(u^+_{ijm}, u^-_{jlm'}) \notin \delta^+(\sigma(X))$.

Sabiendo que todos los ejes que salen del corte $\sigma(X)$ tienen la forma (u^-_{ijm}, u^+_{ijm}), procedemos a mostrar para cuales ejes (i,j,m) es cierto que $(u^-_{ijm}, u^+_{ijm}) \in \delta^+(\sigma(X))$.
Analizando (4.43) se puede ver que aquellos (i,j,m) que cumplen con $u^-_{ijm} \in \sigma(X)$ y $u^+_{ijm} \notin \sigma(X)$ son aquellos que están en el conjunto:

$$\left\{u^-_{ijm} \mid (i,j,m) \in A^+(X) \vee (j = k \wedge (i,j,m) \notin X)\right\} \quad (4.44)$$

Este conjunto se puede reescribir como una unión disjunta de otros dos conjuntos de la siguiente manera.

$$\left\{u^-_{ijm} \mid (i,j,m) \in A^+(X)\right\} \cup \left\{u^-_{ikm} \mid (i,k,m) \notin (X \cup A^+(X))\right\} \quad (4.45)$$

Por lo tanto,

$$\begin{aligned}
c(\sigma(X)) &= \sum_{(i,j,m)\in\delta^+(\sigma(X))} c(u^-_{ijm}, u^+_{ijm}) \\
&= \sum_{(i,j,m)\in A^+(X)} c(u^-_{ijm}, u^+_{ijm}) + \sum_{(i,k,m)\notin(X\cup A^+(X))} c(u^-_{ikm}, u^+_{ikm}) \\
&= \sum_{(i,j,m)\in A^+(X)} x^{*m}_{ij} + \sum_{(i,k,m)\notin(X\cup A^+(X))} x^{*m}_{ik} \\
&= \sum_{(i,j,m)\in A^+(X)} x^{*m}_{ij} + y^*_k - \sum_{(i,k,m)\in(X\cup A^+(X))} x^{*m}_{ik} \\
&= \sum_{(i,j,m)\in A^+(X)} x^{*m}_{ij} + y^*_k - \sum_{(i,k,m)\in X} x^{*m}_{ik} - \sum_{(i,k,m)\in A^+(X)} x^{*m}_{ik} \\
&= \Delta^k_X + y^*_k
\end{aligned} \quad (4.46)$$

Entonces, quedó probado que dado cualquier X que cumpla con las condiciones de la desigualdad (4.27) existe un corte $\sigma(X)$ de la red $G(k)$ tal que $c(\sigma(X)) = \Delta_X^k + y_k^*$. □

En la siguiente proposición vamos a probar la segunda parte de la demostración.

Proposición 4.4.5. *Sea S^* un corte mínimo de $G(k)$. Existe $X \subseteq \mathbb{X}$ que cumple las condiciones de (4.27) tal que $\sigma(X) = S^*$.*

Demostración. Para la demostración tomamos $X = \left\{(i,j,m) \in \mathbb{X} \mid u_{ijm}^+ \in S^*\right\}$. Para probar que $\sigma(X) = S^*$ vamos a probar por separado las siguientes 4 equivalencias:

1. $u_s \in S^* \Leftrightarrow u_s \in \sigma(X)$
2. $u_e \in S^* \Leftrightarrow u_e \in \sigma(X)$
3. $u_{ijm}^+ \in S^* \Leftrightarrow u_{ijm}^+ \in \sigma(X)$
4. $u_{ijm}^- \in S^* \Leftrightarrow u_{ijm}^- \in \sigma(X)$

La primera y segunda equivalencia son sencillas, ya que u_s pertenece a todos los cortes y u_e a ninguno, por ser ambos la fuente y el sumidero respectivamente. Para la tercera, basta ver la siguiente cadena de equivalencias:

$$u_{ijm}^+ \in S^* \Leftrightarrow (i,j,m) \in X \Leftrightarrow u_{ijm}^+ \in \sigma(X) \tag{4.47}$$

Para probar el cuarto punto primero veamos qué sucede cuándo $j = k$. Por el Lema 4.4.3 sabemos que $u_{ikm}^- \in S^*$, pero además por (4.43) sabemos que $u_{ikm}^- \in \sigma(X)$, por lo cual este caso es válido. Ahora veamos que sucede cuando $j \neq k$. Para esto vamos a separar la demostración en la ida y vuelta de la doble implicación.

$\Leftarrow$) Para la vuelta tenemos dos casos: o bien $(i,j,m) \in X$ o $(i,j,m) \in A^+(X)$.
Caso 1: Si $(i,j,m) \in X \Rightarrow u_{ijm}^+ \in S^* \xRightarrow{(4.4.2)} u_{ijm}^- \in S^*$.
Caso 2: Si $(i,j,m) \in A^+(X) \Rightarrow \exists (l,i,m') \in X$ tal que (i,j,m) es su sucesor $\Rightarrow u_{lim'}^+ \in S^* \xRightarrow{(4.4.1)} u_{ijm}^- \in S^*$.

$\Rightarrow$) Si $u_{ijm}^+ \in S^*$ entonces $(i,j,m) \in X$ y por lo tanto $u_{ijm}^- \in \sigma(X)$. Si no, como $j \neq k$ existe un vértice $u_{lim'}^+ \in S^*$ que es predecesor de u_{ijm}^- ya que para u_{ijm}^- estar en S^* tiene que ser alcanzable desde u_s en la red residual correspondiente al flujo máximo. Como $u_{lim'}^+ \in S^*$ entonces $(l,i,m') \in X$ y por lo tanto $(i,j,m) \in A^+(X)$ lo que implica que $u_{ijm}^- \in \sigma(X)$.

Como se probaron todos los casos, queda demostrado que $u_{ijm}^- \in S^* \Leftrightarrow u_{ijm}^- \in \sigma(X)$. Por lo tanto, podemos concluir que $S^* = \sigma(X)$. □

Teorema 4.4.6. *La separación de la familia de desigualdades TDEGCS se puede resolver con un problema de flujo-máximo/corte-mínimo.*

Demostración. Las Proposiciones 4.4.4 y 4.4.5 muestran que tomando cada vértice $k \in V$, construyendo su correspondiente red $G(k)$ y obteniendo un corte mínimo S^* sobre ella se puede obtener un conjunto $X = \left\{(i,j,m) \in \mathbb{X} \mid u_{ijm}^+ \in S^*\right\}$ que junto con el vértice k

forman la desigualdad (4.27) más violada entre todas las que utilizan el mismo k.

Esto es así dado que la Proposición 4.4.4 garantiza que para cada vértice k hay un corte con capacidad igual a $\Delta_X^k + y_k^*$ para todo $X \in \mathbb{X}$ y por otra parte la Proposición 4.4.5 muestra que existe un X' que viola la desigualdad tanto como la capacidad del corte mínimo S^* menos la constante y_k^*, es decir, $\Delta_{X'}^k = c(S^*) - y_k^*$. □

5. EXPERIMENTACIÓN

Los modelos se implementaron en el lenguaje de programación C++11. Los experimentos se realizaron en una computadora con CPU Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz con 16GB de memoria. Para todos los experimentos se utiliza un límite de tiempo de 2 horas. Se utiliza CPLEX 12.7.1 como paquete de resolución PL y PLEM para resolver la relajación del nodo raíz y para desarrollar algoritmos de B&C.

5.1. Instancias

El dataset utilizado para los experimentos se realizó en base al conjunto de instancias presentado por Dabia [6]. Este dataset se generó teniendo en mente que el problema TD-PESPPRC suele ser resuelto, generalmente, como problema esclavo del TD-VRPRC.
Los pasos para generar el dataset son los siguientes:

- Se toman las instancias TDVRPRC del dataset de Dabia.
- Se le quitan los Time Windows.
- Se plantea una formulación por *set partitioning* para resolver la relajación del problema maestro utilizando generación de columnas.
- Se identifican columnas con costo reducido negativo usando distintos algoritmos (heurísticos o exactos) hasta alcanzar un timeout o no encontrar más columnas.
- Se toman columnas con funcionales del PL distantes entre sí.

En nuestro caso, utilizamos una instancia de cada tipo $R1, R2, C1, C2, RC1, RC2$ y se eligieron 5 columnas para cada instancia. Quedan conformadas así las columnas que se presentan en el apéndice. Las instancias se denominan con el formato I_C donde I es el nombre de la instancia TDVRPRC asociada, y C es el identificador de la columna: 0 si es la primera columna del problema maestro, y letras ascendentes según cuán adelante son generadas.

Por ejemplo, C_101_0 es la primera instancia correspondiente a la primera columna que se genera para el problema esclavo en la instancia C_101 de TDVRPRRC. Luego, en algún punto de la resolución de TDVRPRC se toma el problema esclavo que se está resolviendo y esa es la instancia C_101_a, luego se repite lo mismo generando las instancias C_101_b, C_101_c, y así sucesivamente. En particular, se puede observar que hay una relación entre el valor del funcional de cada instancia y el momento en que es generada. Cuanto más adelante en el proceso de generación de columnas fue creada, más alto es el funcional, tendiendo a ser cero (la última columna). Esto es particularmente importante a la hora de analizar la experimentación dado que algunas instancias alcanzan su óptimo con z^* en el orden de los miles, otras en los cientos y algunas otras en las decenas.

