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

Un algoritmo branch-and-price-and-cut para diseño de redes con p-ciclos

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

Remberto Emanuel Delgadillo Cárdenas

Directora de tesis: Dra. Irene Loiseau

Consejera de estudios: Dra. Paula Zabala

Lugar de trabajo: Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires

Buenos Aires, 2021

UN ALGORITMO BRANCH-AND-PRICE-AND-CUT PARA DISEÑO DE REDES CON P-CICLOS

El concepto de redes de p-ciclos se introdujo a fines de los años 90 en el contexto de redes ópticas supervivientes. Una red de comunicaciones se dice superviviente si puede continuar brindando servicio pese a la falla de alguno de sus componentes. Las topologías basadas en p-ciclos combinan las mejores características para asegurar la supervivencia: velocidad en la recuperación y poca capacidad redundante, lo que implica menor costo. Un p-ciclo es un ciclo preconfigurado formado por un canal de reserva en cada enlace que lo compone. Cada p-ciclo protege a todos los enlaces que forman el ciclo y también a cada enlace que no es parte del ciclo pero cuyos nodos sí lo son.

A partir de la necesidad de diseñar redes basadas en p-ciclos de costo mínimo, surgieron varios problemas de optimización combinatoria, de los cuales el más elemental es el problema de Asignación de Capacidad de Reserva (*Spare Capacity Allocation* - SCA). En este problema se tiene una red con demandas asociadas a cada enlace previamente asignadas. Se debe determinar la disposición de la capacidad de reserva mediante la ubicación de p-ciclos de forma que quede garantizada la recuperación de las comunicaciones ante la falla de alguno de sus enlaces. Cada enlace adicional de reserva incrementa el costo de la solución, que debe ser minimizado.

En este trabajo desarrollamos un algoritmo *branch-and-price-and-cut* para SCA. Para eso presentamos una nueva formulación como problema de programación lineal entera, la cual que tiene una estructura diagonal en bloque usual en este tipo de problemas. A partir de esta formulación aplicamos la descomposición Dantzig-Wolfe para obtener el problema maestro y el problema de *pricing*. Con la descomposición eliminamos el inconveniente de la simetría proveniente de la estructura diagonal de la formulación original, aunque también mostramos que el problema de *pricing* resultante es $\mathcal{NP}$-hard. Detallamos las modificaciones necesarias que permiten aplicar reglas de *branching* y planos de corte en el problema maestro sin modificar la estructura del problema de *pricing* en la mayoría de los casos, y realizando modificaciones poco significativas en otros. Resolvemos las instancias de *pricing* de forma exacta, y también proponemos heurísticas para esto, con el objetivo de acelerar el proceso de generación de columnas.

Para la experimentación contamos con instancias correspondientes a redes reales para comparar nuestro algoritmo con trabajos previos y generamos instancias más grandes para evaluar los distintos parámetros. Los resultados obtenidos mostraron ser muy superadores respecto a trabajos anteriores para este problema.

Palabras claves: p-ciclos, redes supervivientes, programación lineal entera, generación de columnas, heurísticas.

A BRANCH-AND-PRICE-AND-CUT ALGORITHM FOR P-CYCLE NETWORK DESIGN

The p-cycle networking concept was introduced in the late 90's in the context of survivable optical networks. A communication network is said to be survivable if it keeps operating although any of its components fails. P-cycles based topologies gather the best features to ensure survivability: restoration speed and low spare capacity, which implies a low cost. A p-cycle is a preconfigured cycle composed of a single channel of spare capacity on each link it crosses. Each p-cycle protects all the links that belong to the cycle and also each link that is not part of the cycle but its two end nodes are.

Due to the necessity of designing p-cycle networks of minimum cost, several combinatorial optimization problems were defined. The basic problem of those is the Spare Capacity Allocation problem - SCA. Its input has a network with previously assigned working demands associated to each link. The goal is to organize the spare capacity of the network into a set of p-cycles so that communication restoration is guaranteed in case of any single link failure. Each additional spare link increases solution cost, that has to be minimized.

We developed a *branch-and-price-and-cut* algorithm for SCA. We introduce a new MIP formulation for it, which has the typical diagonal block structure found in this kind of problems. We apply Dantzig-Wolfe decomposition to this formulation to get the Master Problem and the Pricing Problem. After decomposition, the symmetry issue due to the diagonal block structure is eliminated, although the Pricing Problem is shown to be $\mathcal{NP}$-hard. We detail all modifications needed to apply branching rules and cutting planes on the Master Problem without modifying the Pricing Problem structure in most cases, and making little modifications in others. Pricing instances are solved by an exact algorithm (branch-and-cut), and we also introduce heuristics to finish the column generation process as early as possible.

For computational experiments, we have a set of real-world network instances to compare our algorithm with previous works. We also generated larger instances to evaluate the introduced features of the algorithm. Results show that the developed *branch-and-price-and-cut* algorithm outperforms previously presented works found in literature for SCA.

Keywords: p-cycles, survivable networks, integer linear programming, column generation, heuristics.

Índice general

Capítulo 1

INTRODUCCIÓN

1.1. Redes de comunicaciones, supervivencia y p-ciclos

La industria de las telecomunicaciones ha sido siempre una fuente interesante de problemas de optimización combinatoria (Resende & Pardalos, 2006). La mayoría de ellos tienen aplicaciones importantes que tienen relevancia para la economía y la sociedad. Muchos de ellos son computacionalmente difíciles de resolver.

Las redes de fibra óptica se han convertido en una infraestructura importante en todo el mundo. Casi todas las comunicaciones requieren la transmisión a través de una red troncal implementada en fibra óptica. Por ese motivo, las fallas en esos sistemas implican impactos económicos, personales, y sociales negativos.

En caso de una falla accidental, como un corte en un enlace de fibra óptica o la caída de un nodo de la red, es necesario que el sistema sea capaz de recuperarse a sí mismo tan pronto como sea posible para minimizar la pérdida de datos. Una red se dice superviviente si tiene esta habilidad. La supervivencia se logra redirigiendo tráfico a través de una parte distinta de la red que tiene recursos de reserva destinados a ser usados en esa situación.

La capacidad de los enlaces es un recurso clave porque limita la cantidad de datos que pueden ser transmitidos en un período determinado de tiempo. Se debe asignar capacidad de reserva para lograr supervivencia, es decir, más canales de comunicación que los necesarios para satisfacer la demanda prevista. Distintos sistemas (y tecnologías) administran la capacidad de reserva de formas distintas pero se pueden distinguir dos enfoques comunes: protección en malla y anillos.

Para redes en malla, la capacidad de reserva se asigna de tal manera que es posible la recuperación usando pocos canales adicionales. Existe poca o nula configuración predeterminada, siendo más importante el bajo uso de recursos de reserva. Cuando ocurre una falla, el sistema de protección (centralizado o distribuido) tiene que tomar las acciones necesarias para redirigir el tráfico asignado del enlace dañado hacia otras partes de la red. Debido a que la capacidad de reserva no se asigna de un modo simple y existen muchos caminos alternativos, el proceso de recuperación podría demandar el tiempo suficiente para ser notado por los usuarios o para perder información.

Por otro lado, en los sistemas basados en anillos cada enlace es una parte de una estructura de anillo de la red, lo cual significa que existe una camino alternativo trivial que va en el sentido opuesto en el anillo. Por lo tanto, se lleva a cabo una simple acción cuando falla un solo enlace, logrando de ese modo un proceso de recuperación muy rápido. La clara desventaja es la alta cantidad de capacidad de reserva necesaria para construir este tipo de sistema. En la literatura específica de redes supervivientes (Grover & Stamatelakis, 2000) se menciona que es frecuente necesitar más de un 100 % de capacidad extra para lograr proteger toda la red, contra un máximo de 70 % requerido por sistemas supervivientes basados en topología de malla. Por este motivo se consideran que no son eficientes en relación a la capacidad.

Como resumen, se tiene una dicotomía entre estos sistemas donde las ventajas de uno

son las desventajas del otro y viceversa:

Sistema	Costo	Tiempo de recuperación
Malla	Bajo	Lento
Anillos	Alto	Rápido

Es en este contexto en que surge el esquema de protección por p-ciclos, que se propuso a fines de los noventa en (Grover & Stamatelakis, 1998). El nombre p-ciclo refiere a ciclo pre-configurado pre-conectado de protección. La idea es asignar la capacidad de reserva formando ciclos en la red, al igual que en sistemas basados en anillos, pero con *hardware* adicional para proteger cada enlace cuyos nodos pertenezcan al ciclo, no solo los enlaces que lo forman. Cada enlace forma parte del ciclo tiene un camino alternativo para redirigir el tráfico, como ocurre con redes de anillos. Mientras que, adicionalmente, cada cuerda del ciclo (enlaces que no forman parte del p-ciclo pero sus dos nodos sí) tiene dos caminos alternativos. (Ver figuras 1.1 y 1.2). Estos se conocen como protección *on-span* y *straddle* respectivamente en la literatura referida a p-ciclos. Esta característica simple y distinta (protección de cuerdas del ciclo) permite un sistema de recuperación más eficiente respecto a la capacidad, mientras que se mantiene casi tan eficiente en tiempo de recuperación como un sistema de anillos.

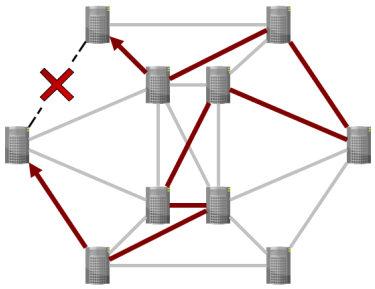


Fig. 1.1: Protección *on-span*, al igual que con anillos.

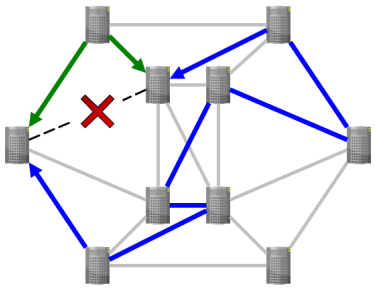


Fig. 1.2: Protección de cuerdas, una mejora para lograr eficiencia de capacidad.

La topología y ruteo de la capacidad de trabajo es independiente de la protección por p-ciclos. Los p-ciclos se configuran para detectar y recuperar cada falla de enlace

simple. Por falla simple se entiende que en un instante determinado puede fallar un único enlace. El enlace puede estar constituido por varios canales de comunicación, cada uno de ellos protegido por diferentes p-ciclos, posiblemente de composición diferente. Desde que fueron propuestos se presentaron diferentes formas de implementarlos, y también se presentaron diferentes aplicaciones dependiendo del contexto de fallas. De este modo se definieron varios problemas de optimización combinatoria relacionados con diseño de redes con p-ciclos.

1.1.1. *Spare Capacity Allocation* - SCA

El problema de asignación de capacidad de reserva (SCA por su nombre en inglés: *Spare Capacity Allocation*) es el problema básico que involucra el diseño de una red usando p-ciclos. Su objetivo es proteger una red contra una falla simple de uno de sus enlaces usando p-ciclos a costo mínimo. La entrada del problema es un grafo $G = (V, E)$ que representa la red. Ya que cada eje $e \in E$ tiene que pertenecer al menos a un ciclo o ser cuerda de un ciclo, G es 2-conexo (sin puntos de articulación). Cada eje tiene una demanda y un costo. La demanda es el número de canales de trabajo que fueron previamente asignados a ese eje, y que se perderán en caso de que falle, y es por lo tanto un entero no negativo. El costo es un precio a pagar por asignar un canal de comunicación de reserva en ese eje. Es un número no negativo, pero no necesariamente entero.

El costo de un ciclo es la suma de los costos de todos sus ejes. Una solución factible para SCA es un multiconjunto [1] de ciclos de G tal que la suma de los canales de protección de cada eje es mayor o igual a su demanda. Una solución óptima es por lo tanto un multiconjunto de ciclos que satisface la condición previa a costo mínimo.

La figura 1.3 ilustra una instancia y su solución. No se muestran los costos por simplicidad.

Cuando se trata de redes reales de gran escala, es frecuente limitar la longitud de los p-ciclos debido a la degradación de la señal causada principalmente por ruido. Se propusieron estas alternativas:

- Límite de saltos: es una cota superior del número de nodos del ciclo, independientemente de su costo o longitud.
- Límite de circunferencia: se asigna un peso a cada eje, y la suma de los pesos de los ejes de cada ciclo debe ser menor o igual a un valor predefinido. Usualmente este peso es el mismo costo, o el costo multiplicado por alguna constante.
- Una combinación de ambas: por ejemplo, una cota para el límite de circunferencia más el límite de saltos multiplicado por un factor predeterminado.

Para mayor detalle sobre esto, ver (Kodian et al., 2004).

1.1.2. Asignación conjunta de capacidad de trabajo y de reserva

Otro problema definido luego de la introducción de p-ciclos es la asignación conjunta de capacidad de trabajo y de reserva con p-ciclos, JCA por su nombre en inglés: *Joint -working and spare- capacity allocation* (Doucette & Grover, 2002). La estrategia es optimizar la elección de rutas de demanda al mismo tiempo de la asignación de p-ciclos para capacidad

[1] Es un multiconjunto porque puede tener elementos (ciclos) repetidos.

de reserva. El objetivo es minimizar la capacidad total. Respecto a los datos de entrada, se tiene un grafo $G = (V, E)$ como red y un costo por asignar un canal en cada enlace (eje) $e \in E$. Un canal puede ser destinado a tráfico o para reserva. Por otro lado, la demanda en el contexto de JCA es el número requerido de canales de comunicación (entero no negativo) entre cada par de nodos.