5.2. Nomenclatura

Existe una serie de nombres que se le da a variables que se pueden analizar de los PLEM y es importante aclarar su significado antes de evaluar los resultados. Todos estos nombres están basados en que se trata de un problema de minimización. En el caso de problemas de maximización cambia el sentido de las cotas (inferiores por superiores y viceversa). Para cada instancia se conoce el valor óptimo de la función objetivo z^*, el cual se obtuvo dejando correr el algoritmo sin límite de tiempo.

- Root Lower Bound (z_{RLB}): Indica el valor de la relajación lineal del modelo luego de terminar de procesar el nodo raíz. La principal diferencia con la relajación del modelo es que una o varias iteraciones de planos de corte suceden antes de reportar z_{RLB}.
- Best Lower Bound (z_{LB}): Es la mejor cota inferior encontrada luego de terminar la ejecución del algoritmo. Generalmente corresponde al nodo abierto con el funcional más bajo.
- Upper Bound (z_{UB}): Indica el funcional de la mejor solución factible encontrada al finalizar el algoritmo.
- Root Gap (%rGap): Indica la distancia entre z_{RLB} y la solución óptima z^* de la instancia. El %rGap se calcula como $100 * (z^* - z_{RLB})/(|z^*| + 1e^{-10})$.
- Final Gap (%fGap): Indica la distancia entre el funcional de la mejor solución z_{UB} encontrada al finalizar el algoritmo y la solución óptima z^* de la instancia. El %fGap se calcula como $100 * (z_{UB} - z^*)/(|z^*| + 1e^{-10})$.
- Tiempo de cómputo (Tiempo): El tiempo de ejecución total que es medido en segundos. Este tiempo incluye tanto la resolución del PLEM como de las heurísticas si existen.
- Nodos enumerados (#Nodos): Es la cantidad de nodos que se enumeraron y resolvieron en el árbol de B&B o B&C.

Es importante notar que como es una medida relativa cuando z^* es cercano a 0, esta medida se vuelve muy sensible a la distancia absoluta. Por ejemplo, si $z^* = 0{,}4$ y $z_{LB} = 0{,}1$ entonces $\%rGap = \%300$, mientras que cuando $z^* = 100{,}4$ y $z_{LB} = 100{,}1$ $\%rGap = \%29$. Esto hay que tenerlo en cuenta a la hora de analizar los resultados porque como se mencionó antes cuánto más grande es la letra que representa a la instancia *0, a, b, c, d* más cercano a 0 tiende a ser el funcional.

5.3. Diseño de experimentos

En general los PLEM tienen muchas variables que son interesantes de analizar en una experimentación, entre ellas se encuentran:

- Tiempo total de ejecución
- Cantidad de nodos que se enumeran en el árbol de Branch&Bound.

- Calidad de la relajación lineal del modelo con respecto a la solución óptima.
- Calidad de la heurística inicial.
- Calidad e impacto de los planos de cortes.
  - Reducción en la cantidad de nodos enumerados
  - Mejora de las cotas provistas por la relajación lineal.
  - Tiempo total de ejecución por iteración del algoritmo de separación.
  - Peso agregado al modelo para resolver iteraciones de Simplex.

La calidad de los planos de corte es importante de analizar porque nos permite evaluar cómo configurarlos posteriormente. Generalmente hay un compromiso entre las primeras métricas expuestas (reducción de la cantidad de nodos y mejora de las cotas) y las últimas dos (tiempo de ejecución y peso del modelo). A su vez se cuenta con un número muy grande de distintos experimentos que se puede realizar, ya que se puede además ajustar distintos parámetros. Algunas variantes incluyen:

- Tiempo límite de ejecución.
- Tiempo invertido en una heurística inicial.
- Limitar la cantidad total de iteraciones de separación que se hace por cada familia de desigualdades.
- Restringir la cantidad de iteraciones de separación por nodo.
- Parar de agregar cortes cuando el funcional no cambió por k iteraciones.
- Agregar desigualdades de una familia únicamente si no se encontró ninguna desigualdad violada de un conjunto de otras familias.

Es por esto es importante diseñar un plan de experimentación que permita encontrar los parámetros adecuados para obtener los mejores resultados posibles.

El plan que se lleva a cabo está compuesto por los siguientes pasos:

1. *Evaluar la calidad de la heurística inicial*: Para esto se ejecuta el algoritmo heurístico y se obtiene el costo de cada camino resultante. Esto, luego, se compara con la mejor solución encontrada a lo largo de toda la experimentación y se informa cuál es su Gap.
2. *Analizar el impacto de cada corte en el nodo raíz*: En este caso se intenta medir cuánto mejora la relajación con cada corte evaluado de forma independiente, y además qué impacto tiene sobre el tiempo realizar todas estas iteraciones de separación. Para evitar que un tiempo innecesario se invierta en separación de cortes, se define un criterio de parada que establece que si luego de 3 iteraciones no hay una mejora de la relajación de más de 0.1 en el valor absoluto del funcional, entonces se para de agregar cortes. Este valor fue encontrado a partir de experimentación en algunos casos aislados. Luego de experimentar se reporta el %rGap, el tiempo de separación total y promedio por iteración, la cantidad de iteraciones, y el tiempo total invertido en cerrar el nodo raíz.

3. *Elegir y analizar configuraciones de los parámetros*: Una vez que se tiene una noción de la efectividad de cada corte, se procede a definir un conjunto de parámetros con potencial de maximizar el rendimiento de nuestro modelo. Para esta elección se tendrá en cuenta el impacto sobre el %rGap y el tiempo de cómputo de la separación principalmente.
   Luego se ejecuta el algoritmo B&C (con un tiempo límite de 2 horas) con cada configuración de parámetros, partiendo de la solución heurística. Por cada conjunto de parámetros, se reporta el resultado de cada instancia. Éste está compuesto por #Nodos, cantidad de cortes de cada tipo que fueron agregados y el tiempo invertido en cada uno de ellos, z_{RLB}, z_{LB}, z_{UB}. Por último se elige la mejor configuración basándose en un compromiso entre cantidad de instancias resueltas, tiempo que tarda en resolver cada una de ellas, y cuán buenas son las soluciones encontradas en cada una de las instancias que no se han podido resolver dentro del tiempo límite establecido.

4. *Evaluar el impacto de los cortes de propósito general de CPLEX*: CPLEX viene con una batería de algoritmos de plano de corte que pueden ser aplicados a formulaciones en general. Sin embargo, todas las experimentaciones anteriores no incluyen estos cortes para no agregar ruido a los resultados e identificar la contribución real de los cortes particulares propuestos. En esta sección se evalúa la mejora en la resolución de cada instancia cuando se activan estos cortes.

5. *Comparar Branch&Cut con Cut&Branch*: Un último experimento consiste en hacer uso de un nuevo algoritmo que se incorpora a CPLEX llamado dynamic search. Este algoritmo hace uso de todos los threads disponibles pero no permite utilizar cortes personalizados durante su ejecución. Es por esto que suele ser utilizado como Cut&Branch, es decir, se agregan todos los cortes al nodo raíz y luego se resuelve utilizando dynamic search. En esta experimentación se analiza cuán conveniente es resignar el agregado de cortes durante la enumeración con respecto a utilizar otro algoritmo utilizado en la práctica para llevarla a cabo.

6. *Impacto de la solución inicial:* Para intentar determinar cuán beneficioso sería diseñar una heurística primal o una heurística inicial más sofisticada que encuentre mejores soluciones se experimenta resolviendo el modelo con la configuración elegida pero partiendo desde la solución óptima. Esto permite luego comparar con los valores obtenidos partiendo desde la solución heurística y determinar si el cambio es significativo.

6. RESULTADOS

A continuación se presentan los resultados de la experimentación realizada respecto al modelo y las desigualdades presentadas. Los resultados se muestran en el orden en que fueron descriptas las etapas de la experimentación en la sección anterior.

6.1. Calidad de la heurística inicial

El primer experimento consiste en evaluar el funcional de la solución de la heurística presentada en cada instancia. En la Figura 6.1 se muestra la distancia entre cada solución y la óptima de cada instancia.

Se puede ver que en general cuanto mayor es la letra (i.e. $d > c > b > a > 0$) de las instancias, peor es la calidad de la solución heurística. Esto puede explicarse debido a que las instancias fueron generadas en base a las instancias del problema esclavo de un TD-VRPRC. Esto quiere decir que cuanto más avanzado el problema maestro, la heurística comienza a ser menos precisa.

Más adelante se muestra un análisis respecto a cuál es el impacto de comenzar con estas soluciones y si es conveniente invertir tiempo en mejorar este procedimiento.

6.2. Impacto de cortes en el nodo raíz

En esta parte de la experimentación se evalúa cada familia de desigualdades de forma independiente, tal como se describe en el capítulo anterior. A continuación se exhiben los resultados de esta experimentación en los 4 gráficos de la Figura 6.2 donde se presentan las siguientes métricas:

- Figura 6.2a: Mejora del gap luego de terminar de procesar el nodo raíz. Este valor está calculado utilizando la fórmula $\frac{g_0 - g_r}{g_0}$, donde g_0 es el gap de la relajación del modelo con respecto a la solución óptima, y g_r es el gap luego de aplicar los cortes al nodo raíz. Esta métrica permite evaluar la mejora del gap de manera porcentual, dejando afuera las diferencias absolutas ya que distintas instancias pueden variar su gap debido a que el valor absoluto del óptimo está cercano a 0 como ya fue explicado anteriormente.
- Figura 6.2b: Tiempo total, en segundos, que se tarda en resolver el nodo raíz.
- Figura 6.2c: Tiempo promedio del algoritmo de separación en segundos.
- Figura 6.2d: Cantidad de iteraciones de separación.