1.1.3. Otros problemas relacionados

Se pueden encontrar varios problemas de diseño de redes con p-ciclos en (Grover et al., 2006) y (Asthana et al., 2010) que fueron definidos poco tiempo después de la introducción de p-ciclos.

Algunas aplicaciones más recientes incluyen más detalles técnicos del dominio de las telecomunicaciones. Ejemplos de ellos son p-ciclos multicast y redes de líneas de velocidad mixta.

En el contexto de p-ciclos multicast (Smutnicki & Walkowiak, 2012; Panayiotou et al., 2016), se considera un subconjunto de nodos de V tal que cada uno de sus nodos es una fuente idéntica de los mismos datos. Por lo tanto, un p-ciclo tiene un modo alternativo de protección al incluir un nodo de ese subconjunto.

El diseño de redes de líneas de velocidad mixta (Drid et al., 2012) es una generalización de la asignación de capacidad de reserva en el contexto de redes heterogéneas donde diferentes canales pueden tener diferentes tasas de transferencia de datos. Entonces se debe decidir la velocidad de línea del p-ciclo además de su ubicación. A mayor tasa de transferencia de datos, el costo es mayor y la longitud física del ciclo tiene un umbral menor debido a la degradación de la señal.

1.2. Trabajos previos

1.2.1. Asignación de Capacidad de Reserva con p-ciclos

Los primeros algoritmos presentados para SCA se centran en heurísticas de dos etapas. La primera de ellas consiste en seleccionar un subconjunto de todos los ciclos del grafo, al que denominan conjunto de ciclos candidatos o elegibles. En la segunda etapa se procede a construir una solución a partir de ese subconjunto, usualmente con un algoritmo goloso que selecciona un ciclo por iteración hasta satisfacer todas las demandas, o bien de forma exacta. Desde la introducción del concepto de p-ciclos, se encuentra en la literatura una formulación definida en base a un conjunto de ciclos candidatos.

Sea P un conjunto de ciclos candidatos, se definen C_p como el costo del ciclo $p \in P$ y, para cada eje $e \in E$ y ciclo $p \in P$,

$$y_e^p = \begin{cases} 1 & \text{si } e \text{ es eje del ciclo } p \\ 2 & \text{si } e \text{ es cuerda del ciclo } p \\ 0 & \text{caso contrario} \end{cases}$$

Sea d_e la demanda del enlace $e \in E$ y sea la variable entera λ_p la cantidad de copias del ciclo candidato $p \in P$.

El modelo para SCA que considera solo el conjunto de candidatos P es el siguiente (Grover & Stamatelakis, 1998; Grover et al., 2006).

$$\min \sum_{p \in P} C_p \lambda_p \tag{1.1}$$

$$\text{s.t.} \sum_{p \in P} y_e^p \lambda_p \geq d_e \qquad \forall e \in E \tag{1.2}$$

$$\lambda_p \in \mathbb{Z}_+ \qquad \forall p \in P \tag{1.3}$$

Si P es el conjunto completo de ciclos del grafo, la cantidad de variables de este modelo es exponencial.

Ya sea para la selección de candidatos o para una construcción golosa de solución, en (Grover & Doucette, 2002) se definen criterios para evaluar ciclos de forma heurística. Dos de ellos son el *score* topológico (TS) y la eficiencia *a-priori* (AE). Incluimos su definición ya que serán citados nuevamente en el desarrollo de este trabajo. Entonces:

$$TS_p = \sum_{e \in E} y_e^p d_e \qquad \forall p \in P \tag{1.4}$$

$$AE_p = \frac{TS_p}{C_p} \qquad \forall p \in P \tag{1.5}$$

Ejemplos de algoritmos golosos que usan estos criterios se presentan en (Doucette et al., 2003) y (Zhang & Yang, 2002).

En (Liu & Ruan, 2004) se presenta un algoritmo basado en DFS para enumerar ciclos priorizando aquellos con alta eficiencia *a priori* y cada ciclo corto que protege cada eje. Otra heurística golosa se presenta en (Lo et al., 2007).

Por otro lado, se presentaron modelos de Programación Entera Mixta en (Schupke, 2004; Wu et al., 2010). En total son cuatro formulaciones y todas tienen una cantidad polinomial tanto de variables como de restricciones.

En (Pecorari, 2016) fueron propuestos varios modelos y algoritmos para SCA. Esto incluye algoritmos exactos basados en los modelos de Programación Entera Mixta presentados, heurísticas, generación de columnas y algoritmos basados en programación por restricciones. Además (Pecorari, 2016) introduce instancias interesantes correspondientes a redes ópticas reales que utilizaremos en este trabajo para comparar resultados.

1.2.2. Problemas relacionados

En (Mak & Thomadsen, 2006) se presenta una formulación y estudio poliedral para el problema del Viajante de Comercio Selectivo Cuadrático (QSTSP) que si bien no tiene relación directa con la bibliografía de p-ciclos, la formulación es muy similar a la que utlizaremos para desarrollar nuestro algoritmo, si se considera un único ciclo.

Encontramos algoritmos basados en generación de columnas para el problema de asignación conjunta de capacidad de trabajo y de reserva (JCP) (Rajan & Atamtürk, 2003, 2004; Atamtürk & Rajan, 2008). En estos trabajos se tiene como hipótesis que los p-ciclos son dirigidos y presentan un enfoque en el cual se utiliza generación de columnas hasta obtener la cota dual y luego continúan con un algoritmo *branch-and-cut*.

Los trabajos más recientes en los que se incluyen más detalles técnicos en la formulación como los mencionados (Smutnicki & Walkowiak, 2012; Panayiotou et al., 2016; Drid et al.,

2012) solo presentan la resolución de muy pocas instancias (reales) pequeñas por medio de programación entera, en algunos casos solo de forma heurística usando un enfoque de ciclos candidatos.

1.3. Estructura de este documento

En este trabajo nos proponemos desarrollar un algoritmo eficiente para el problema de Asignación de Capacidad de Reserva (SCA), que pueda ser adaptado para resolver otros problemas derivados.

El capítulo 2 es una breve revisión teórica para el desarrollo de algoritmos de Generación de Columnas.

En el capítulo 3 se presenta una formulación compacta para el problema de Asignación de Capacidad de Reserva (SCA) y luego una nueva formulación con restricciones de *subtour* generalizadas a partir de la cual se aplica la descomposición de Dantzig-Wolfe.

La principal contribución de este trabajo, un algoritmo *branch-and-price-and-cut* para el problema de Asignación de Capacidad de Reserva (SCA), será detallado en el capítulo 4 y en el capítulo 5, donde se presenta un algoritmo exacto y heurísticas para resolver el problema de *pricing* de SCA.

En el capítulo 6 se presenta un conjunto exhaustivo de experimentos computacionales y sus resultados.

Finalmente, las conclusiones y trabajo futuro completarán este documento en el capítulo 7.

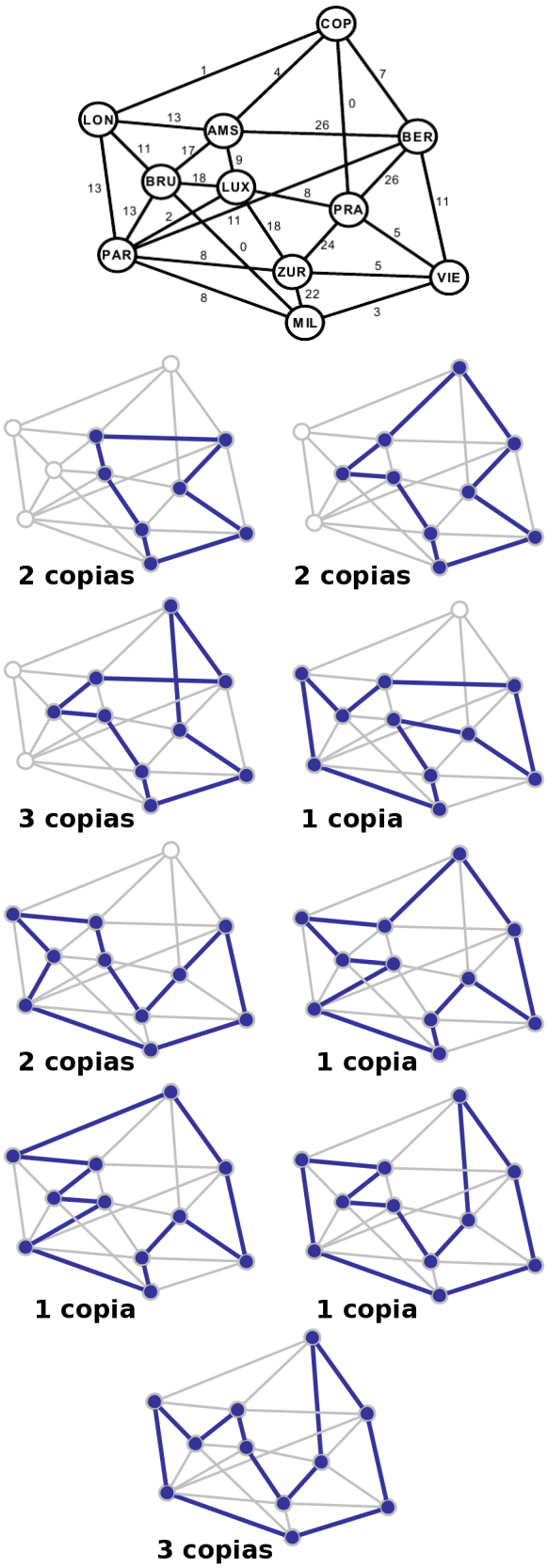


Fig. 1.3: Red con canales de demanda y su solución.

Capítulo 2

PRELIMINARES

Para resolver de forma exacta un problema de Programación Lineal Entera o Entera Mixta es común desarrollar un algoritmo *branch-and-bound* en el cual las relajaciones lineales proveen cotas duales. *Branch-and-bound* implica recorrer un árbol de búsqueda para enumerar de forma implícita las soluciones del problema. Cada nodo del árbol corresponde a un subconjunto de las soluciones, y en cada uno de ellos se resuelve una relajación lineal. En el nodo raíz se tiene la relajación lineal de la formulación del problema que representa todas sus soluciones. Para dividir el problema representado por un nodo se introducen restricciones generando nuevos nodos (*branching*) que serán procesados más adelante. Se obtienen cotas primales con soluciones factibles (enteras), ya sea por relajaciones lineales cuya solución es entera o por medio de heurísticas. Se realizan podas en el árbol de búsqueda utilizando estas cotas o bien por infactibilidad de la relajación lineal de un nodo. Además, para implementar el algoritmo se deben definir las estrategias de *branching* y de selección de nodos. Una descripción completa del método se encuentra en (Wolsey, 1988). *Branch-and-cut* es una generalización de *branch-and-bound* donde se incorporan desigualdades válidas para fortalecer las relajaciones lineales.

Descomponer y reformular problemas de programación lineal entera o (entera mixta) son enfoques clásicos para obtener mejores relajaciones y reducir simetría. A menudo implican el agregado en forma dinámica de variables (columnas) o filas (restricciones) al modelo. Cuando la relajación lineal en cada nodo del árbol de *branch-and-bound* se resuelve con generación de columnas, se denomina *branch-and-price*. Opcionalmente, se pueden agregar planos de corte de modo de fortalecer la relajación y en este caso se conoce como *branch-and-price-and-cut*.

Tanto realizar *branching* en variables con valor fraccionario como fortalecer la formulación con planos de corte interfieren con la generación de columnas y pueden perjudicar las ventajas de la descomposición en caso de hacerse de un modo trivial. Si bien con frecuencia es posible definir una formulación de un modo *ad-hoc* para aplicar generación de columnas (como ocurre con el problema tratado en este trabajo), mantener el vínculo con las variables de la formulación compacta puede ayudar a evitar este inconveniente.

En este capítulo repasaremos algunas definiciones básicas para luego hacer lo propio con la resolución de problemas de Programación Lineal Entera via descomposición de Dantzig-Wolfe y generación de columnas. Para mayor detalle puede verse la bibliografía consultada en (Lübbecke & Desrosiers, 2004; Vanderbeck, 2005; Vanderbeck & Savelsbergh, 2006; Vanderbeck & Wolsey, 2010; Nemhauser & Wolsey, 1988).

2.1. Definiciones

El objetivo es resolver el siguiente problema de Programación Lineal Entera. (Por simplicidad se considera el conjunto X como entero).

$$\begin{aligned} \min \quad & c^t x \\ \text{s.t.} \quad & Ax \geq b \\ & x \in X \end{aligned} \tag{2.1}$$

donde $X \subseteq \mathbb{Z}^n$ es un conjunto discreto que puede ser modelado como un conjunto de puntos que satisface un conjunto de desigualdades lineales.

$$X = P \cap \mathbb{Z}^n \text{ y } P = \{x \in \mathbb{R}^n_{\geq 0} : Dx \geq d\}$$

Definición 1. *Un poliedro $P \subseteq \mathbb{R}^n$ es la intersección de un número finito de semiespacios. En otras palabras existe $D \in \mathbb{R}^{n \times m}$ tal que $P = \{x \in \mathbb{R}^n : Dx \geq d\}$*

Definición 2. *Un poliedro P es una formulación de X si $X = P \cap \mathbb{Z}^n$.*

Si P^1 y P^2 son dos formulaciones de X y $P^1 \subset P^2$ se dice que P^1 es una formulación más fuerte que P^2 ya que

$$\min\{cx : x \in P^1\} \geq \min\{cx : x \in P^2\} \quad \forall c \in \mathbb{R}^n$$

y por lo tanto la cota inferior obtenida por la relajación lineal con la formulación P^1 es siempre mayor o igual que la provista por P^2.