A partir de estos resultados se puede observar algunos patrones particulares. En primer lugar, el corte más significativo en términos de %rGap es GCS, logrando un promedio de mejora significativo y sin un impacto considerable en el tiempo de cómputo. Esto se corresponde con la efectividad de estas desigualdades en otros problemas similares donde

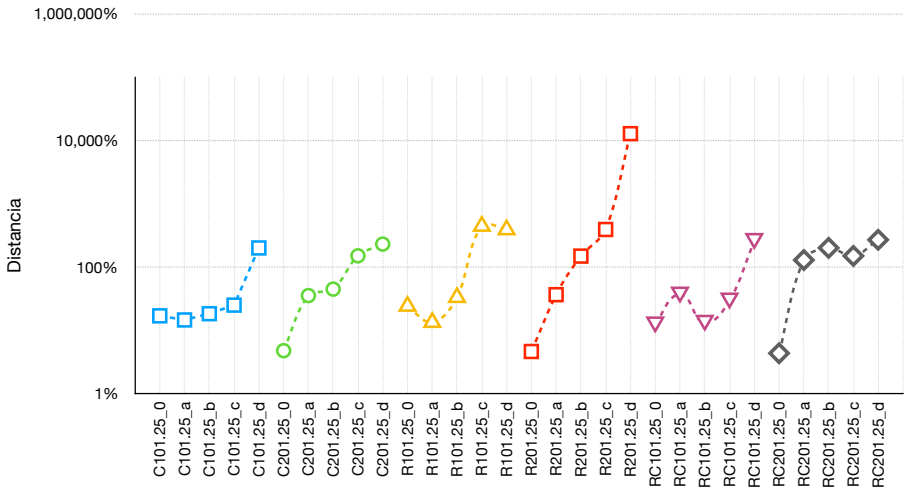
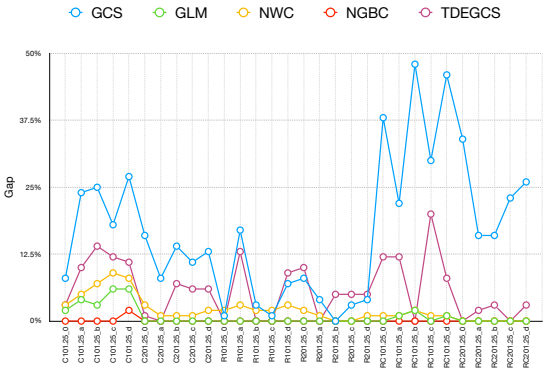
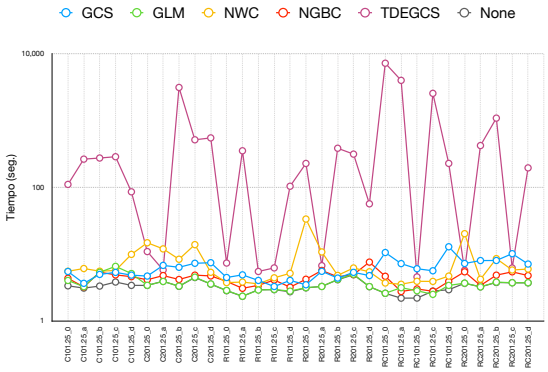


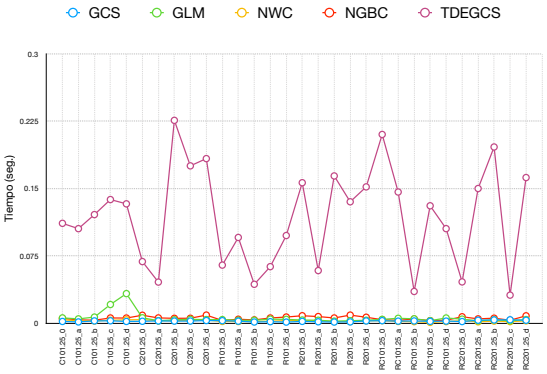
Fig. 6.1: Calidad de la solución heurística en cada grupo de instancias. (Escala logarítmica).



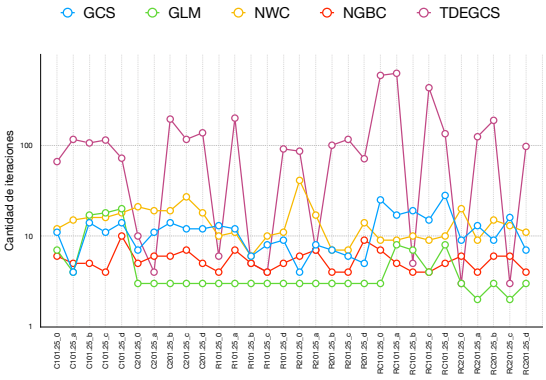
(a) Mejora del Gap luego de procesar el nodo raíz.



(b) Tiempo total, en segundos, de procesamiento del nodo raíz. (Escala logarítmica).



(c) Tiempo promedio, en segundos, de cada iteración de separación. (Escala logarítmica).



(d) Cantidad de iteraciones del algoritmo de separación en el nodo raíz. (Escala logarítmica).

Fig. 6.2: Resultados de experimentación sobre los planos de corte.

se observa un resultado análogo (e.g. Taccari [30]).

Por otra parte, un dato que puede parecer curioso es que la familia TDEGCS logra un impacto menor en términos de Gap que GCS en varias instancias a pesar de ser más general. La explicación de esto se basa en entender el criterio de parada mencionado en la Sección 5.3. Nuestra hipótesis es que si bien TDEGCS es más general, al tener una separación exacta de las desigualdades más violadas puede suceder que estas sean muy especificas de un punto en particular y no ayuden tanto a nivel general. Esto puede causar que cada desigualdad introducida cambie poco el funcional y que haya que agregar un gran número de cortes para lograr un cambio mayor. Esto se corrobora ya que TDEGCS generalmente realiza muchas iteraciones de corte en la Figura 6.2d, y esto provoca tiempos de resolución del nodo raíz demasiado altos. Como consecuencia, se concluye que agregar muchas desigualdades de este tipo no resulta conveniente, al menos con la configuración actual del algoritmo de planos de corte.

Respecto a las desigualdades NWC y NGBC, se ve una mejora leve pero constante en el Gap en la primera familia mencionada, y una considerablemente más grande para la segunda. A su vez, estos cortes son muy rápidos de computar y no agregan tiempo de cómputo excesivo en el nodo raíz.

Por último, la familia de desigualdades GLM introduce una mejora notable en las instancias $C101\_25\_x$ pero no impacta en el resto. Esto sucede debido a que en las instancias propuestas por Dabia et al. [6], de donde fueron construidas aquellas utilizadas en este trabajo, no existe una restricción muy grande por parte del recurso de capacidad de los vehículos en la mayoría de las instancias y en algunas inclusive la capacidad del vehículo permite visitar a todos los clientes. En general el recurso más restrictivo en estas instancias son las ventanas de tiempo. Como GLM busca explotar las demandas de los clientes, al no haber restricciones fuertes asociadas el impacto de este corte se vuelve marginal.

6.3. Elección y analisis de parámetros

Según los resultados obtenidos en la etapa anterior se eligen las siguientes configuraciones de parámetros con mayor potencial. Se representa a cada familia de cortes con una letra: GCS (C), NWC (W), NGBC (G), GLM (L), TDEGCS (T).

1. *None*: No se aplica ningún tipo de corte (tampoco los de propósito general de CPLEX). Se utiliza el algoritmo de CPLEX Dynamic Search.

2. *C*: Solamente se utiliza la familia GCS, sin límite en la cantidad de cortes a agregar.

3. *CWG*: Es igual que *C* pero cuando no encuentra más desigualdades violadas se intenta agregar desigualdades de las familias NWC y NGBC, sin límite por iteración.

4. *CWGL*: Es igual que *CWG* pero al no encontrar ningún nuevo plano de corte se intenta agregar desigualdades de la familia GLM, sin límite por iteración.

5. *CWGT*: Es igual que *CWG* pero análogamente, al no encontrar ninguna desigualdad nueva para agregar se agrega hasta 1 desigualdad por iteración de TDEGCS, y hasta 20 desigualdades en total de toda la ejecución.

Estas elecciones están basadas en el hecho de que GCS es muy importante a la hora de reducir el %rGap, que NWC y NGBC son muy rápidos computacionalmente y ofrecen una mejora del %rGap, además de cortar las variables de dependencia del tiempo x_{ij}^m. Por otra parte, GLM se prueba pero con la hipótesis de no ser muy relevante por lo tanto se agrega exclusivamente con los cortes más prometedores. Por último, TDEGCS son más generales que GCS por lo que pueden ayudar a salir de una solución cuando no hay más desigualdades violadas pero agregar muchas de esta familia puede resultar en ralentizar el algoritmo, por lo tanto se decide limitarlo.

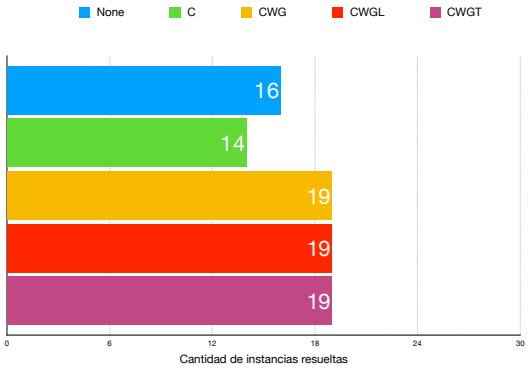
Por otro lado, los demás parámetros tales como branching, criterio de selección de nodo y otros quedan con sus valores prestablecidos. Además, salvo cuando se utiliza el algoritmo Dynamic Search se utiliza un solo thread.

Los resultados obtenidos se encuentran condensados en la Figura 6.3. Estos están compuestos por 6 gráficos, que agregando la información permiten mostrar tendencias respecto al funcionamiento de cada configuración de parámetros.

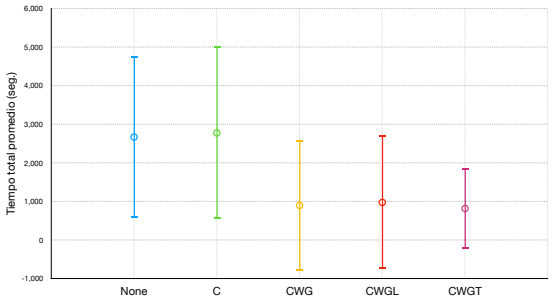
- *Figura 6.3a*: Cantidad de instancias resueltas por cada configuración.
- *Figura 6.3b*: Tiempo total promedio, en segundos, de las instancias resueltas. Se marca además su desvío estándar.
- *Figura 6.3c*: Gap final promedio de las instancias no resueltas. También se muestra su desvío estándar.
- *Figura 6.3d*: Porcentaje del tiempo total invertido en iteraciones de los algoritmos de separación de planos de corte.
- *Figura 6.3e*: Cantidad de nodos resueltos por configuración y separados por instancia.
- *Figura 6.3f*: Igual que la Figura 6.3e pero haciendo foco en los algoritmos que aplicaron cortes para lograr un cambio de escala.

El mensaje principal de los resultados es el impacto positivo de las nuevas familias de desigualdades introducidas, aquellas que atacan a la dependencia del tiempo. Se puede observar que aquellas configuraciones que incluyen las nuevas familias de desigualdades (CWG, CWGL, CWGT) logran resolver más instancias que el resto. Además de esto, el tiempo promedio de ejecución de las instancias resueltas es considerablemente menor, más precisamente, alrededor de un tercio.

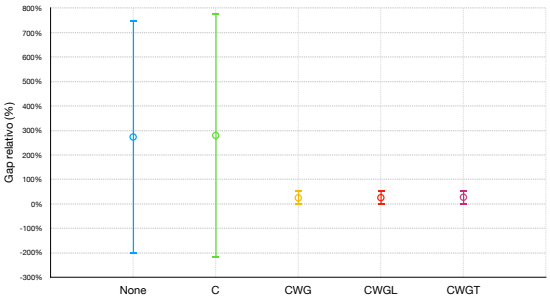
Por otro lado, al analizar las instancias no resueltas se observa que en las configuraciones CWG, CWGL y CWGT el %fGap promedio es también más cercano a 0 %. Además, vemos que la cantidad de nodos sigue un patrón definido: si no se agrega ningún corte la enumeración es muy grande y por eso se resuelven pocas instancias. Por este motivo, la Figura 6.3f muestra el mismo gráfico pero con el foco puesto en el resto de los algoritmos, logrando que se puede apreciar mejor la relación entre ellos. En particular, se observa que al agregar al menos las dos familias de cortes de dependencia del tiempo NWC y NGBC se reduce a la mitad, o incluso más en algunas instancias, la cantidad de nodos enumerados.



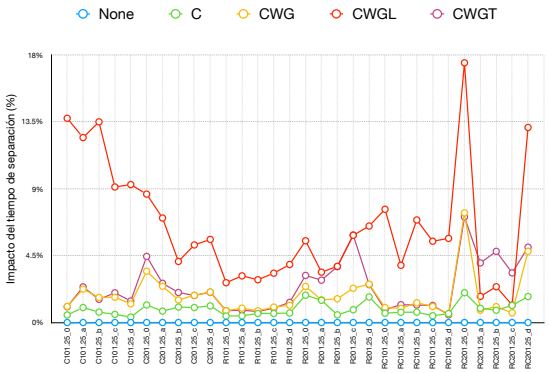
(a) Cantidad de instancias resueltas.



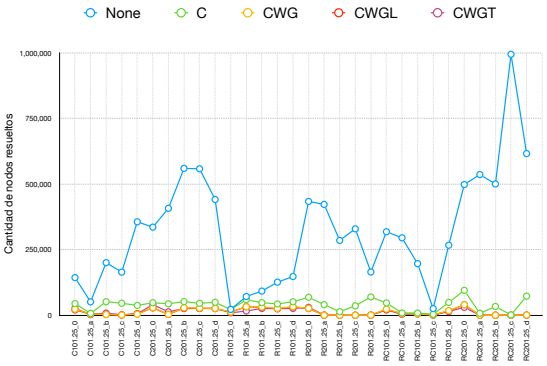
(b) Tiempo total promedio, en segundos, de las instancias resueltas.



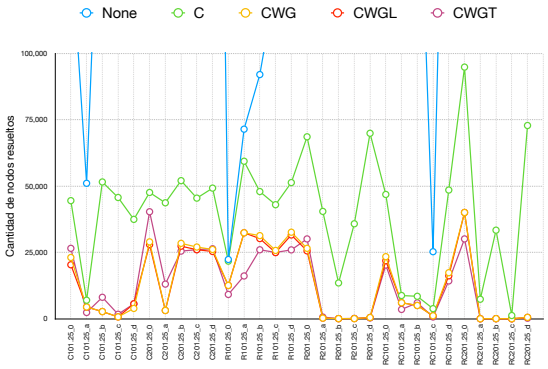
(c) Gap promedio final de las instancias no resueltas.



(d) Porcentaje del tiempo invertido en los algoritmos de separación.



(e) Cantidad de nodos resueltos.



(f) Cantidad de nodos resueltos enfocado en las configuraciones con cortes.

Fig. 6.3: Resultados de experimentación de la formulación.

Por estos motivos, conlcuímos que el utilizar las nuevas familias de desigualdades propuestas es beneficioso para obtener un mejor resultado.

Respecto a la separación, se ve que el costo de la misma es razonable en todos los casos, tomando cerca de un 5 %, excepto con algunas instancias cuando se utiliza el corte GLM. La separación de este corte se realiza utilizando un PLEM y es por esto que puede resultar más lento que las demás. Sin embargo, sabemos que existe un algoritmo lineal en el tamaño del grafo soporte que haría que ese tiempo reduzca como todos los demás.

Para elegir la mejor configuración, nos basamos principalmente en la Figura 6.3b, ya que la performance de los últimos 3 algoritmos es similar en todo el resto de los aspectos. Básicamente, en esta figura notamos que el uso de GLM no tiene un impacto grande en la cantidad de nodos enumerados ni en los tiempos de ejecución. Esto se puede deber a lo mencionado anteriormente, es decir, que las instancias no están muy restringidas por la capacidad del vehículo.

Por otra parte, el uso de TDEGCS parece tener un efecto positivo lo cual se puede deber a que si bien son muy pocas las desigualdades de esta familia que se agregan, las mismas serían suficientes para cambiar la solución óptima de la relajación y permitir que las otras familias de desigualdades puedan seguir generando otras que corten este punto.

Como resultado de esta etapa de experimentación se decide que la configuración CWGT es la más equilibrada y con mejores resultados, y se utiliza de acá en más para el resto del trabajo.

6.4. Impacto de los cortes de CPLEX

A continuación se muestra los resultados obtenidos al habilitar los cortes de CPLEX en la configuración CWGT. En este caso, se comparan los algoritmos: CPLEX con su configuración default (CPLEX), el algoritmo CWGT presentado anteriormente, y CWGT con los cortes de propósito general de CPLEX activados (CWGT+CPLEX).

Al ser únicamente 3 algoritmos los que se comparan se muestra la información relevante en la Tabla 6.1. Por cada algoritmo e instancia se muestra Tiempo, %rGap, %fGap y #Nodos. En el caso del Tiempo Promedio solamente se consideran las instancias resueltas, y en el caso de %fGap Promedio solamente se consideran aquellas que no fueron resueltas.

Los resultados muestran que los cortes de CPLEX ayudan de manera significativa a mejorar el rendimiento de nuestro algoritmo cuando son combinados con los cortes particulares que explotan la dependencia del tiempo. En particular, logra incrementar el número de instancias resueltas a 28 de un total de 30, en lugar de solo 19. Además, se logra mejorar consistentemente el %rGap en las instancias.