Definición 3. *Dado $X \subseteq \mathbb{Z}^n$, la cápsula convexa de X, notada como conv(X), es el conjunto convexo más pequeño que contiene a X.*

La cápsula convexa de un conjunto entero X es un poliedro. Por lo tanto la formulación más fuerte posible para X es la provista por la cápsula convexa, ya que para toda formulación P' de X se tiene que $\text{conv}(X) \subseteq P'$. Además $\min\{cx : x \in \text{conv}(X)\} = \min\{cx : x \in X\} \quad \forall c \in \mathbb{R}^n$.

Definición 4. *Dado un poliedro no vacío P.*

- *$x \in P$ es un punto extremo de P si y solo si $x = \lambda x^1 + (1-\lambda)x^2$, $0 < \lambda < 1$, $x^1 \in P$, $x^2 \in P$ implica que $x = x^1 = x^2$.*
- *r es un rayo de P si $r \neq 0$ y $x \in P$ implica que $x + \mu r \in P$ para todo $\mu \in \mathbb{R}_{\geq 0}$.*
- *r es un rayo extremo de P si r es un rayo de P y $r = \mu_1 r^1 + \mu_2 r^2$, $\mu_i > 0$ $(i = 1, 2)$, r^1, r^2 rayos de P implica que $r^1 = \alpha r^2$ para algún $\alpha > 0$.*

Teorema 1 (Minkowski)**.** *Sea $P = \{x \in \mathbb{R}^n : Dx \geq d\}$ un poliedro y $\{x_g\}_{g \in G}$ los puntos extremos de P y $\{x_r\}_{r \in R}$ los rayos extremos de P. Entonces P puede representarse de la forma*

$$P = \left\{ x \in \mathbb{R}^n : x = \sum_{g \in G} x_g \lambda_g + \sum_{r \in R} x_r \lambda_r, \quad \sum_{g \in G} \lambda_g = 1, \quad \lambda \in \mathbb{R}^{|G|+|R|}_{\geq 0} \right\} \tag{2.2}$$

En general, el número de puntos extremos y rayos extremos de $\text{conv}(X)$ no es polinomial respecto a la cantidad de variables y restricciones requeridas inicialmente para describir X. Lo mismo ocurre con la cantidad de desigualdades requeridas para describir $\text{conv}(X)$.

2.2. Descomposición Dantzig-Wolfe

La descomposición de Dantzig-Wolfe (Dantzig & Wolfe, 1960) en programación lineal consiste en una forma especial de redefinición de variables y resulta muy útil aplicada en programación lineal entera.

Consideramos el problema definido previamente 2.1 en el que $X = P \cap \mathbb{Z}^n$ y $P = \{x \in \mathbb{R}^n_{\geq 0} : Dx \geq d\}$ es un poliedro.

Una posible motivación para aplicar la descomposición se da cuando resolver un problema de optimización en el conjunto X es relativamente fácil y esto no puede ser aprovechado en la formulación 2.1. En este contexto llamamos a 2.1 como el problema *original* y a x como las *variables originales*.

El enfoque clásico de descomposición consiste en la aplicación del teorema de Minkowski para reemplazar $\text{conv}(X)$ por su definición en función de su conjunto de puntos y rayos extremos. Este enfoque es denominado como de *convexificación* por algunos autores (Vanderbeck, 2000). Al substituir 2.2 por x y aplicar las transformaciones lineales $c_j = cx_j$ y $a_j = Ax_j$ con $j \in P \cup R$ se obtiene una formulación equivalente que será el *Problema Maestro*. Notamos esta descomposición como **DWc**.

$$\min \quad \sum_{g \in G} c_g \lambda_g + \sum_{r \in R} c_r \lambda_r \tag{2.3}$$

$$\text{s.t.} \quad \sum_{g \in G} a_g \lambda_g + \sum_{r \in R} a_r \lambda_r \geq b \tag{2.4}$$

$$\sum_{g \in G} \lambda_g = 1 \tag{2.5}$$

$$\lambda \geq 0 \tag{2.6}$$

$$x = \sum_{g \in G} x_g \lambda_g + \sum_{r \in R} x_r \lambda_r \tag{2.7}$$

$$x \in \mathbb{Z}^n_{\geq 0} \tag{2.8}$$

A diferencia del enfoque clásico que consiste en reformular $\text{conv}(X)$, el enfoque introducido por (Vanderbeck, 2000) denominado como *discretización* es una reformulación de X en sí mismo y se basa en el siguiente teorema (Nemhauser & Wolsey, 1988).

Teorema 2. *Sean $P = \{x \in \mathbb{R}^n : Dx \geq d\}$ un poliedro no vacío y $X = P \cap \mathbb{Z}^n$. Entonces existen un conjunto finito de puntos enteros $\{x_g\}_{g \in G} \subset X$ y un conjunto finito de rayos enteros $\{x_r\}_{r \in R}$ de P tal que*

$$X = \left\{ x \in \mathbb{R}^n_{\geq 0} : x = \sum_{g \in G} x_g \lambda_g + \sum_{r \in R} x_r \lambda_r, \quad \sum_{g \in G} \lambda_g = 1, \quad \lambda \in \mathbb{Z}^{|G|+|R|}_{\geq 0} \right\} \tag{2.9}$$

Aplicar este resultado como una substitución que notamos **DWd** lleva al siguiente Problema Maestro.

$$\min \quad \sum_{g \in G} c_g \lambda_g + \sum_{r \in R} c_r \lambda_r \tag{2.10}$$

$$\text{s.t.} \quad \sum_{g \in G} a_g \lambda_g + \sum_{r \in R} a_r \lambda_r \geq b \tag{2.11}$$

$$\sum_{g \in G} \lambda_g = 1 \tag{2.12}$$

$$\lambda \in \mathbb{Z}_{\geq 0}^{|G|+|R|} \tag{2.13}$$

Ambos enfoques proveen la misma cota inferior (Vanderbeck & Savelsbergh, 2006). En el caso especial que X es acotado y $X \subseteq \{0,1\}^n$ (x variables binarias) **DWc** y **DWd** coinciden. Esto no es poco frecuente y surge en muchas aplicaciones, por ejemplo las descomposiciones que dan lugar a problemas de *set-covering* o *set-partition*. En (Vanderbeck & Savelsbergh, 2006) se puede consultar además el caso general de descomposición Dantzig-Wolfe de problemas de programación lineal entera mixta.

2.2.1. Relajación lineal

Las relajaciones lineales de **DWc** y **DWd** son equivalentes. Esto se cumple porque en **DWc** no se requiere que $x \in \mathbb{Z}^n$ por lo que esa restricción simplemente se ignora. En el caso de **DWd** se relaja el requisito de $\lambda \in \mathbb{Z}^n$. Por lo tanto, en ambos casos la formulación para la relajación lineal es la siguiente.

$$\min \quad \sum_{g \in G} c_g \lambda_g + \sum_{r \in R} c_r \lambda_r \tag{2.14}$$

$$\text{s.t.} \quad \sum_{g \in G} a_g \lambda_g + \sum_{r \in R} a_r \lambda_r \geq b \tag{2.15}$$

$$\sum_{g \in G} \lambda_g = 1 \tag{2.16}$$

$$\lambda \geq 0 \tag{2.17}$$

2.3. Generación de columnas

La solución de la relajación lineal de DW es una cota inferior para el problema 2.1. Como se mencionó anteriormente, esta formulación con frecuencia tiene una cantidad de variables exponencial. Por este motivo, se resuelve inicialmente con un subconjunto de variables y se van incorporando otras dinámicamente hasta finalizar su resolución.

A partir de ahora asumiremos que X es acotado, así que el conjunto de rayos R de P es vacío y solo consideramos el conjunto G para determinar las columnas de la formulación. Notamos como G^i para $i \in \mathbb{Z}_{\geq 0}$ un subconjunto de G tal que para todo $t \in \mathbb{Z}_{\geq 0}$, $G^i \subset G^{i+1}$. De esta manera entendemos por i a una iteración del algoritmo de generación de columnas, y por esto $i \leq |G|$.

La formulación que se obtiene reemplazando G por G^i es el Problema Maestro Restringido.

$$\min \quad \sum_{g \in G^i} c_g \lambda_g \tag{2.18}$$

$$\text{s.t.} \quad \sum_{g \in G^i} a_g \lambda_g \geq b \tag{2.19}$$

$$\sum_{g \in G^i} \lambda_g = 1 \tag{2.20}$$

$$\lambda \geq 0 \tag{2.21}$$

El dual de este problema es:

$$\max \quad \pi b + \sigma \tag{2.22}$$

$$\text{s.t.} \quad \pi a_g + \sigma \leq c_g \quad \forall g \in G^i \tag{2.23}$$

$$\pi \geq 0 \tag{2.24}$$

$$\sigma \in \mathbb{R} \tag{2.25}$$

donde π es la variable correspondiente a 2.19 y σ a la restricción 2.20.

Sean λ' y (π', σ') soluciones del problema maestro restringido y su dual respectivamente. Entonces, para cada i:

- El valor de la solución $\sum_{g \in G^i} c_g \lambda_g = \pi b + \sigma$ es una cota superior para la relajación lineal del problema original. Además si λ' es entera, es una solución factible (cota superior) para el problema original entero.
- El costo reducido asociado a una variable λ_g es $c_g - \pi' a_g - \sigma'$. Para cada variable incluida en G^i el costo reducido es mayor o igual a cero. El algoritmo de generación de columnas se basa en encontrar $g \notin G^i$ tal que su costo reducido sea negativo. Esto se plantea como un problema de minimización, al que llamamos problema de *pricing*.

$$\zeta_i = \min_{g \in G}((cx_g) - \pi'(Ax_g) - \sigma') \tag{2.26}$$

- Si la solución de este problema es mayor o igual a 0, el problema maestro está resuelto y finaliza la generación de columnas, teniendo al valor de la solución $\sum_{g \in G^i} c_g \lambda_g$ como una cota inferior para el problema original entero.
- En caso de existir al menos una variable con costo reducido negativo, el problema no está resuelto y la variable debe ser incorporada al conjunto G^{i+1} para la siguiente iteración. Para esto no es estrictamente necesario hallar el óptimo del problema de pricing ζ_i, si bien suele ser conveniente.
- Desde el punto de vista del dual, si $\zeta_i < 0$ entonces la solución del dual restringido (π', σ') no es factible para el dual original. Por esto, el problema de *pricing* también

puede verse como la búsqueda de una restricción violada en el dual original. Notar que a partir de ζ_i se puede obtener una solución factible del dual no restringido, que es $(\pi', \sigma' + \zeta_i)$, ya que la variable σ es irrestricta. Por la definición del problema de *pricing* se tiene $\pi' a_g + \sigma' + \zeta_i \leq c_g \ \forall g \in G$. Esto implica que el valor de la solución $\pi' b + \sigma' + \zeta_i$ es una cota inferior del problema maestro original.

2.3.1. Estructura diagonal y subsistemas idénticos

La descomposición DW suele aplicarse en casos donde se tiene una estructura diagonal con K subproblemas:

$$\min \left\{ \sum_{k=1}^{K} \sum_{g \in G_k} c_g \lambda_{kg} : \sum_{k=1}^{K} \sum_{g \in G_k} a_g \lambda_{kg} \geq b, \ \sum_{g \in G_k} \lambda_{kg} = 1 \quad \forall 1 \leq k \leq K, \ \lambda \geq 0 \right\} \tag{2.27}$$

Si los subproblemas son idénticos para $k = 1, \ldots, K$ el modelo admite muchas representaciones diferentes para la misma solución: cualquier permutación de índices k define una solución simétrica. En estos casos es frecuente introducir variables agregadas $\nu_g = \sum_{k=1}^{K} \lambda_{kg}$ para obtener la siguiente reformulación.

$$\begin{aligned}
\min \quad & \sum_{g \in G^*} c_g \nu_g && (2.28)\\
\text{s.t.} \quad & \sum_{g \in G^*} a_g \nu_g \geq b && (2.29)\\
& \sum_{g \in G^*} \nu_g = K && (2.30)\\
& \nu \geq 0 && (2.31)
\end{aligned}$$

donde $G^* = G^1 = \cdots = G^K$.

2.4. Planos de corte y *branching*

Agregar desigualdades válidas ayuda a fortalecer la relajación lineal tal como ocurre con algoritmos *branch-and-cut*. Sin embargo, surgen dificultades ya que es necesario tener en cuenta los coeficientes de esas desigualdades en el problema de *pricing*, y además deben ser modificadas al agregar variables.

En (Desaulniers et al., 2011) se presentan posibles enfoques para aplicar. Reproducimos el caso en que se introducen desigualdades válidas expresadas en función de las variables originales.

Sea $Fx \geq f$ un conjunto de desigualdades válidas para el problema 2.1. Entonces, la siguiente formulación tiene las mismas soluciones enteras.

$$\begin{aligned}
\min \quad & c^t x \\
\text{s.t.} \quad & Ax \geq b \\
& Fx \geq f \\
& x \in X
\end{aligned} \tag{2.32}$$

Como resultado de la descomposición, reemplazamos las nuevas desigualdades por

$$\sum_{g \in G} f_g \lambda_g \geq f \tag{2.33}$$

con la transformación lineal $f_g = Fx_g$ para $g \in G$. Las variables duales α asociadas a las nuevas desigualdades se incorporan al problema de *pricing*.

$$\min_{g \in G}((cx_g) - \pi(Ax_g) - \alpha(Fx_g) - \sigma)$$

De esta manera solo se necesita actualizar la función objetivo del problema de *pricing* al agregar desigualdades válidas de este tipo.