Por otra parte, la tabla muestra que las instancias que logran resolver los algoritmos CPLEX y CWGT no son necesariamente las mismas. Por ejemplo, mientras que CWGT logra resolver en menos de 2000 segundos todas las instancias RC201_25, CPLEX no resuelve ninguna de ellas en el tiempo límite. Lo opuesto, sin embargo, sucede con las instancias

	CPLEX				CWGT				CWGT+CPLEX			
Instancia	Tiempo	%rGap	%fGap	#Nodos	Tiempo	%rGap	%fGap	#Nodos	Tiempo	%rGap	%fGap	#Nodos
C101_25_0	4,351.95	5.69	0.00	147,392	7,200.01	3.24	0.78	26562	1,566.38	3.07	0.00	18684
C101_25_a	1,183.29	9.31	0.00	51,049	270.37	2.49	0.00	2366	313.79	2.04	0.00	3319
C101_25_b	190.03	17.20	0.00	19,100	1,530.18	5.18	0.00	8102	409.49	4.81	0.00	3927
C101_25_c	7,200.01	22.54	18.98	239,539	379.52	13.59	0.00	1703	270.65	9.92	0.00	1990
C101_25_d	80.99	304.44	0.00	4,410	1,045.37	169.36	0.00	5634	452.28	119.71	0.00	3997
C201_25_0	7,200.02	3.45	7.31	286,037	7,200.08	2.08	4.67	40360	7,200.00	2.02	1.10	85777
C201_25_a	7,200.68	5.28	27.91	370,199	1,765.95	1.87	0.00	13073	479.60	1.87	0.00	7016
C201_25_b	7,200.87	16.78	62.21	482,177	7,200.04	7.48	4.98	25512	4,074.07	6.94	0.00	48705
C201_25_c	7,200.01	45.79	417.76	402,675	7,200.03	22.49	16.98	26060	6,867.13	19.09	0.00	108736
C201_25_d	7,200.01	91.42	365.88	310,235	7,200.01	48.63	65.18	26408	5,567.17	43.34	0.00	83691
R101_25_0	368.35	14.75	0.00	17,545	2,401.31	13.39	0.00	9172	1,069.77	12.35	0.00	10087
R101_25_a	1,200.98	27.68	0.00	80,859	3,921.83	19.10	0.00	16140	1,547.80	18.69	0.00	22567
R101_25_b	1,281.40	33.51	0.00	104,696	7,200.02	30.55	7.15	26016	2,359.10	30.05	0.00	29215
R101_25_c	1,564.89	316.75	0.00	127,682	7,200.00	289.01	64.72	25022	2,364.69	264.01	0.00	37957
R101_25_d	1,142.34	326.70	0.00	140,260	7,200.02	250.74	81.31	25976	3,221.26	242.81	0.00	47426
R201_25_0	7,204.58	3.76	3.92	274,763	7,200.01	2.99	2.33	30108	5,321.04	2.69	0.00	91758
R201_25_a	7,200.37	4.37	60.54	338,881	188.09	2.23	0.00	648	38.06	2.07	0.00	405
R201_25_b	5,618.18	13.06	0.00	320,802	50.78	12.09	0.00	37	148.24	6.04	0.00	673
R201_25_c	4,185.06	39.66	0.00	222,909	45.41	29.37	0.00	135	30.68	13.74	0.00	123
R201_25_d	3,501.25	1449.72	0.00	217,111	133.69	781.79	0.00	314	44.35	508.70	0.00	394
RC101_25_0	7,200.02	29.59	6.71	231,423	7,200.01	12.35	6.77	20258	7,200.01	11.72	3.81	49411
RC101_25_a	7,202.95	28.65	27.21	105,142	572.61	13.45	0.00	3522	684.27	12.02	0.00	5431
RC101_25_b	3,876.71	46.07	0.00	158,897	1,316.51	15.90	0.00	6010	181.54	14.23	0.00	1502
RC101_25_c	694.57	26.47	0.00	19,009	174.64	9.20	0.00	676	120.84	6.69	0.00	931
RC101_25_d	7,200.01	538.72	1404.43	272,008	7,200.03	171.01	42.92	14267	1,870.75	136.71	0.00	19481
RC201_25_0	7,200.01	3.87	7.10	247,732	1,586.24	1.05	0.00	30112	464.21	1.05	0.00	14952
RC201_25_a	7,200.82	27.07	32.99	533,096	31.53	2.02	0.00	2	19.52	0.00	0.00	9
RC201_25_b	7,200.01	43.14	99.65	323,677	40.61	4.85	0.00	25	33.55	0.00	0.00	5
RC201_25_c	7,201.26	47.35	139.71	472,991	34.79	1.10	0.00	0	13.56	0.00	0.00	5
RC201_25_d	7,200.81	117.85	420.02	385,741	64.51	16.85	0.00	210	94.32	6.32	0.00	750
Promedio:	2,088.57	122.02	193.90	230,268	818.63	65.18	27.07	12814.33	1,415.29	50.09	2.45	23297.46

Tab. 6.1: Impacto de utilizar cortes de CPLEX.

R101_25 en donde CPLEX logra resolver todas con un tiempo menor a 2000 segundos y CWGT solamente resuelve 2 de ellas y con un tiempo mayor. Esto muestra que cada tipo de corte funciona de manera distinta y por lo tanto la combinación de ellos tiene sentido.

Otro patrón que se encuentra en las tablas es la superioridad de %rGap en CWGT frente a CPLEX dado que el primer algoritmo logra acercar más el %rGap a la solución óptima en todas las instancias. A su vez, se observa una segunda mejora cuando se combinan todas las familias de desigualdades logrando la mejor configuración también en lo que respecta al %rGap.

Con respecto a los nodos, se puede ver que CWGT enumera uno o más ordenes de magnitud menos que CPLEX en general. Esto nos da una idea de que los cortes de CWGT son más efectivos a la hora de cortar puntos que los de propósito general de CPLEX tomados de forma independiente. Sin embargo, se observa que al combinar ambos algoritmos no necesariamente se logra reducir la cantidad de nodos. Si bien se utiliza una mayor cantidad de familias de desigualdades válidas para aplicar cortes, CWGT+CPLEX enumera más nodos en varios casos pero tardando menos tiempo en resolver cada uno de ellos. Esto puede ser debido a que al agregar cortes de las familias incluidas en CPLEX se mueve la solución óptima de la relajación a otro punto y causa que menos cortes de los propuestos en este trabajo sean incluidos.

Para corroborar esto tomamos el promedio sobre todas las instancias de desigualdades que se agregan en todo el árbol en cada uno de los dos algoritmos (i.e. CWGT, CWGT+CPLEX). En el caso de CWGT en promedio se agregan 256.8 NWC, 211.03 NGBC, 3536.83 GCS y 19.87 TDEGCS, mientras que en CWGT+CPLEX se incluyen en promedio solamente 160 NWC, 140.53 NGBC, 807.63 GCS y 17.70 TDEGCS. Además, notamos que se agregan 29.40 cortes de propósito general de CPLEX en promedio en cada instancia. Considerando que CWGT agrega una mayor cantidad de cortes al modelo, creemos que resolver la relajación lineal en cada nodo puede llevar más tiempo, lo cual explicaría por qué CWGT+CPLEX puede resolver más nodos en el mismo lapso. Además, otra hipótesis es que los cortes propuestos en este trabajo son densos lo cual ralentizaría aún más la resolución de cada nodo.

Se observa, en resumen, que la combinación entre los cortes de CPLEX y las desigualdades particulares superan el rendimiento que se tiene al utilizar el algoritmo default de CPLEX, lograndose resolver casi el doble de las instancias en un tiempo promedio menor. Este resultado muestra el impacto de las desigualdades propuestas y la ventaja que trae su incorporación al modelo.

6.5. B&C vs C&B

Como se menciona en el capítulo anterior, CPLEX utiliza un algoritmo llamado Dynamic Search cuando no se utilizan cortes personalizados durante la ejecución del algoritmo de Branch&Bound. Este algoritmo tiene un funcionamiento distinto al del tradicional B&C y es por esto que se evalúa cuál es el compromiso de utilizarlo. Este compromiso surge de únicamente poder agregar cortes en el nodo raíz, pero mejorar el tiempo de enumeración.

	CWGT+CPLEX (B&C)			CWGT+CPLEX (C&B)			CWGT+CPLEX (C&B, OPT)		
Instancia	Tiempo	%fGap	#Nodos	Tiempo	%fGap	#Nodos	Tiempo	%fGap	#Nodos
C101_25_0	1566.38	0.00	18684	1139.51	0.00	28420	128.18	0.00	10788
C101_25_a	313.79	0.00	3319	224.05	0.00	7914	11.14	0.00	1372
C101_25_b	409.49	0.00	3927	112.89	0.00	5188	63.73	0.00	6862
C101_25_c	270.65	0.00	1990	470.41	0.00	14855	68.52	0.00	4933
C101_25_d	452.28	0.00	3997	1121.55	0.00	60278	229.17	0.00	25535
C201_25_0	7200.00	1.10	85777	7200.02	2.81	151133	7200.00	0.49	382888
C201_25_a	479.60	0.00	7016	96.82	0.00	7346	37.082	0.00	8915
C201_25_b	4074.07	0.00	48705	1284.29	0.00	81380	825.01	0.00	73332
C201_25_c	6867.13	0.00	108736	4394.35	0.00	452812	970.41	0.00	203719
C201_25_d	5567.17	0.00	83691	7200.15	54.99	155578	2523.57	0.00	265812
R101_25_0	1069.77	0.00	10087	448.51	0.00	15110	75.62	0.00	9266
R101_25_a	1547.80	0.00	22567	551.33	0.00	30537	165.47	0.00	16661
R101_25_b	2359.10	0.00	29215	739.99	0.00	56929	274.19	0.00	34579
R101_25_c	2364.69	0.00	37957	962.71	0.00	76715	659.41	0.00	73032
R101_25_d	3221.26	0.00	47426	848.72	0.00	96017	279.27	0.00	58597
R201_25_0	5321.04	0.00	91758	3869.53	0.00	366403	1754.00	0.00	267871
R201_25_a	38.05	0.00	405	112.06	0.00	3865	12.71	0.00	911
R201_25_b	148.24	0.00	673	282.38	0.00	8722	15.54	0.00	3684
R201_25_c	30.68	0.00	123	48.18	0.00	1060	16.47	0.00	311
R201_25_d	44.35	0.00	394	97.518	0.00	4170	16.11	0.00	2970
RC101_25_0	7200.00	3.81	49411	2560.75	0.00	102615	1242.68	0.00	90819
RC101_25_a	684.27	0.00	5431	269.74	0.00	5946	80.38	0.00	4335
RC101_25_b	181.54	0.00	1502	310.29	0.00	12395	80.74	0.00	5639
RC101_25_c	120.84	0.00	931	69.01	0.00	2607	26.68	0.00	1935
RC101_25_d	1870.75	0.00	19481	2344.22	0.00	162382	540.86	0.00	59113
RC201_25_0	464.21	0.00	14952	643.29	0.00	85506	512.42	0.00	96987
RC201_25_a	19.52	0.00	9	43.29	0.00	1968	13.99	0.00	756
RC201_25_b	33.55	0.00	5	57.31	0.00	1605	19.28	0.00	684
RC201_25_c	13.56	0.00	5	65.40	0.00	1533	16.84	0.00	0
RC201_25_d	94.32	0.00	750	224.08	0.00	8980	29.84	0.00	5176
Promedio:	1,415.28	2.45	23297.47	835.43	28.90	66998.97	368.60	0.49	57249.40

Tab. 6.2: Comparación entre Branch&Cut, Cut&Branch y este último partiendo del óptimo.

En la Tabla 6.2 se observan los resultados de la comparación entre el algoritmo CGWT+CPLEX utilizando la técnica Branch&Cut contra Cut&Branch, es decir, agregando cortes en el nodo raíz y utilizando el algoritmo Dynamic Search. Por otra parte se agrega también el algoritmo CWGT+CPLEX utilizando Cut&Branch pero en lugar de partir de la solución heurística encontrada se toma como solución inicial una de las óptimas del problema.

En la tabla se incluye por cada algoritmo e instancia, las columnas Tiempo, %fGap, y #Nodos. Notar que se excluye el %rGap dado que es exactamente igual en todas porque no cambia los cortes que se agregan en el nodo raíz. Al final de la tabla se agregan los datos mostrando el promedio de cada una de las columnas. Al igual que en la Tabla 6.1 el Tiempo Promedio se considera sobre las instancias resueltas, y el %fGap Promedio solamente se consideran aquellas que no fueron resueltas.

En este caso, ambos algoritmos CWGT+CPLEX y CWGT+CPLEX (CB) resuelven 28 instancias, y tienen resultados similares. La mayor diferencia radica en el tiempo, en promedio, que lleva resolver las instancias donde el algoritmo de Cut&Branch tiene una ventaja significativa. Sin embargo, se ve reflejado en la cantidad de nodos que esta ventaja es fruto del hecho de estar utilizando Dynamic Search, ya que enumera más nodos en un tiempo menor.

Respecto al %fGap, si bien parece tener peores resultados C&B, el %fGap de la instancia C201_25_d hay que mirarlo con cuidado al compararlo con otros, ya que esta instancia tiene un funcional muy cercano a 0 en el óptimo y por lo tanto la distancia relativa a otras soluciones puede ser alta aunque en términos absolutos no lo sea como se mencionó previamente.

Al considerar las instancias resueltas por ambos algoritmos, se puede ver que una de las dos instancias que no resuelven ambos es distinta. Creemos que esto se puede deber a que ambos algoritmos difieren en que uno enumera más nodos y el otro genera mejores cotas inferiores, y cada instancia puede ser más fácil de resolver con alguno de ellos según sus propiedades.

Por otro lado, si bien utilizar C&B tiene mejor tiempo en promedio, hay grupos de instancias que se resuelven más rápido utilizando B&C como por ejemplo las RC201_25. Sin embargo, en otros grupos de instancias sí funciona significativamente mejor C&B. Un ejemplo de esto último son las instancias R101_25.

Resultados similares a los presentados acá, en términos de comparación entre algoritmos de B&C y C&B usando Dynamic Search se pueden encontrar en Montero et al. [26] donde también se observa que al solo agregar cortes en el nodo raíz y luego usar Dynamic Search se logra reducir considerablemente los tiempos.

6.6. Impacto de la solución inicial

Finalmente se exhiben los resultados de la ejecución del mejor algoritmo experimentado hasta el momento, es decir, CWGT con los cortes de CPLEX habilitados, y utilizando la técnica de Cut&Branch, pero partiendo de la solución óptima en cada instancia.

En la Tabla 6.2 se observa que partir de la solución óptima incide muy positivamente en la eficiencia del algoritmo. Respecto al tiempo de ejecución promedio de las instancias resueltas podemos ver una reducción del alrededor del 60 % cuando se elige la solución óptima como inicial. Por otro lado se logra resolver una instancia más y además se reduce el %fGap promedio aproximadamente un 98 %.

A partir de estos resultados se puede concluir que una mejora en la heurística puede ser muy beneficiosa para mejorar la performance del algoritmo. Esto abre una línea de trabajo futuro para el desarrollo de una heurística inicial o primal, que es un aspecto a mejorar del algoritmo y que puede tener incidencia también a la hora de resolver la generación de columnas de VRPRC, logrando evitar que se tenga que ejecutar el algoritmo exacto para algunas columnas.

7. CONCLUSIONES Y TRABAJO A FUTURO

En esta tesis se aborda el problema del TD-PESPPRC mediante un algoritmo exacto basado en técnicas de Programación Lineal Entera. Para ello, se toma como punto de partida la formulación propuesta por Sun et al. [29], que es mejorada a fin de obtener un nuevo modelo más ajustado. Se proponen tres nuevas familias de desigualdades válidas que explotan explícitamente la estructura dependiente del tiempo de la red de transporte subyacente, y se estudian los correspondientes problemas de separación. Las mismas son incorporadas a un algoritmo de plano de cortes junto con otras familias tomadas de problemas relacionados, que luego son estudiadas experimentalmente en detalle a fin de obtener evidencia respecto a su performance.

Los resultados obtenidos muestran que la mejor combinación de parámetros es capaz de resolver 28 de las 30 instancias de 25 clientes consideradas en un lapso de 2 horas.

Uno de los desafíos de este problema es que la cantidad de variables y restricciones que se modelan son elevadas ya para un grafo de tamaño chico. Sin embargo, se mostró que mediante la utilización de familias de desigualdades se puede ajustar bastante bien la relajación y lograr resolver en muchos casos las instancias en tiempos razonables. En particular, las desigualdades NWC, NGBC y TDEGCS mostraron ser muy útiles en combinación con las ya conocidas GCS ya que atacan la dependencia del tiempo, es decir, la relación entre variables x_{ij}^m, lo cual las GCS no consideran.

Finalmente, esta tesis deja varias líneas de investigación a futuro abiertas, entre ellas:

1. *Mejorar heurística inicial*: En el capítulo 6, se mostró la importancia de comenzar a enumerar nodos partiendo de una buena cota primal, es decir, una solución inicial. La heurística presentada en esta tesis es bastante simple y su propósito fue el de proveer algún punto de inicio. Esta heurística puede ser mejorada mediante el uso de meta-heurísticas como GRASP o Tabu-Search.

2. *Analizar TDEGCS*: Estas desigualdades son la versión generalizada de las GCS que tienen en cuenta la dependencia del tiempo. Al ser tan generales resultan ineficientes de ser agregadas todas aquellas que son violadas. Es por esto que resulta importante determinar qué conjunto de desigualdades es el más importante y cómo separarlas eficientemente.

3. *Generalizar a otros problemas dependientes del tiempo*: Las desigualdades propuestas son válidas para TD-PESPPRC, sin embargo, intuitivamente pueden ser generalizadas para otros problemas que también utilicen variables x_{ij}^m. Una vía de desarrollo es intentar ver cómo se comportan las familias de desigualdades en otros contextos dado que atacan una característica central como la dependencia temporal de la red de transporte.