Respecto al *branching*, a la dificultad de mantener el problema de *pricing* sin modificaciones significativas se suma la necesidad de mantener un árbol de búsqueda balanceado, que no es posible realizando el *branching* en una simple variable del problema maestro. En (Vanderbeck & Wolsey, 2010) se describen posibles enfoques y en (Vanderbeck, 2011) se propone un esquema genérico. Uno de los enfoques posibles es también utilizar las variables originales, aunque pueden surgir dificultades cuando se cuenta con subproblemas idénticos. El presente trabajo es un ejemplo de ese caso.

Capítulo 3

ASIGNACIÓN DE CAPACIDAD DE RESERVA (SCA) CON P-CICLOS

En este capítulo se presentará formalmente el problema SCA incluyendo dos modelos de programación entera mixta y la descomposición de Dantzig-Wolfe de uno de ellos que será la base para desarrollar el algoritmo de generación de columnas.

Datos de entrada del problema

- Grafo $G = (V, E)$ 2-eje-conexo que representa la red.

$c_{(u,v)}$: costo de agregar una unidad de capacidad al eje $(u, v) \in E$

$d_{(u,v)}$: canales de trabajo (demanda) en eje $(u, v) \in E$

A lo largo de este trabajo asumiremos que el grafo de entrada no tiene puntos de articulación, aunque no es un requerimiento necesario. Esto se debe a que en caso de existir un punto de articulación es posible descomponer el grafo en componentes biconexas para las cuales existe solución de SCA individualmente. De este modo, además, se simplifica el problema ya que se tienen instancias de menor tamaño.

Proposición 1. *El problema de asignación de capacidad de reserva con p-ciclos es $\mathcal{NP}$-hard.*

Demostración. Mostraremos una reducción en tiempo polinomial de MWECCP (*Minimum Weight Edge Cycle Cover Problem*) a SCA. La demostración de que la versión de decisión de este problema es $\mathcal{NP}$-complete puede encontrarse en (Thomassen, 1997).

La entrada para MWECCP es un grafo $G = (V, E)$ con pesos en sus ejes c_e ($\forall e \in E$). La solución es un conjunto de ciclos de costo mínimo tal que cada eje en el grafo pertenece al menos a un ciclo del conjunto.

Definimos un grafo $G' = (V', E')$ del siguiente modo.

- Por cada eje $e \in E$, se introduce un nuevo vértice w_e, entonces $V' = V \cup \bigcup_{e \in E} \{w_e\}$.
- Definimos el conjunto de ejes $E' = \bigcup_{(u,v) \in E} \{(u, w_{(u,v)}), (v, w_{(u,v)})\}$. Como consecuencia $|E'| = 2|E|$.

$G' = (V', E')$ es el grafo que representa la red para SCA. Las demandas y costos se definen de la siguiente manera:

- $d_e = 1$ para todo $e \in E'$.

- La función de costo $c' : E' \longrightarrow \mathbb{R}$ se define como

$$c'_{(u,w_{(u,v)})} = c'_{(v,w_{(u,v)})} = \frac{c_{(u,v)}}{2}$$

para cada eje $(u, v) \in E$.

Claramente, el grafo G' no tiene cuerdas (Figura 3.1). Esto significa que una solución para SCA debe incluir al menos un ciclo por cada eje de G'. El costo de una solución óptima para SCA será entonces igual al costo de cada solución óptima de MWECCP. □

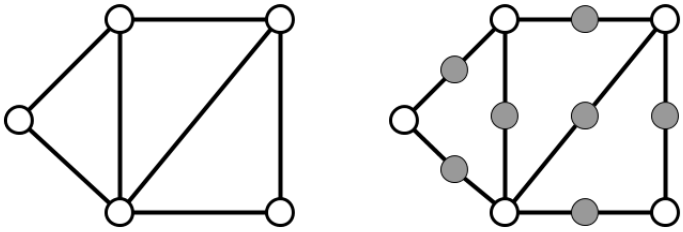


Fig. 3.1: Se eliminan las cuerdas de la instancia de MWECCP para resolver como instancia de SCA.

La asignación de capacidad de reserva es el problema base para diseño de redes con p-ciclos. A continuación presentaremos una formulación compacta, la cual fue introducida anteriormente en (Delgadillo, 2013).

Notación adicional

J : cota superior para el número de ciclos en la solución.

j : índice para los ciclos de la solución ($1 \leq j \leq J$).

$N_G(u)$: conjunto de nodos v tales que $(u, v) \in E$

Es importante notar que G es un grafo no dirigido, por lo tanto un eje incidente en los nodos u y v puede ser notado (u, v) o (v, u) indistintamente. Asumimos que la cota superior J es suficiente para incluir todas las soluciones óptimas del problema. En (Pecorari, 2016) se presenta un valor apropiado para este fin obtenido en función de los coeficientes de entrada, aunque esa cota puede ser muy holgada en la mayoría de los casos.

Esta formulación tiene restricciones de flujo para garantizar un único ciclo por cada índice j al igual que la formulación propuesta por (Gavish & Graves, 1978) para TSP. Por este motivo se incluyen dos vértices adicionales s y t para representar el flujo.

Variables

$x^j_{(u,v)}$: Binaria. $\forall (u, v) \in E$. $x^j_{(u,v)} = 1$ si y solo si el eje (u, v) pertenece al ciclo j.

$y^j_{(u,v)}$: Real (≥ 0). $\forall (u, v) \in E$. Representa la cantidad de unidades de capacidad de protección que el ciclo j otorga al eje (u, v). Según la definición del problema esta

variable debe tomar valores enteros 0, 1 ó 2, pero es posible relajarla como un número real ya que para una solución factible con valor $y^j_{(u,v)} = y^*$ fraccionario, la solución con $y^j_{(u,v)} = \lceil y^* \rceil$ es factible y tiene el mismo valor de función objetivo.

z^j_u : Binaria. $\forall u \in V$. Representa un flujo en la dirección $u \to t$ para el ciclo j. Alternativamente, esta variable puede definirse de forma tal que vale 1 si y solo si el nodo u pertenece al ciclo j.

q^j_{uv} : Real (≥ 0). $\forall u \in V, v \in N_G(u)$ y para $u = s, v \in V$. Representa un flujo en la dirección $u \to v$ para el ciclo j.

r^j_u : Binaria. $\forall u \in V$. $r^j_u = 1$ si y solo si se admite flujo desde s hacia u.

Modelo de Programación Entera Mixta

$$\min \quad \sum_{j=1}^{J} \sum_{(u,v)\in E} c_{(u,v)} x^j_{(u,v)} \tag{3.1}$$

$$\text{s.t.} \quad \sum_{v\in N_G(u)} x^j_{(u,v)} = 2 \times z^j_u \qquad \forall j \leq J, \forall u \in V \tag{3.2}$$

$$\sum_{v\in N_G(u)\cup\{s\}} q^j_{vu} - \sum_{v\in N_G(u)} q^j_{uv} = z^j_u \quad \forall j \leq J, \forall u \in V \tag{3.3}$$

$$q^j_{uv} + q^j_{vu} \leq (|V| - 1) \times x^j_{(u,v)} \qquad \forall j \leq J, \forall (u,v) \in E \tag{3.4}$$

$$\sum_{u\in V} r^j_u \leq 1 \qquad \forall j \leq J \tag{3.5}$$

$$q^j_{su} \leq |V| \times r^j_u \qquad \forall j \leq J, \forall u \in V \tag{3.6}$$

$$y^j_{(u,v)} \leq 2 \times z^j_w \qquad \forall j \leq J, \forall (u,v) \in E, \forall w \in \{u,v\} \tag{3.7}$$

$$y^j_{(u,v)} \leq 2 - x^j_{(u,v)} \qquad \forall j \leq J, \forall (u,v) \in E \tag{3.8}$$

$$\sum_{j=1}^{J} y^j_{(u,v)} \geq d_{(u,v)} \qquad \forall (u,v) \in E \tag{3.9}$$

$$x^j_{(u,v)} \in \{0,1\}, \quad y^j_{(u,v)} \in \mathbb{R}_+ \qquad \forall j \leq J, \forall (u,v) \in E \tag{3.10}$$

$$q^j_{uv} \in \mathbb{R}_+ \qquad \forall j \leq J, (\forall u \in V, v \in N_G(u)) \wedge (u = s, v \in V) \tag{3.11}$$

$$z^j_u \in \{0,1\}, \quad r^j_u \in \{0,1\} \qquad \forall j \leq J, \forall u \in V \tag{3.12}$$

La función objetivo (3.1) representa el costo de la solución. Con (3.2) se definen ciclos, pero estas restricciones solo establecen que cada nodo tiene grado 0 o 2 en la solución, por lo que es necesario agregar restricciones de eliminación de *subtours*. Las restricciones (3.3) aseguran conservación de flujo en cada nodo. Las restricciones (3.4) garantizan que se admite flujo solo en los ejes que pertenecen al ciclo. $(|V| - 1)$ es el mínimo valor que permite ciclos de longitud menor o igual a $|V|$ (necesario para admitir todos los ciclos simples). Las restricciones (3.5) aseguran que, por cada ciclo, solo un nodo puede recibir

flujo desde s, y (3.6) permite la existencia de flujo desde s hacia ese nodo. El coeficiente $|V|$ puede ser reemplazado por un valor mayor. $|V|$ permite que cada ciclo simple sea parte de la solución. Las restricciones (3.7) y (3.8) imponen límites para la protección otorgada por un ciclo. Si uno de los nodos de un eje no es parte del ciclo, la protección para ese eje debe ser 0. Por el contrario, si ambos nodos son parte del ciclo la protección será a lo sumo 2 (restricciones (3.7)). Notar que hay dos restricciones (3.7) por cada ciclo y por cada eje. Si un eje no pertenece al ciclo, su protección será a lo sumo 2. De lo contrario, será a lo sumo 1. Esto es determinado por las desigualdades (3.8). Las restricciones (3.9) aseguran que todos los ciclos de la solución aportan la protección suficiente para cubrir todas las demandas.

3.1. Formulación con restricciones de eliminación de *sub-tour* generalizadas

Presentamos a continuación una nueva formulación para el problema SCA. Está basada principalmente en la formulación propuesta en (Balas, 1989) para el Prize Collecting Traveling Salesman Problem. Para esta formulación solamente es necesario un subconjunto de las variables del modelo anterior.

Variables

$x^j_{(u,v)}$: Binaria. $\forall (u,v) \in E$. $x^j_{(u,v)} = 1$ si y solo si el eje (u,v) pertenece al ciclo j.

z^j_u : Binaria. $\forall u \in V$. $z^j_u = 1$ si y solo si el nodo u pertenece al ciclo j.

$y^j_{(u,v)}$: Real (≥ 0). $\forall (u,v) \in E$. Representa la cantidad de canales de reserva que el ciclo j otorga al eje (u,v).

Modelo de Programación Entera Mixta

$$\min \quad \sum_{j=1}^{J} \sum_{(u,v)\in E} c_{(u,v)} x^j_{(u,v)} \tag{3.13}$$

$$\text{s.t.} \quad \sum_{v\in N_G(u)} x^j_{(u,v)} = 2 \times z^j_u \qquad \forall j \le J, \forall u \in V \tag{3.14}$$

$$\sum_{\substack{(u',v')\in E \\ u'\in S \\ v'\in V\setminus S}} x^j_{(u',v')} \ge 2z^j_u + 2z^j_v - 2 \qquad \begin{array}{l}\forall j \le J, \forall S \subset V, |S| > 1, \\ \forall u \in S, v \in V \setminus S\end{array} \tag{3.15}$$

$$y^j_{(u,v)} \le 2 \times z^j_w \qquad \forall j \le J, \forall (u,v) \in E, \forall w \in \{u,v\} \tag{3.16}$$

$$y^j_{(u,v)} \le 2 - x^j_{(u,v)} \qquad \forall j \le J, \forall (u,v) \in E \tag{3.17}$$

$$\sum_{j=1}^{J} y^j_{(u,v)} \ge d_{(u,v)} \qquad \forall (u,v) \in E \tag{3.18}$$

$$x^j_{(u,v)} \in \{0,1\} \qquad \forall j \le J, \forall (u,v) \in E \tag{3.19}$$

$$y^j_{(u,v)} \in \mathbb{R}_{\ge 0} \qquad \forall j \le J, \forall (u,v) \in E \tag{3.20}$$

$$z^j_u \in \{0,1\} \qquad \forall j \le J, \forall u \in V \tag{3.21}$$

La mayoría de las variables y restricciones son las mismas del modelo previo.

La función objetivo (3.13) representa el costo de la solución. Definimos los ciclos con (3.14), que establecen que cada nodo tiene grado 0 ó 2 en la solución y por lo tanto se deben incluir restricciones de eliminación de *subtour*. Las restricciones de eliminación de *subtour* generalizadas (3.15) aseguran que existe solo un único p-ciclo para cada índice j. Las restricciones restantes se definen del mismo modo que en el modelo anterior.

Al igual que en el modelo previo, las variables y deberían tomar valor entero por su definición. Sin embargo, es suficiente el valor de las variables x para determinar la solución del problema y no es necesario garantizar que $z^j_u = 1 \wedge z^j_v = 1 \Rightarrow y^j_{(u,v)} > 0$ para cada eje (u,v) y ciclo j.

3.2. Descomposición de Dantzig-Wolfe

Ambas formulaciones presentadas en la sección anterior tienen el inconveniente de la simetría: para cada solución factible, es posible obtener otra equivalente realizando una permutación de índices j. Aplicar la descomposición de Dantzig-Wolfe en cualquiera de ellas llevará al mismo Problema Maestro Restringido. Elegimos la segunda formulación para aplicar la descomposición porque el modelo resultante para el problema de *pricing* será el empleado para desarrollar un algoritmo exacto para su resolución.