Apéndice

A. INSTANCIAS TD-PESPPRC

Instancia	Profits
C101_25_0	[0, 1179, 1147.2, 1117.34, 1144.34, 1151.67, 1216.67, 1166.67, 1262, 1300, 1125.45, 1264, 1470, 1413.34, 1686, 1500, 1383.6, 1349.55, 1429.5, 1368, 1085.71, 1068.33, 1101.67, 1075.5, 1102.5, 1180.43, 0]
C101_25_a	[0, 975.54, 1077.98, 1059.3, 1044.31, 1151.66, 1216.66, 763.02, 710.57, 539.25, 763.97, 1192.45, 988.89, 1413.33, 947.55, 1142.62, 999.37, 1020.09, 636.31, 989.6, 1085.71, 1068.33, 1101.66, 795.2, 1102.5, 752.53, 0]
C101_25_b	[0, 943.1, 1105.21, 1117.34, 930.74, 945.25, 598.12, 911.4, 921.11, 1101.57, 951.17, 969.71, 983.45, 1243.98, 926.46, 1044.94, 978.28, 1189.44, 868.78, 902.42, 922.5, 1068.33, 970.8, 926.05, 1102.5, 915.75, 0]
C101_25_c	[0, 1005.97, 1069.12, 988.57, 966.97, 970.97, 972.97, 981.45, 995.25, 995.61, 981.57, 1004.85, 1051.07, 1074.49, 445.45, 1105.67, 1100.92, 935.48, 1054.01, 1055.62, 998.43, 982.23, 965.96, 1026.94, 975.85, 1021.61, 0]
C101_25_d	[0, 948.01, 954.91, 931.51, 933.61, 935.41, 933.11, 941.11, 945.11, 956.11, 924.07, 963.69, 1048.58, 987.69, 962.71, 1009.45, 1000.26, 901.63, 1013.28, 977.46, 961.14, 937.87, 915.63, 917.73, 943.27, 968.27, 0]

Tab. A.1: Instancias C101 para el TD-PESPPRC

Instancia	Profits
C201_25_0	[0, 1315.5, 1147.2, 1291.5, 1245.59, 1151.66, 1320, 1348.33, 1132, 1366, 1125.45, 1264, 1470, 1413.33, 1686, 1500, 1383.6, 1349.55, 1429.5, 1368, 1085.71, 1093.33, 1101.66, 1311.75, 1102.5, 1345.71, 0]
C201_25_a	[0, 1315.5, 1147.2, 1291.5, 1245.59, 1151.67, 1320, 1348.33, 547.1, 1366, 920.83, 1264, 1284.91, 895.21, 983.9, 1084.34, 799.33, 712.09, 842.8, 915.49, 1085.71, 1093.33, 1101.67, 1311.75, 1102.5, 765.3, 0]
C201_25_b	[0, 1057.43, 1058.61, 1291.5, 945, 1079.43, 1320, 1010.12, 936.5, 1041.91, 892.14, 972.24, 953.42, 908.04, 1225.19, 901.85, 900.09, 923.24, 1033.58, 972.84, 897.57, 1093.33, 1061, 1028.49, 967.32, 1032.16, 0]
C201_25_c	[0, 1112.75, 981.38, 913.36, 972.43, 907.83, 1140.25, 949.92, 1014.73, 969.12, 882.95, 1036.89, 960.84, 950.07, 686.08, 1042.15, 1081.63, 1034.34, 1027.93, 863.68, 960.17, 911.04, 937.32, 969.6, 887.97, 973.14, 0]
C201_25_d	[0, 1038.7, 973.68, 940.8, 948.76, 919.78, 1053.35, 1015.97, 1001.35, 942.61, 849.03, 1032.8, 956.78, 1004.17, 881.62, 1052.52, 992, 966.73, 918.21, 951.22, 973.45, 918.99, 925.25, 973.36, 979.08, 961.11, 0]

Tab. A.2: Instancias C201 para el TD-PESPPRC

Instancia	Profits
R101_25_0	[0, 328, 316, 401.05, 437.5, 443.33, 285, 453.33, 820, 965, 442.9, 751.25, 325, 285, 965, 638.4, 449.2, 510.4, 337, 484, 764.95, 400, 582.4, 586, 505, 751.25, 0]
R101_25_a	[0, 328, 316, 318.39, 437.5, 255.46, 285, 453.33, 223.5, 414.57, 223.48, 306.53, 325, 285, 356.4, 529.5, 364.1, 510.4, 337, 484, 280.26, 179.56, 203.83, 272.61, 325.56, 254.53, 0]
R101_25_b	[0, 234.06, 159.56, 300.73, 281.91, 222.36, 188.36, 222.08, 97.15, -132.14, 342.12, 347.83, 433.51, 254.8, 359.12, 194.93, 104, 218.91, 298.62, 0, 322.89, 334.43, 87.56, 552.08, 505, 0, 0]
R101_25_c	[0, 203.22, 166.16, 220.11, 221.81, 189.11, 163.41, 209.62, 206.08, 238.54, 167.02, 414.6, 228.26, 133.07, 353.57, 216.43, 138.35, 197.3, 245.2, 65.27, 259.7, 212.9, 211.6, 224.27, 238.44, 220.14, 0]
R101_25_d	[0, 203.83, 98.58, 231.22, 219.51, 189.11, 159.75, 220.12, 193.84, 247.1, 136.08, 206, 225.96, 200.65, 250.79, 239.11, 206.23, 213.16, 257.43, 294.31, 250.53, 202.6, 209.7, 243.35, 225.04, 217.84, 0]

Tab. A.3: Instancias R101 para el TD-PESPPRC

Instancia	Profits
R201_25_0	[0, 328, 316, 401.05, 437.5, 443.33, 285, 453.33, 624, 740, 442.9, 722.14, 325, 285, 740, 606.66, 449.2, 510.4, 337, 484, 686.85, 400, 548.33, 586, 505, 722.14, 0]
R201_25_a	[0, 328, 138.76, 401.05, 272.2, 443.33, 246.33, 202.92, 239.4, 283.51, 442.9, 405.14, 297.75, 285, 249.17, 165.23, 319.92, 510.4, 209.29, 133.68, 165.8, 241.24, 256.6, 461.78, 249.8, 325.5, 0]
R201_25_b	[0, 328, 245.55, 179.15, 437.5, 214.24, 212.01, 239.2, 100.06, 222.12, 193.27, 56.3, 309.06, 185.44, 347.68, 61.54, 99.9, 158.85, 260.83, 338.14, 65.55, 202.85, 273.98, 81.61, 143.53, 30.08, 0]
R201_25_c	[0, 204.23, 139.64, 173.39, 159.44, 172.19, 210.73, 178.64, 129.89, 269.58, 102.07, 218.87, 216.7, 207.27, 247.44, 219.6, 148.13, 203.85, 239.63, 332.82, 115.56, 226.04, 186.24, 214.55, 202.21, 178.91, 0]
R201_25_d	[0, 203.01, 117.74, 155.35, 193.79, 149.28, 178.36, 218.08, 150.81, 241.18, 160, 288.73, 208.18, 201.16, 297.16, 213.07, 130.26, 183.88, 192.36, 124.45, 183.73, 227.33, 131.54, 225.45, 225.7, 173.02, 0]

Tab. A.4: Instancias R201 para el TD-PESPPRC

Instancia	Profits
RC101_25_0	[0, 670, 469.6, 630.54, 586, 800.8, 741.4, 720.8, 1145, 1007.5, 526.6, 756.25, 584.5, 764, 1052.5, 783.2, 594.4, 644.04, 776.5, 580, 835, 962, 716, 707.5, 572.5, 961, 0]
RC101_25_a	[0, -454.41, 469.6, 278.43, 122.35, 800.8, 0, 0, 811.62, 450.59, 171.01, 474.93, 584.5, -40.37, 493.65, 279.8, 594.4, -243.2, 776.5, -157.85, -142.66, 500.96, 680.16, 707.5, -443.3, 545.31, 0]
RC101_25_b	[0, 393.15, 5.14, 131.94, 447.35, 699.53, 110, 0, 241.28, 250.78, 187.96, 216.37, 235.08, 323.92, 246.76, 209.36, 235.08, 430.26, 500.37, 186.94, 291.58, 165.48, 315.53, 264.27, 98.69, 377.88, 0]
RC101_25_c	[0, 615.56, 116.63, 68.8, 485.13, 43.35, -7.95, 182.29, 0, 359.7, 215.54, 285, 412.3, 89.34, 0, 0, 93.83, 169.81, 776.5, 580, -245.13, 0, 69.52, 247.22, 562.47, 0, 0]
RC101_25_d	[0, 549.58, 127.47, 114.12, 128.14, 128.35, 118.02, 159.01, 137.51, 344.41, 136.93, 179.5, 149.74, 178.66, 86.66, 191.35, 185.69, 189.8, 481.6, 128.52, 145.27, 100.02, 85.33, 249.99, 180.99, 161.8, 0]

Tab. A.5: Instancias RC101 para el TD-PESPPRC

Instancia	Profits
RC201_25_0	[0, 670, 469.6, 630.54, 586, 771.66, 655, 688.33, 880, 770, 526.6, 722.14, 584.5, 733.33, 806, 753.33, 594.4, 644.04, 776.5, 580, 750, 850, 683.33, 707.5, 572.5, 939.42, 0]
RC201_25_a	[0, 524.5, 207.69, -3.65, 391.89, 94.3, 217.74, -21.85, 0, 417.59, 87, 170.91, 459.8, 39.44, 0, 79.02, 197.21, 234.71, 776.5, 188.6, 77.5, 0, 103.6, 114.4, 108.4, 118, 0]
RC201_25_b	[0, 442.62, 163.53, 136.42, 264.7, -73.94, 116.5, 213.67, 147.12, 365.32, 159.55, 164.5, 165.55, 260.22, -82.85, 153.69, 192.18, 262.81, 472.05, 348.67, -90.48, 0, 124.92, 98.5, 286.42, 0, 0]
RC201_25_c	[0, 270.29, 163.7, 367, 170.5, -11.03, 134.67, 213.85, 101.65, 68.37, 72.87, 233.37, 183.52, 306.98, 0, 106.93, 224.18, 277.75, 163.75, 394.32, 398.55, -11.48, 216.77, 98.67, 146.66, 0, 0]
RC201_25_d	[0, 289.91, 168.58, 115.77, 149.84, 192.65, 134.67, 235.6, 123.61, 109.86, 189.47, 124.1, 177.28, 241.28, 45.51, 151.98, 269.19, 188.79, 184.4, 373.45, 196.31, 157.57, 101.41, 134.99, 236.72, 0, 0]