Se puede identificar una estructura diagonal en bloque en ambos modelos. Notamos A como las "restricciones vinculantes" (3.18) que involucran demandas (comunes a todos los ciclos), y B como las restricciones restantes (3.14), (3.15), (3.16) y (3.17). Llamamos B^j a cada bloque de B con el mismo índice j. Todos las sub-matrices B^j son idénticas y cada una de ellas define un p-ciclo.

$$\begin{pmatrix} B^1 & 0 & \cdots & 0 \\ 0 & B^2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & B^J \\ & & A & \end{pmatrix}$$

Ya que todos los subsistemas determinados por cada B^j son idénticos, omitimos momentáneamente el índice j para definirlos. Sea

$$X^B = \left\{ (x,y,z) : \begin{array}{c} x \in \{0,1\}^{|E|}, y \in {\mathbb{R}_+}^{|E|}, z \in \{0,1\}^{|V|}, \\ (x,y,z) \text{ satisface (3.14), (3.15), (3.16) y (3.17)} \end{array} \right\} \tag{3.22}$$

Para esta formulación, las versiones de convexificación y discretización de la descomposición de Dantzig-Wolfe son equivalentes porque todas las variables enteras son a su vez binarias. En esta sección se aplica la versión de convexificación, es decir que se reemplaza la formulación de X^B por su representación en función de sus puntos extremos. Sea P el conjunto de todos los puntos extremos de la cápsula convexa de X^B.

$$P = \{p = (x^p, y^p, z^p) : p \text{ es un punto extremo de } conv(X^B)\} \tag{3.23}$$

El subsistema X^B puede expresarse como:

$$X^B = \left\{ (x,y,z) = \sum_{p \in P} p\lambda_p : \lambda \geq 0, \sum_{p \in P} \lambda_p = 1, \sum_{p \in P} x^p \lambda_p \in \{0,1\}^{|E|}, \sum_{p \in P} z^p \lambda_p \in \{0,1\}^{|V|} \right\} \tag{3.24}$$

Es posible omitir $\sum_{p \in P} z^p \lambda_p \in \{0,1\}^{|V|}$ porque se cumple como consecuencia de los otros requerimientos.

Si volvemos a introducir los índices j y agregamos variables λ_p^j, las variables de la formulación para cada eje $(u,v) \in E$ son:

$$x^j_{(u,v)} = \sum_{p \in P} x^p_{(u,v)} \lambda^j_p, \qquad \sum_{p \in P} \lambda^j_p = 1 \quad 1 \leq j \leq J \tag{3.25}$$

$$y^j_{(u,v)} = \sum_{p \in P} y^p_{(u,v)} \lambda^j_p, \qquad \sum_{p \in P} \lambda^j_p = 1 \quad 1 \leq j \leq J \tag{3.26}$$

Sea $C_p = \sum_{(u,v) \in E} c_{(u,v)} x^p_{(u,v)}$. El coeficiente C_p representa el costo de cada copia del p-ciclo p. La función objetivo es:

$$\begin{aligned}\sum_{j=1}^{J}\sum_{(u,v)\in E} c_{(u,v)}x^j_{(u,v)} &= \sum_{j=1}^{J}\sum_{(u,v)\in E} c_{(u,v)}\sum_{p\in P} x^p_{(u,v)}\lambda^j_p \\ &= \sum_{p\in P}\sum_{j=1}^{J}\left(\sum_{(u,v)\in E} c_{(u,v)}x^p_{(u,v)}\right)\lambda^j_p \\ &= \sum_{p\in P}\sum_{j=1}^{J} C_p\lambda^j_p \\ &= \sum_{p\in P} C_p\sum_{j=1}^{J}\lambda^j_p\end{aligned}$$

Al reemplazar (3.26) en (3.18) de manera análoga, se obtiene el Problema Maestro resultante de la descomposición Dantzig-Wolfe.

$$\min \quad \sum_{p\in P} C_p\sum_{j=1}^{J}\lambda^j_p \tag{3.27}$$

$$\text{s.t.} \quad \sum_{p\in P} y^p_{(u,v)}\sum_{j=1}^{J}\lambda^j_p \geq d_{(u,v)} \qquad \forall (u,v)\in E \tag{3.28}$$

$$\sum_{p\in P}\lambda^j_p = 1 \qquad 1\leq j\leq J \tag{3.29}$$

$$\lambda \geq 0 \tag{3.30}$$

$$x^j = \sum_{p\in P} x^p\lambda^j_p \qquad 1\leq j\leq J \tag{3.31}$$

$$x^j \in \{0,1\}^{|E|} \qquad 1\leq j\leq J \tag{3.32}$$

La igualdad 3.31 y la omitida $z^j = \sum_{p\in P} z^p\lambda^j_p \quad (1\leq j\leq J)$ serán la base para definir los enfoques de *branching* en el capítulo 4. A partir de este punto el objetivo será simplificar este modelo hasta obtener uno similar al presentado en la sección 1.2.1.

Sea $\{(x,y,z)^j\}_{1\leq j\leq J}$ una solución factible del problema original y sea $\{p_j\}_{1\leq j\leq J}$ un conjunto de J puntos extremos de X^B ($p_j = (x^{p_j}, y^{p_j}, z^{p_j}) \in P,\ \forall j = 1\dots J$), tal que $x^{p_j} = x^j, z^{p_j} = z^j, y^{p_j} \geq y^j$ para cada $j = 1\dots J$. Resulta trivial mostrar que $\{p_j\}_{1\leq j\leq J}$ es también factible, ya que satisface la restricción (3.28). Además, el valor de la función objetivo para $\{p_j\}_{1\leq j\leq J}$ es igual al de $\{(x,y,z)^j\}_{1\leq j\leq J}$ porque $x^j = x^{p_j}\ \forall j = 1\dots J$. Como consecuencia de esto, es admisible considerar únicamente los puntos extremos de X^B. Entonces, establecemos que λ debe ser entera (binaria por 3.29) y las variables x no son más necesarias en el Problema Maestro.

$$\min \quad \sum_{p\in P} C_p \sum_{j=1}^{J} \lambda_p^j \tag{3.27}$$

$$\text{s.t.} \quad \sum_{p\in P} y_{(u,v)}^p \sum_{j=1}^{J} \lambda_p^j \geq d_{(u,v)} \qquad \forall (u,v) \in E \tag{3.28}$$

$$\sum_{p\in P} \lambda_p^j = 1 \qquad 1 \leq j \leq J \tag{3.29}$$

$$\lambda \in \{0,1\}^{J\times|P|} \tag{3.33}$$

Se usará un procedimiento de agregación para remover el índice j y así eliminar la simetría debida a subproblemas idénticos. Definimos variables enteras λ_p del siguiente modo.

$$\lambda_p = \sum_{j=1}^{J} \lambda_p^j \quad \forall p \in P \tag{3.34}$$

Las variables λ_p cuentan el número de copias del p-ciclo p que forman la solución. El Problema Maestro resultante es:

$$\min \sum_{p\in P} C_p \lambda_p \tag{3.35}$$

$$\text{s.t.} \sum_{p\in P} y_{(u,v)}^p \lambda_p \geq d_{(u,v)} \qquad \forall (u,v) \in E \tag{3.36}$$

$$\sum_{p\in P} \lambda_p = J \tag{3.37}$$

$$\lambda_p \in \mathbb{Z}_+ \qquad \forall p \in P \tag{3.38}$$

$$\lambda_p \leq J \qquad \forall p \in P \tag{3.39}$$

La cota J para la cantidad de ciclos en la solución se puede eliminar teniendo en cuenta algunas consideraciones. Sea λ_0 la variable correspondientes al ciclo nulo, que corresponde también a un punto extremo de X^B. Si consideramos que λ_0 es una simple variable de holgura, se puede reemplazar la restricción por igualdad 3.37 por una desigualdad:

$$\sum_{p\in P} \lambda_p \leq J \tag{3.40}$$

Asumiendo que J siempre es suficientemente grande para que la variable de holgura λ_0 sea mayor que cero, la variable dual asociada a esta restricción siempre tendrá valor 0 y no es necesario tenerla en cuenta durante el *pricing*. Luego de eliminar la restricción referida a J, el poliedro resultante no es acotado pero el conjunto de soluciones óptimas no cambia.

El problema puede entonces definirse con (3.35), (3.36) y (3.38), la cual es casi la misma formulación propuesta en la literatura de un modo *ad-hoc* (Grover & Stamatelakis, 1998; Grover et al., 2006) (sección 1.2.1). La única diferencia es que se incluyen en P el p-ciclo nulo ($p = 0$) y otros p-ciclos con protección cero en algunos de sus ejes o cuerdas, ya que también son puntos extremos de X^B. La variable correspondiente al ciclo nulo no será

generada porque su costo reducido es trivialmente 0. Para evitar los ciclos con protección nula en ejes o cuerdas, basta con determinar las columnas generadas dinámicamente a partir de las variables binarias x.

Sean $\pi_{(u,v)}$ las variables duales correspondientes a las restricciones (3.36). El problema de *pricing* se define a continuación.

$$\min \quad \sum_{(u,v)\in E} c_{(u,v)}x_{(u,v)} - \sum_{(u,v)\in E} \pi_{(u,v)}y_{(u,v)} \tag{3.41}$$

$$\text{s.t.} \quad \sum_{v\in N_G(u)} x_{(u,v)} = 2 \times z_u \qquad \forall u \in V \tag{3.42}$$

$$\sum_{\substack{(u',v')\in E \\ u'\in S \\ v'\in V\setminus S}} x_{(u',v')} \geq 2z_u + 2z_v - 2 \qquad \begin{array}{l}\forall S \subset V, |S| > 1, \\ \forall u \in S,\ \forall v \in V \setminus S\end{array} \tag{3.43}$$

$$y_{(u,v)} \leq 2 \times z_w \qquad \forall (u,v) \in E, \forall w \in \{u,v\} \tag{3.44}$$

$$y_{(u,v)} \leq 2 - x_{(u,v)} \qquad \forall (u,v) \in E \tag{3.45}$$

$$x_{(u,v)} \in \{0,1\} \qquad \forall (u,v) \in E \tag{3.46}$$

$$y_{(u,v)} \in \mathbb{R}_+ \qquad \forall (u,v) \in E \tag{3.47}$$

$$z_u \in \{0,1\} \qquad \forall u \in V \tag{3.48}$$

Como resultado de la descomposición, el modelo para el problema de *pricing* corresponde a un ciclo del grafo para el cual existen costos determinados por los mismos coeficientes $c_{(u,v)}$ y "premios" determinados por el valor de las variables duales $\pi_{(u,v)}$ que en este contexto son también coeficientes de entrada. Las desigualdades 3.44 y 3.45 determinan que este premio se obtiene al utilizar el eje asociado y es doble en caso que el eje sea cuerda del ciclo. Además, las restricciones 3.42 determinan que el grado de cada vértice es 0 ó 2 mientras que las restricciones de *subtour* generalizadas 3.43 garantizan un único ciclo.

Proposición 2. *El problema de pricing de SCA es $\mathcal{NP}$-hard.*

Demostración. Se presentará una reducción de tiempo polinomial del Problema del Viajante de Comercio al problema de *pricing* de SCA.

La entrada del problema es un grafo completo $G = (V, E)$ con costos c_e para cada eje $e \in E$. (Asumimos costos positivos). El mismo grafo y costos en los ejes son entrada para el problema de *pricing* de SCA. Los valores π_e deben ser definidos de forma tal que sean lo suficientemente grandes para asegurar que todas las soluciones óptimas son a su vez ciclos Hamiltonianos. Entonces, sea $C = \sum_{e\in E} c_e$ y se define $\pi_e = C$ para cada eje e. Como el grafo es completo, la cantidad de cuerdas de un ciclo (solución) s se calcula a partir de su longitud l_s como $\frac{l_s \times (l_s - 3)}{2}$. Toda solución s tendrá valor objetivo igual a $f(s) = -C \times l_s \times (l_s - 2) + c_s$ con c_s igual al costo de la solución s, calculado con los coeficientes de entrada. Como $C > c_s$ para todo s, para dos soluciones s_1 y s_2, ocurre que $l_{s_1} > l_{s_2} \Rightarrow f(s_1) < f(s_2)$. En particular, la solución óptima será un ciclo Hamiltoniano s^* de costo c_{s^*} mínimo. □

Si bien el problema de *pricing* es $\mathcal{NP}$-hard, muchos resultados de los trabajos sobre el problema de viajante de comercio (TSP) y problemas relacionados, como por ejemplo (Gendreau et al., 1998; Bérubé et al., 2009), pueden ser aplicados para resolverlo de forma eficiente para un tamaño de entrada acotado, acorde a instancias reales del problema SCA provenientes de la industria de las telecomunicaciones.

A lo largo del siguiente capítulo serán introducidas algunas modificaciones a esta formulación derivadas de la introducción de restricciones de *branching* y planos de corte para fortalecer la relajación lineal del Problema Maestro. Dichas modificaciones no alteran la validez de la proposición 2.

Capítulo 4

ALGORITMO BRANCH-AND-PRICE-AND-CUT PARA SCA

En este capítulo se detalla el algoritmo *branch-and-price-and-cut* desarrollado para resolver el problema SCA, basado en el modelo presentado en la sección 3.1. Primero introduciremos la notación utilizada para referirnos a los subconjuntos del conjunto P de puntos extremos (o ciclos, o columnas).