Tab. A.6: Instancias RC201 para el TD-PESPPRC

Instancia	Costo	Camino
C101_25_0	-4,587.17	0 17 18 13 19 15 14 12 11 9 4 1 5 26
C101_25_a	-2,005.46	0 21 20 24 22 13 15 19 11 6 4 3 5 26
C101_25_b	-1,151.78	0 21 24 22 13 17 10 9 4 2 1 3 5 26
C101_25_c	-747.57	0 21 20 24 25 23 10 11 9 6 2 1 3 5 26
C101_25_d	-58.98	0 20 18 19 15 14 16 12 11 9 4 3 5 26
C201_25_0	-8,311.96	0 21 8 10 11 5 2 1 7 3 4 9 15 12 14 16 19 17 18 13 25 23 6 24 20 22 26
C201_25_a	-3,986.00	0 4 3 7 1 2 5 11 10 9 15 14 12 13 23 6 24 21 20 22 26
C201_25_b	-1,601.90	0 21 22 24 6 23 25 13 12 14 16 19 17 18 9 10 11 5 2 1 4 3 7 26
C201_25_c	-490.42	0 20 22 6 23 17 18 16 12 15 25 11 1 2 26
C201_25_d	-263.87	0 4 3 7 1 2 5 11 16 12 15 19 13 25 23 6 24 22 20 8 26
R101_25_0	-4,046.79	0 6 14 16 8 19 11 10 20 9 3 26
R101_25_a	-1,702.54	0 6 13 2 15 14 16 17 19 11 7 26
R101_25_b	-1,026.83	0 21 4 12 24 3 1 20 10 7 18 26
R101_25_c	-70.50	0 6 5 16 14 15 22 23 25 4 12 26
R101_25_d	-67.43	0 12 4 25 23 22 15 14 16 5 6 13 26
R201_25_0	-7,444.62	0 6 5 17 16 14 15 2 21 22 23 25 4 12 24 3 1 9 20 10 11 19 8 18 7 13 26
R201_25_a	-2,527.10	0 6 14 16 17 5 8 19 11 10 9 1 3 24 12 4 25 23 22 21 2 13 26
R201_25_b	-804.15	0 1 3 12 4 13 2 21 22 14 6 5 19 10 7 18 26
R201_25_c	-220.00	0 6 5 17 16 14 15 22 21 12 3 9 1 11 19 8 18 13 26
R201_25_d	-8.43	0 21 12 4 25 24 3 9 20 1 7 10 11 19 8 17 16 14 15 2 13 26
RC101_25_0	-6,795.48	0 24 23 25 21 18 20 22 14 15 9 13 26
RC101_25_a	-3,560.45	0 2 8 5 12 18 21 25 23 22 26
RC101_25_b	-1,349.78	0 4 1 3 5 8 12 14 17 16 15 13 26
RC101_25_c	-1,797.49	0 1 4 12 10 9 24 18 19 26
RC101_25_d	-117.58	0 24 25 23 18 9 15 7 1 2 26
RC201_25_0	-13,196.97	0 24 22 20 19 18 21 23 25 13 9 10 11 12 14 17 16 15 7 6 8 5 3 1 4 2 26
RC201_25_a	-1,353.99	0 6 4 1 2 12 17 16 11 10 9 24 18 19 26
RC201_25_b	-1,031.59	0 2 4 1 3 8 6 7 15 16 17 12 11 10 9 13 19 18 24 22 26
RC201_25_c	-867.41	0 24 22 20 19 13 12 11 17 16 7 6 4 3 1 2 26
RC201_25_d	-384.37	0 24 19 23 21 18 20 13 9 10 11 12 17 16 15 7 6 8 5 3 1 4 2 26

Tab. A.7: Soluciones óptimas de las instancias TD-PESPPRC

Bibliografía

[1] Hernán Abeledo, Ricardo Fukasawa, Artur Pessoa, and Eduardo Uchoa. The time dependent traveling salesman problem: polyhedra and algorithm. *Mathematical Programming Computation*, 5(1):27–55, September 2012.

[2] José Albiach, José María Sanchis, and David Soler. An asymmetric tsp with time windows and with time-dependent travel times and costs: An exact solution through a graph transformation. *European Journal of Operational Research*, 189(3):789 – 802, 2008.

[3] Anna Arigliano, Gianpaolo Ghiani, Antonio Grieco, and Emanuela Guerriero. Time dependent traveling salesman problem with time windows: Properties and an exact algorithm. Technical report.

[4] Vasek Chvátal, editor. *Linear Programming.* W. H. Freeman, 1983.

[5] Jean-François Cordeau, Gianpaolo Ghiani, and Emanuela Guerriero. Analysis and branch-and-cut algorithm for the time-dependent travelling salesman problem. *Transportation Science*, 48(1):46–58, 2012.

[6] Said Dabia, Stefan Ropke, Tom van Woensel, and Ton De Kok. Branch and price for the time-dependent vehicle routing problem with time windows. *Transportation Science*, 47(3):380–396, 2013.

[7] Sanjeeb Dash, Oktay Günlük, Andrea Lodi, and Andrea Tramontani. A time bucket formulation for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(1):132–147, 2012.

[8] Michael Drexl. A note on the separation of subtour elimination constraints in elementary shortest path problems. *European Journal of Operational Research*, 229:595 – 598, 2013.

[9] M. Fischetti, G. Laporte, and S. Martello. The delivery man problem and cumulative matroids. *Oper. Res.*, 41(6):1055–1064, 1993.

[10] Fabio Furini, Martin Philip Kidd, Carlo Alfredo Persiani, and Paolo Toth. Improved rolling horizon approaches to the aircraft sequencing problem. *J. Scheduling*, 18(5):435–447, 2015.

[11] Michel Gendreau, Gianpaolo Ghiani, and Emanuela Guerriero. Time-dependent routing problems: A review. *Computers & Operations Research*, 64:189–197, 2015.

[12] Maria Teresa Godinho, Luis Gouveia, and Pierre Pesneau. Natural and extended formulations for the Time-Dependent Traveling Salesman Problem. *Discrete Applied Mathematics*, 164:138–153, February 2014.

[13] Luis Gouveia and S Voß. A classification of formulations for the (time-dependent) traveling salesman problem. *European Journal of Operational Research*, 2217(93), 1995.

[14] Arthur V Hill and WC Benton. Modelling intra-city time-dependent travel speeds for vehicle scheduling problems. *Journal of the Operational Research Society*, pages 343–351, 1992.

[15] Yixiao Huang, Lei Zhao, Tom Van Woensel, and Jean-Philippe Gross. Time-dependent vehicle routing problem with path flexibility. *Transportation Research Part B: Methodological*, 95:169 – 195, 2017.

[16] Soumia Ichoua, Michel Gendreau, and Jean-Yves Potvin. Vehicle dispatching with time-dependent travel times. *European journal of operational research*, 144(2):379–396, 2003.

[17] Mads Kehlet Jepsen and David Pisinger. *Branch-and-cut and Branch-and-Cut-and-Price Algorithms for Solving Vehicle Routing Problems*. PhD thesis, Technical University of Denmark, 2011.

[18] N Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[19] A. Lucena. Time-dependent traveling salesman problem—the deliveryman case. *Networks*, 20(6):753–763, 1990.

[20] Chryssi Malandraki and Mark S. Daskin. Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. *Transportation Science*, 26(3):185–200, 1992.

[21] Penélope Aguiar Melgarejo, Philippe Laborie, and Christine Solnon. A time-dependent no-overlap constraint: Application to urban delivery problems. In *Integration of AI and OR Techniques in Constraint Programming*, pages 1–17. Springer, 2015.

[22] I Méndez-Dıaz, JJ Miranda-Bront, P Toth, and P Zabala. Infeasible path formulations for the time-dependent tsp with time windows. In *10 th Cologne-Twente Workshop on Graphs and Combinatorial Optimization CTW 2011*, pages 198–202, 2011.

[23] Isabel Méndez-Díaz, Paula Zabala, and Abilio Lucena. A new formulation for the traveling deliveryman problem. *Discrete Appl. Math.*, 156(17):3223–3237, October 2008.

[24] Juan José Miranda-Bront. *Integer Programming approaches to the Time Dependent Travelling Salesman Problem*. PhD thesis, Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires, 2012.

[25] Juan José Miranda-Bront, Isabel Méndez-Díaz, and Paula Zabala. Facets and valid inequalities for the time-dependent travelling salesman problem. *European Journal of Operational Research*, May 2013.

[26] Agustín Montero, Isabel Méndez-Díaz, and Juan José Miranda-Bront. An integer programming approach for the time-dependent traveling salesman problem with time windows. *Computers Operations Research*, 88:280 – 289, 2017.

[27] Jean-Claude Picard and Maurice Queyranne. The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research*, 26(1):86–110, 1978.

[28] Gabriella Stecco, Jean-François Cordeau, and Elena Moretti. A branch-and-cut algorithm for a production scheduling problem with sequence-dependent and time-dependent setup times. *Comput. Oper. Res.*, 35(8):2635–2655, 2008.

[29] Peng Sun, Lucas P. Veelenturf, Said Dabia, and Tom Van Woensel. The time-dependent capacitated profitable tour problem with time windows and precedence constraints. *European Journal of Operational Research*, 2017.

[30] Leonardo Taccari. Integer programming formulations for the elementary shortest path problem. *European Journal of Operational Research*, 252(1):122–130, jul 2016.

[31] Paolo Toth and Daniele Vigo, editors. *Vehicle Routing: Problem, Methods and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2014.