P_e es el conjunto de ciclos que contienen el eje e, de modo que $P_e = \{p \in P : y_e^p = 1\}$. Notar que este conjunto también puede definirse como $P_e = \{p \in P : x_e^p = 1\}$.

P_e^2 es el conjunto de ciclos tales que el eje e es una cuerda, de manera que $P_e^2 = \{p \in P : y_e^p = 2\}$.

P_e^+ es el conjunto de ciclos que protegen el eje e, es decir, $P_e^+ = \{p \in P : y_e^p > 0\}$. Notar que $P_e^+ = P_e \cup P_e^2$.

P_C para $C \subseteq E$ es el conjunto de ciclos que contienen todos los ejes de C, de modo que $P_C = \{p \in P : x_e^p = 1,\ \forall e \in C\}$. Notar que con esta definición, $P_\emptyset = P$.

P_C^2 para $C \subseteq E$ es el conjunto de ciclos que tienen todos los ejes de C como cuerdas, de modo que $P_C^2 = \{p \in P : y_e^p = 2,\ \forall e \in C\}$.

$\mathring{P}_u$ es el conjunto de ciclos que contienen el vértice u, de modo que $\mathring{P}_u = \bigcup_{\substack{(v,w)\in E \\ u\in\{v,w\}}} P_{(v,w)}$.

4.1. Problema Maestro Restringido

El algoritmo *branch-and-price-and-cut* se inicia con un número limitado de columnas. Con la notación presentada reescribimos la formulación del problema maestro del capítulo 3.

$$\min \sum_{p\in P} C_p \lambda_p \tag{3.35}$$

$$\text{s.t.} \sum_{p\in P_e} \lambda_p + \sum_{p\in P_e^2} 2\lambda_p \geq d_e \qquad \forall e \in E \tag{3.36}$$

$$\lambda_p \in \mathbb{Z}_+ \qquad \forall p \in P \tag{3.38}$$

4.2. Problema de Pricing

El problema de *pricing* de SCA y su resolución se tratarán en detalle en el capítulo 5. En cada iteración se puede agregar más de una columna con el objetivo de reducir la

cantidad total de iteraciones. Sin embargo, esto puede eventualmente degradar el rendimiento general del algoritmo debido a la inclusión de columnas irrelevantes en la relajación lineal. Por lo tanto, se incluye un parámetro para elegir el máximo número de columnas agregadas en cada iteración de *pricing*.

Con cada solución del problema de *pricing* se construye una nueva columna a partir del valor de las variables enteras x y z:

$$y_{(u,v)} = \begin{cases} 2 - x_{(u,v)} & \text{si } z_u = z_v = 1 \\ 0 & \text{caso contrario} \end{cases} \qquad \forall (u,v) \in E$$

Esto refiere al hecho que las variables y pueden tomar valor cero en el óptimo de la solución de *pricing* a pesar de que el eje esté protegido por el ciclo, debido a que relajamos esta condición para simplificar la formulación.

En este capítulo serán introducidas más modificaciones relacionadas a restricciones adicionales que luego se resumirán en el capítulo 5.

4.3. *Branching*

4.3.1. Esquemas de *branching*

La aplicación de generación de columnas en un esquema *branch-and-bound* da lugar a un algoritmo *branch-and-price* que para que sea exitoso debe evitar el desbalanceo y la modificación significativa de la estructura del problema de *pricing*.

La estrategia más común en un algoritmo *branch-and-bound* consiste en elegir una variable entera tal que su valor es fraccionario en la solución de la relajación lineal. Luego de elegir una variable λ_p con valor ${\lambda_p}^* \notin \mathbb{Z}$, se generan dos subnodos con las siguientes restricciones

$$\lambda_p \leq \lfloor {\lambda_p}^* \rfloor \tag{4.1}$$

en uno de ellos y

$$\lambda_p \geq \lceil {\lambda_p}^* \rceil \tag{4.2}$$

en el otro. El principal problema de aplicar este procedimiento en esta formulación es que el árbol de búsqueda generado resultaría muy desbalanceado. La restricción 4.2 implica fijar un mínimo número de copias de un ciclo específico. Esto reduce drásticamente el número de soluciones factibles en dicha rama. La restricción 4.1 determina una cota superior en el número de copias a usar de ese ciclo particular en la solución, el cual puede ser uno entre miles de millones. Esto significa que la restricción 4.1 no implica una reducción significativa del espacio de soluciones. Desde otro punto de vista, el valor de la relajación lineal obtenida en este caso no aumenta de manera significativa.

En lugar de seleccionar las variables λ_p directamente, realizamos el *branching* en una variable x de la formulación original. Dado que tenemos una formulación con variables en forma agregada, no es posible seleccionar una única variable x_e^j sino la suma de ellas correspondiente al mismo eje. Por lo tanto, se elige un eje $e \in E$ tal que $\sum_{j=1}^{J} x_e^j = s_e^*$ y $s_e^* \notin \mathbb{Z}$ y se generan subnodos con las restricciones de *branching* $\sum_{j=1}^{J} x_e^j \leq \lfloor s_e^* \rfloor$ y $\sum_{j=1}^{J} x_e^j \geq \lceil s_e^* \rceil$. Teniendo en cuenta la definición respecto a las variables resultantes de la descomposición DW:

$$\sum_{j=1}^{J} x_e^j = \sum_{j=1}^{J} \sum_{p \in P} x_e^p \lambda_p^j = \sum_{p \in P_e} \lambda_p$$

las restricciones agregadas a cada subnodo son:

$$\sum_{p \in P_e} \lambda_p \leq \lfloor s_e^* \rfloor \tag{4.3}$$

$$\sum_{p \in P_e} \lambda_p \geq \lceil s_e^* \rceil \tag{4.4}$$

El significado de esto es la introducción de cotas (superior o inferior) para el número de copias de ciclos (variables λ_p) que usan el eje e:

4.3.2. Cambios en el problema de *pricing*

Cada restricción de *branching* agregada tiene una variable dual correspondiente que debe ser considerada. La fila agregada desde el punto de vista de la formulación original solo tiene coeficientes igual a 1 en las columnas correspondientes a x_e (de cada índice j). De manera que la variable dual nueva afecta solamente a los coeficientes de la función objetivo del problema de *pricing* correspondientes a la variable x_e. Esta es la única modificación a realizar. La estructura del problema de *pricing* no se modifica. Notamos como ν_e a la variable dual correspondiente a la restricción de *branching* del eje e. Entonces, la nueva función objetivo es la siguiente.

$$\min \quad \sum_{e \in E} (c_e - \nu_e) x_e - \sum_{e \in E} \pi_e y_e \tag{4.5}$$

Para las restricciones (4.3) se tiene que $\nu_e \leq 0$ y para (4.4) $\nu_e \geq 0$.

Durante la ejecución del algoritmo, es posible tener en el mismo nodo más de una restricción para el mismo eje, pero a lo sumo a una sola de ellas le corresponderá una variable dual con valor distinto de cero. En caso de que existan ambas (4.3) y (4.4) para la misma constante, se incluye una única restricción por igualdad y su variable asociada ν_e es irrestricta.

Cada vez que una nueva columna es generada, es necesario incluir sus coeficientes distintos de cero en las restricciones de *branching* previas así como en las restricciones de demandas.

4.3.3. Un esquema de *branching* más general

El enfoque presentado previamente no es suficiente para hallar todas las soluciones enteras, ya que podría no existir una variable agregada x_e con valor fraccionario aún teniendo una solución fraccionaria de la relajación lineal. (Ver (Vanderbeck & Wolsey, 2010) para más detalles).

Presentaremos a continuación un esquema de *branching* más general. Por cada par de ejes e y e' se puede definir una variable binaria $t_{ee'} = x_e \times x_{e'}$ en el problema de *pricing*, lo cual da más opciones para establecer restricciones de *branching*. Y para hacerlo de una forma más general, para cada conjunto $C \subset E$ definimos la variable binaria

$$t_C = \prod_{e \in C} x_e \tag{4.6}$$

Para cada solución fraccionaria de la relajación lineal, siempre es posible encontrar un subconjunto $C \subset E$ tal que $\sum_{j=1}^{J} t_C^j = t_C^*$ and $t_C^* \notin \mathbb{Z}$.

Ya que

$$\sum_{j=1}^{J} t_C^j = \sum_{p \in P} t_C^p \lambda_p = \sum_{p \in P_C} \lambda_p$$

podemos usarlo para definir las siguiente restricciones de *branching*

$$\sum_{p \in P_C} \lambda_p \leq \lfloor t_C^* \rfloor \tag{4.7}$$

$$\sum_{p \in P_C} \lambda_p \geq \lceil t_C^* \rceil \tag{4.8}$$

La desventaja de esto es que se debe incluir la nueva variable t_C en el modelo del problema de *pricing* junto con las restricciones necesarias para su definición en cada rama que involucre el conjunto C. La definición 4.6 se introduce con las siguientes restricciones lineales.

$$t_C \leq x_e \ \forall e \in C \tag{4.9}$$

$$t_C \geq (\sum_{e \in C} x_e) - (|C| - 1) \tag{4.10}$$

Sea τ_C la variable dual asociada a las restricciones de *branching* (4.7) ó (4.8). La función objetivo modificada es

$$\min \quad \sum_{e \in E} c_e x_e - \sum_{e \in E} \pi_e y_e - \tau_C t_C \tag{4.11}$$

Ya que $\tau_C \leq 0$ para la restricción de *branching* (4.7), y tenemos un problema de minimización, solo se necesita (4.10). Para la restricción (4.8) con $\tau_C \geq 0$, solo se incluyen las restricciones (4.9). El requerimiento de $t_C \in \{0, 1\}$ se cumple como consecuencia de las otras restricciones, por lo que puede relajarse en ambos casos.

Notar que este enfoque generaliza tanto (4.1)/(4.2) como (4.3)/(4.4). En el primer caso, el conjunto C contiene todos los ejes de un ciclo, y en el segundo caso el conjunto $C = \{e\}$ para algún $e \in E$ y $t_{\{e\}} = x_e$.

Dependiendo del tamaño del conjunto C, encontrarlo puede resultar muy costoso computacionalmente. Por lo tanto, resulta más conveniente usar las restricciones (4.1/4.2) si se supera un tamaño máximo. (Implementamos nuestro algoritmo de forma que solo se consideran tamaños de $|C| < 4$). Si esto ocurre, se puede realizar una modificación alternativa al problema de *pricing* sin la variable t_C en la cual simplemente se elimina el conjunto C del espacio de soluciones factibles con la siguiente restricción.

$$\sum_{e \in C} x_e \leq |C| - 1 \tag{4.12}$$

Esto tiene las desventajas ya mencionadas y debe ser utilizado con la menor prioridad posible.

4.3.4. *Branching* en vértices

Otra alternativa considerada para *branching* es hacerlo de forma agregada en las variables originales z correspondientes a vértices. De esta forma podemos evaluar reglas combinadas donde la prioridad puede estar en variables de vértices o de ejes. Los trabajos consultados que tratan problemas relacionados a TSP asignan mayor prioridad a las variables correspondientes a vértices, como por ejemplo (Bérubé et al., 2009; Gendreau et al., 1998). Ambos presentan algoritmos *branch-and-cut*.

Sea $u \in V$ tal que $\sum_{j=1}^{J} z_u^j = s_u^*$ y s_u^* no es entero. Las restricciones agregadas a cada subnodo son:

$$\sum_{p \in \mathring{P}_u} \lambda_p \leq \lfloor s_u^* \rfloor \tag{4.13}$$

$$\sum_{p \in \mathring{P}_u} \lambda_p \geq \lceil s_u^* \rceil \tag{4.14}$$

El único cambio introducido en el problema de *pricing* es un coeficiente en la función objetivo para la variable z_u (4.15). Previo a introducir este tipo de *branching* asumíamos que estos coeficientes en la función objetivo del problema de *pricing* tenían valor cero. Para cada $u \in V$ notamos como μ_u a la variable dual correspondiente a la restricción de *branching* 4.13 ó 4.14 del vértice u.

$$\min \quad \sum_{e \in E} (c_e - \nu_e) x_e - \sum_{e \in E} \pi_e y_e - \sum_{u \in V} \mu_u z_u \tag{4.15}$$

Al igual como ocurre con las restricciones correspondientes al *branching* en ejes, consideramos (a lo sumo) una única variable μ_u por cada $u \in V$ aunque pueden estar presentes tanto 4.13 como 4.14. Para las restricciones (4.13) $\mu_u \leq 0$ y para (4.14) $\mu_u \geq 0$.

4.3.5. Reglas de *Branching*

Las reglas de *branching* que elegimos para experimentar aplican las restricciones presentadas anteriormente. Para esto evaluamos establecer la prioridad en variables x (ejes) o en variables z (vértices) de la formulación original. Para el caso de variables x evaluamos también tomar en cuenta el coeficiente de costo de los ejes implicados en la elección de la variable para aplicar el *branching*. Respecto a los vértices, consideramos elegir con mayor prioridad los que tengan un valor más "cercano" a la cota inferior que se presenta en la sección 4.4.3, notada como L_u para cada $u \in V$. Tomamos en cuenta además que eventualmente puede ser necesario aplicar una restricción "débil" (4.1 ó 4.2) ya que no existe garantía que se den siempre las condiciones para utilizar los otros enfoques.

Sea λ^* la relajación lineal obtenida en el nodo actual. Sea $s_e^* = \sum_{p \in P_e} {\lambda_p}^*$ para todo $e \in E$. Medimos el puntaje para elección de la variable según la parte fraccionaria de la variable en la relajación lineal:

$$\text{fs}(e, \lambda^*) = 0{,}5 - |s_e^* - \lfloor s_e^* \rfloor - 0{,}5|$$

Observar que si $s_e^* \in \mathbb{Z}$ entonces $\text{fs}(e, \lambda^*) = 0$.

Para cada vértice $u \in V$ definimos de forma análoga $\text{fs}(u, \lambda^*)$ con $s^*_u = \sum_{p \in \mathring{P}_u} {\lambda_p}^*$. Y para $C \subseteq E$ definimos $\text{fs}(C, \lambda^*)$ con $s^*_C = \sum_{p \in P_C} {\lambda_p}^*$.

Formalizamos las reglas de *branching* a evaluar en los experimentos de la siguiente manera, dada una solución λ^* a la relajación lineal no entera.

- Regla 1:

  procedure BRANCH1(λ^*)
  para $SIZE$ de 1 a 3 **hacer**
  Seleccionar el conjunto de ejes $C' = \underset{C \subseteq E,\ |C|=SIZE}{\arg\max} \text{fs}(C, \lambda^*)$.
  Si $\text{fs}(C', \lambda^*) > 0$ **finalizar** realizando el *branching* sobre el conjunto C'.
  fin para
  Si no fue seleccionado un conjunto, aplicar *branching* en alguna variable λ_p con ${\lambda_p}^* \notin \mathbb{Z}$.
  fin procedure

- Regla 2:

  procedure BRANCH2(λ^*)
  para $SIZE$ de 1 a 3 **hacer**
  Seleccionar el conjunto de ejes $C' = \underset{C \subseteq E,\ |C|=SIZE}{\arg\max} \text{fs}(C, \lambda^*) \times \sum_{e \in C} c_e)$
  Si $\text{fs}(C', \lambda^*) > 0$ **finalizar** realizando el *branching* sobre el conjunto C'.
  fin para
  Si no fue seleccionado un conjunto, aplicar *branching* en alguna variable λ_p con ${\lambda_p}^* \notin \mathbb{Z}$.
  fin procedure

- Regla 3 (ó 4):

  Se define $\text{dist}(u, \lambda^*) = \lfloor s^*_u \rfloor - L_u \quad \forall u \in V$
  procedure BRANCH3O4(λ^*)
  $V' \leftarrow \{u \mid u \in V,\ (\forall v \in V)\ \text{dist}(u) \leq \text{dist}(v)\}$
  Se selecciona el vértice $v = \underset{u \in V'}{\arg\max}\, \text{fs}(u, \lambda^*)$
  Si $\text{fs}(u, \lambda^*) > 0$ se aplica *branching* sobre el conjunto C de ejes
  En caso contrario, se aplica regla 1 (ó 2)
  fin procedure

Las reglas 1 y 2 corresponden la realización del *branching* en variables de ejes. La regla 2 pondera el coeficiente de costo mientras que la regla 1 no lo hace. Las reglas 3 y 4 corresponden a la prioridad de *branching* en variables de vértices teniendo como regla secundaria la 1 y 2 respectivamente.

El algoritmo *branch-and-price-and-cut* se configura con una de estas reglas. En el capítulo 6 se presentarán los resultados de los experimentos y la regla elegida.

4.3.6. Cotas para cantidad de ciclos y *early branching*

Para tener la posibilidad de finalizar la generación de columnas de forma prematura y al mismo tiempo contar con alguna cota dual correspondiente al nodo que está siendo

procesado, utilizaremos una cota superior para la cantidad de ciclos que puede tener una solución.

En lugar de reintroducir una cota global J que podría resultar muy holgada (como en la formulación original), se realizará por medio de una última restricción de *branching* que representará la cantidad total de ciclos.

Si $\sum_{p \in P} {\lambda_p}^* = s^*$ y $s^* \notin \mathbb{Z}$, se crean dos subnodos, uno con la restricción

$$\sum_{p \in P} \lambda_p \leq \lfloor s^* \rfloor \tag{4.16}$$

y el otro con la siguiente

$$\sum_{p \in P} \lambda_p \geq \lceil s^* \rceil \tag{4.17}$$

Si σ es la variable dual asociada a estas restricciones, entonces $\sigma \leq 0$ para 4.16 y $\sigma \geq 0$ para 4.17. El cambio en el problema de *pricing* es el agregado de esta variable en la función objetivo, que en ese contexto es simplemente una constante.

La cota superior 4.16 para la cantidad de ciclos permite calcular una cota dual antes de completar la generación de columnas en los nodos del árbol de búsqueda que cuenten con dicha restricción en su rama. Por lo tanto, combinando esto con soluciones primales de calidad se pueden realizar podas o *branching* prematuramente.

Sea λ^* la solución de la relajación lineal del problema maestro restringido en el nodo que está siendo procesado y $\sum_{p \in P' \subset P} C_p \lambda_p^*$ es el valor de función objetivo correspondiente. (P' es el conjunto de columnas en ese nodo). Este valor no es una cota dual válida para el nodo porque el dual correspondiente no es factible. Sea ζ la solución óptima del problema de *pricing* ($\zeta < 0$) y $K \in \mathbb{Z}_{\geq 0}$ el lado derecho de la restricción 4.16 incorporada en el problema maestro. La siguiente es una cota inferior válida para el nodo procesado.

$$\sum_{p \in P' \subset P} C_p \lambda_p^* + K \times \zeta \tag{4.18}$$

Esto permite finalizar la generación de columnas antes de resolver completamente la relajación lineal correspondiente al nodo. Sean $\hat{C}$ la actual cota primal (valor objetivo de la mejor solución entera encontrada) y $\check{C}$ la cota dual global (cota inferior para todos los nodos que restan procesar).

1. Si la cota dada 4.18 es mayor o igual a $\hat{C}$ se puede podar la rama actual como se haría con la solución de la relejación lineal.

2. Si la cota es mayor o igual a $\hat{C} \times \alpha + \check{C} \times (1 - \alpha)$ para algún $\alpha \in \mathbb{R},\ 0 < \alpha < 1$, se finaliza la generación de columnas estableciendo 4.18 como cota inferior del nodo y se procede a realizar el *branching*. Esto no necesariamente lleva a un mejor rendimiento del algoritmo, pero busca minimizar el efecto negativo de la lenta convergencia propia de la generación de columnas.

4.4. Planos de corte

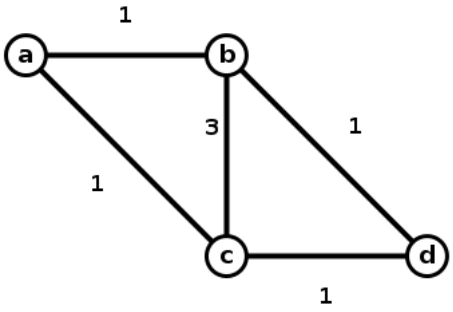
Tal como ocurre con las restricciones agregadas para *branching*, si incorporamos restricciones para fortalecer la relajación lineal debemos tomar en cuenta los valores de las

variables duales asociadas a esas restricciones. Esto implica modificar, en mayor o menor medida, el problema de *pricing*.

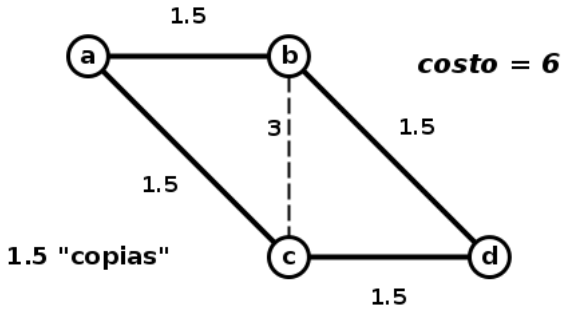
La formulación del Problema Maestro Restringido es muy sencilla y es posible aplicar algún método general de planos de corte [1], pero se pierde la relación con las variables de la formulación original haciendo muy difícil mantener la estructura del problema de *pricing* sin grandes alteraciones. Además, esos planos de corte pueden resultar irrelevantes si no se conoce como agregar las nuevas variables que se generan posteriormente.

En esta sección presentaremos desigualdades válidas que pueden ser separadas fácilmente sin modificar significativamente el problema de *pricing*. Tienen además un significado relacionado con el contexto del problema que resuelve el algoritmo, aprovechando la característica de la protección por cuerdas que aporta 2 canales de reserva.

4.4.1. Cotas inferiores para cantidad de p-ciclos con demandas impares



(a) Grafo y demandas. $c_e = 1\ \forall e \in E$.



(b) Solución a la relajación lineal.

Fig. 4.1: Ejemplo de instancia y solución fraccionaria

Proposición 3. *Para cada eje $e \in E$,*

$$\sum_{p \in P_e^+} \lambda_p \geq \left\lceil \frac{d_e}{2} \right\rceil \tag{4.19}$$

es una desigualdad válida.

[1] Verificamos esto con un solver comercial

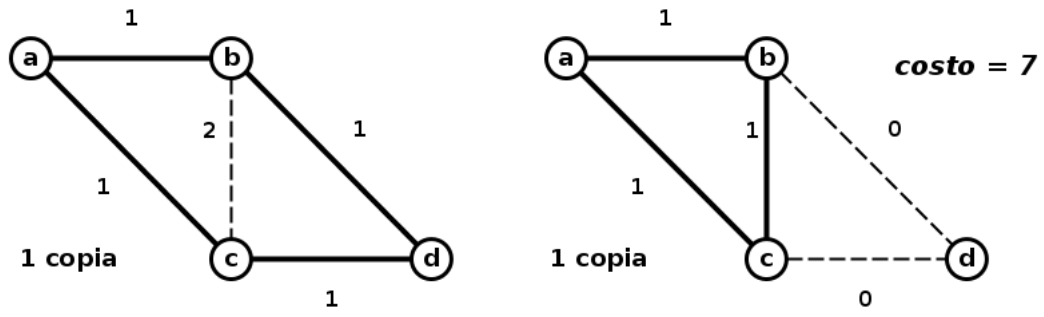


Fig. 4.2: Solución de 6.1a luego de incorporar cota inferior para eje (b, c).

Demostración. Se deriva de las restricciones (3.36). Dividiendo ambos términos por 2:

$$\sum_{p \in P_e} \frac{1}{2} \lambda_p + \sum_{p \in P_e^2} \lambda_p \geq \frac{d_e}{2}$$

Luego

$$\sum_{p \in P_e} \frac{1}{2} \lambda_p + \sum_{p \in P_e^2} \lambda_p \leq \sum_{p \in P_e} \lambda_p + \sum_{p \in P_e^2} \lambda_p = \sum_{p \in P_e^+} \lambda_p$$

Y redondeando $\frac{d_e}{2}$ hacia arriba, porque $\lambda \in \mathbb{Z}_{\geq 0}^{|P|}$.

□

Esta desigualdad es claramente útil cuando la demanda del eje d_e es impar. También tiene un significado en términos del problema real: es una cota inferior para el número de p-ciclos necesarios para lograr proteger el eje e. Es probable que sea violada si d_e es la mayor demanda en un sector de la red. Existen a lo sumo $|E|$ de estas desigualdades, por lo que su separación puede hacerse de forma eficiente.

En la figura 4.1 vemos un ejemplo con la solución a la relajación lineal. En este caso la demanda del eje central (b, c) es 3 y puede ser cubierta con "un ciclo y medio" formado por (a, b, d, c) ya que el eje (b, c) es cuerda del ciclo. Al incorporar la restricción que determina que la cantidad de ciclos que protegen (b, c) debe ser al menos 2 $(= \lceil \frac{3}{2} \rceil)$ se obtiene la solución mostrada en la figura 4.2 que además es entera.

Al igual que las restricciones de *branching*, las nuevas variables deben ser incluidas en los planos de corte generados si tienen ejes o cuerdas incluidas en ellos. Además, incluir cortes hace necesario modificar el problema de *pricing*. Por lo tanto introducimos a la formulación original una variable binaria auxiliar r_e^j definida como sigue:

$$r_e^j = \begin{cases} 1 & \text{si } e \text{ es un eje o cuerda del ciclo } j \\ 0 & \text{caso contrario} \end{cases} \tag{4.20}$$

La desigualdad equivalente en la formulación original es:

$$\sum_{j=1}^{J} \sum_{e \in E} r_e^j \geq \left\lceil \frac{d_e}{2} \right\rceil \tag{4.21}$$

Omitimos el índice j para la formulación del problema de *pricing*. Cada restricción agregada (4.19) para el eje $e \in E$ tiene una variable dual asociada ρ_e. La función objetivo modificada es:

$$\min \quad \sum_{e \in E} c_e x_e - \sum_{e \in E} \pi_e y_e - \sum_{e \in E} \rho_e r_e \tag{4.22}$$

Ya que $\rho_e \geq 0$ y es un problema de minimización, para cada $e = (u, v) \in E$ solamente se necesita introducir las siguientes restricciones.

$$r_e \leq z_u \qquad r_e \leq z_v \tag{4.23}$$

También se puede relajar la restricción $r_e \in \{0, 1\}$ reemplazándola por $r_e \in \mathbb{R}$.

Es posible eliminar las variables y_e al introducir las variables r_e, ya que $\forall e \in E$, $y_e = 2 \times r_e - x_e$. En el capítulo 5 se tratará este tema, ya que se usará r_e en lugar de y_e en la implementación para simplificar la modificación relacionada a estas desigualdades.

4.4.2. Fortaleciendo las restricciones de *branching*

Proposición 4. *Para una restricción de branching* $\sum_{p \in P_e} \lambda_p \geq a$ *y* $d_e > a$*:*

$$\sum_{p \in P_e^+} \lambda_p \geq \left\lceil \frac{d_e - a}{2} \right\rceil + a \tag{4.24}$$

es una desigualdad válida.

Demostración. Se comienza con la suma de las restricciones de *branching* y del modelo (3.36):

$$\begin{aligned}
2 \sum_{p \in P_e} \lambda_p + \sum_{p \in P_e^2} 2\lambda_p &\geq d_e + a = d_e - a + 2a \\
\sum_{p \in P_e} \lambda_p + \sum_{p \in P_e^2} \lambda_p &\geq \frac{d_e - a}{2} + a && \text{dividiendo por 2} \\
\sum_{p \in P_e \cup P_e^2} \lambda_p &\geq \left\lceil \frac{d_e - a}{2} \right\rceil + a && \text{porque } \lambda_p \in \mathbb{Z}_{\geq 0}^{|P|}
\end{aligned}$$

□

La restricción de *branching* 4.4 establece que un número mínimo de ciclos que contienen el eje e estarán en la solución, lo cual implica que se puede derivar una cota inferior para el número total de ciclos que protegen el eje e como eje o cuerda del ciclo (conjunto P_e^+). Estas desigualdades son útiles cuando $d_e - a$ es impar. Si d_e es impar y existe un corte (4.19) agregado previamente esta desigualdad puede reemplazarlo.

Para el problema de *pricing* se deben tener las mismas consideraciones que para los planos de corte 4.19. Por lo tanto se usará la misma variable r_e y la variable dual asociada ρ_e definidas en la sección 4.4.1.

Proposición 5. *Para una restricción de branching* $\sum_{p \in P_e} \lambda_p \leq b$ *y* $d_e > b$*:*

$$\sum_{p \in P_e^2} \lambda_p \geq \left\lceil \frac{d_e - b}{2} \right\rceil \tag{4.25}$$

es una desigualdad válida.

Demostración. Sumamos la desigualdad $-\sum_{p \in P_e} \lambda_p \geq -b$ y la correspondiente a la demanda de e (3.36) para obtener:

$$\begin{aligned}
&\sum_{p \in P_e^2} 2\lambda_p \geq d_e - b \\
&\sum_{p \in P_e^2} \lambda_p \geq \frac{d_e - b}{2} && \text{dividiendo por 2} \\
&\sum_{p \in P_e^2} \lambda_p \geq \left\lceil \frac{d_e - b}{2} \right\rceil && \text{por } \lambda_p \text{ entera}
\end{aligned}$$

□

Estas desigualdades también pueden separarse eficientemente y son útiles cuando $d_e - b$ es impar. Además tienen el siguiente significado: ya que la restricción de *branching* 4.3 establece una cota superior para la cantidad de veces que e se usa como eje de algún ciclo, es posible derivar una cota inferior para el número de p-ciclos necesarios con el eje e como cuerda.

Se necesitan modificaciones similares a las anteriores para el problema de *pricing*. Introducimos a la formulación original una variable binaria auxiliar $r_e'^j$:

$$r_e'^j = \begin{cases} 1 & \text{si el eje } e \text{ es una cuerda del ciclo } j \\ 0 & \text{en otro caso} \end{cases} \tag{4.26}$$

Sea ρ_e' la variable dual asociada a la restricción (4.25). La función objetivo modificada es:

$$\min \quad \sum_{e \in E} c_e x_e - \sum_{e \in E} \pi_e y_e - \sum_{e \in E} \rho_e' r_e' \tag{4.27}$$

Al igual que antes, $\rho_e' \geq 0$, y para $e = (u, v)$, necesitaremos introducir las siguientes restricciones:

$$r_e' \leq z_u \qquad r_e' \leq z_v \tag{4.28}$$

$$r_e' \leq 1 - x_e \tag{4.29}$$

Nuevamente, el requerimiento $r_e' \in \{0, 1\}$ se puede relajar porque el valor de la variable será entero como consecuencia del resto del modelo.

4.4.3. Cota inferior para uso de vértices

Una cota inferior L_u para un vértice determinado $u \in V$ tiene como significado que son necesarios al menos L_u ciclos en la solución que utilicen el vértice u.

$$\sum_{p \in \mathring{P}_u} \lambda_p \geq L_u \tag{4.30}$$

A partir de las demandas de los ejes que inciden sobre el vértice u es posible calcular cotas inferiores L_u. Una cota trivial sería la demanda máxima de los ejes incidentes dividido 2, lo que significa que se necesita al menos esa cantidad de ciclos para satisfacer las demandas suponiendo que es posible usar ciclos que tienen el eje con demanda máxima como cuerda sin tener en cuenta el resto de los ejes.

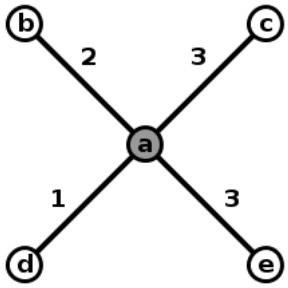
Utilizaremos una cota más ajustada, para la cual se asume que para cada par de ejes (e_1, e_2) incidentes a u existe un ciclo en el grafo que contiene a e_1, e_2 y a todos los vértices adyacentes a u. Con estas condiciones es posible elegir exclusivamente ciclos que protegen a todos los ejes incidentes a u, conjunto al cual notamos como E_u. Cada ciclo otorga 1 unidad de protección a dos de los ejes de E_u y 2 unidades al resto. Ignorando los costos y tomando en cuenta solamente las demandas de los ejes incidentes en u, se construye una solución local al vértice u de cardinalidad mínima de forma iterativa eligiendo en cada paso uno de esos ciclos tal que el par de ejes usados tiene el mínimo de demanda restante por cubrir. El procedimiento se muestra en el algoritmo 1.

Algoritmo 1 Cota inferior para uso de vértice

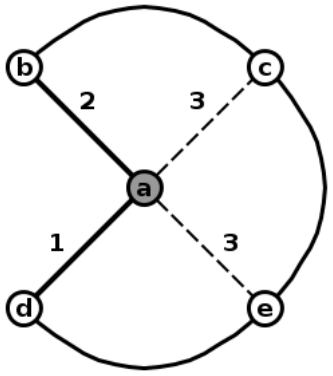
```
1: función cotaInferior(u, E, demandas[])
2:     L_u ← 0
3:     E_u ← ejes incidentes en u.
4:     restantes ← demandas
5:     mientras ∃e ∈ E_u : restantes[e] > 0 hacer
6:         L_u ← L_u + 1
7:         Seleccionar e_1 y e_2 tal que e_1 ≠ e_2 y ∀e ∈ E_u \ {e_1, e_2} restantes[e] ≥
   restantes[e_1] ∧ restantes[e] ≥ restantes[e_2]
8:         restantes[e_1] ← restantes[e_1] - 1
9:         restantes[e_2] ← restantes[e_2] - 1
10:        para todo e ∈ E_u \ {e_1, e_2} hacer
11:            restantes[e] ← restantes[e] - 2
12:        fin para
13:    fin mientras
14:    devolver L_u
15: fin función
```

Se presenta un ejemplo en la figura 4.3 para el vértice a. Solamente se consideran los 4 ejes que inciden sobre el vértice, de los cuales la demanda máxima es 3 (figura 4.3a). Como se asume la existencia de caminos disjuntos de b hacia c, de c hacia e, y de e hacia d, la cantidad mínima de ciclos necesarios es 2 (figura 4.3b), que es la cota inferior obtenida por el algoritmo 1.

Esta cota inferior se calcula previamente a la ejecución del algoritmo para cada vértice del grafo de entrada y por lo tanto se puede verificar de forma eficiente en cualquier nodo



(a) Vértice con demandas en sus ejes incidentes.



(b) Asumiendo que existen los ejes necesarios, dos copias de este ciclo cubren las demandas.

Fig. 4.3: Ejemplo de cota inferior para uso de vértices.

del árbol de búsqueda. En caso de encontrar una desigualdad 4.30 violada, se incorpora como plano de corte. La modificación en el problema de *pricing* es un coeficiente en la función objetivo para la variable z_u correspondiente, del mismo modo que se explicó en la sección 4.3.4 ya que es equivalente a la restricción de branching 4.14. Notar que aplicar esta restricción prematuramente como corte evitaría verificar posteriormente si un subnodo surgido a partir de una restricción de *branching* 4.13 es factible o no, en el caso que su relajación lineal resulte inicialmente no factible (es decir, antes de la primera iteración de *pricing*).

4.4.4. Planos de corte generales asociados a ejes

Proposición 6. *Sea $T \subseteq E$ ($|T| > 1$) tal que $\nexists C \subseteq E$ tal que C es ciclo y $\forall e \in T$ e es cuerda de C.*

Además sea $\check{P}_T$ el conjunto de ciclos que protegen al menos a uno de los ejes de T, es decir

$$\check{P}_T = \bigcup_{e \in T} P_e^+$$

Entonces la siguiente es una desigualdad válida.

$$\sum_{p \in \check{P}_T} \lambda_p \geq \left\lceil \frac{\sum_{e \in T} d_e}{2|T| - 1} \right\rceil \tag{4.31}$$

Demostración. A partir de la suma de todas las restricciones de demanda 3.36 de cada eje

de T se tiene

$$\sum_{e\in T} d_e \leq \sum_{e\in T}\left(\sum_{p\in P_e}\lambda_p + 2\times\sum_{p\in P_e^2}\lambda_p\right) \tag{4.32}$$

$$= \sum_{\substack{A\in\mathcal{P}(T)\\ B\in\mathcal{P}(T\setminus A)}} (|A|+2|B|)\sum_{p\in P_A\cap P_B^2}\lambda_p \qquad \text{en partes} \tag{4.33}$$

$$\leq \sum_{\substack{A\in\mathcal{P}(T)\\ B\in\mathcal{P}(T\setminus A)}} (2|T|-1)\sum_{p\in P_A\cap P_B^2}\lambda_p \qquad \begin{array}{r}|A|+2|B| \text{ no alcanza}\\ \text{su valor máximo}\end{array} \tag{4.34}$$

$$= (2|T|-1)\sum_{p\in\check{P}_T}\lambda_p \tag{4.35}$$

$$\frac{\sum_{e\in T} d_e}{2|T|-1} \leq \sum_{p\in\check{P}_T}\lambda_p \tag{4.36}$$

$$\left\lceil\frac{\sum_{e\in T} d_e}{2|T|-1}\right\rceil \leq \sum_{p\in\check{P}_T}\lambda_p \qquad \text{por } \lambda \text{ entera} \tag{4.37}$$

El valor máximo para $|A|+2|B|$ se da cuando $A=\emptyset$ y $B=T$, pero el conjunto correspondiente es vacío ya que $P_\emptyset = P$ y $P_T^2 = \emptyset$ por hipótesis. □

Para que la aplicación de estas desigualdades resulte útil es necesario que $d_e > 0$ para cada $e\in T$, además de que $\dfrac{\sum_{e\in T} d_e}{2|T|-1}\notin\mathbb{Z}$. El costo computacional necesario para encontrar una desigualdad violada de este tipo crece con el tamaño de T. En este trabajo consideramos solo pares de ejes para su separación, asumiendo inicialmente válida la hipótesis de la ausencia del ciclo y realizando la verificación luego de encontrar un par de ejes para el cual la desigualdad es violada.

La figura 4.4 muestra un ejemplo con $|T|=2$. La instancia tiene solo dos ejes con demanda mayor a cero, (a,c) y (b,d), y no existe un ciclo que tenga esos dos ejes como cuerdas. Por lo tanto, el coeficiente máximo al sumar las dos restricciones correspondientes es 3. Al incorporar la desigualdad 4.31 para $T=\{(a,c),(b,d)\}$ se obtiene la solución de la figura 4.5.

En cuanto al problema de *pricing*, es necesario agregar una variable binaria por cada conjunto T considerado.

$$q_T^j = \begin{cases} 1 & \text{si algún eje } e\in T \text{ es protegido por el ciclo } j\\ 0 & \text{en otro caso}\end{cases} \tag{4.38}$$

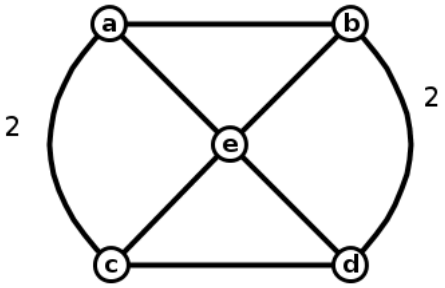
Sea ω_T la variable dual asociada a la restricción 4.31. La función objetivo del problema de *pricing* modificada es

$$\min \quad \sum_{e\in E} c_e x_e - \sum_{e\in E}\pi_e y_e - \omega_T q_T \tag{4.39}$$

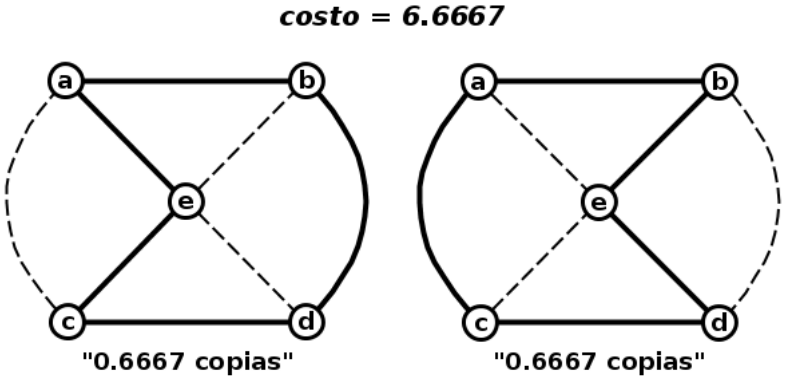
Debido a que $\omega_T \geq 0$ la definición en la formulación lineal es

$$q_T \leq \sum_{e\in T} r_e \tag{4.40}$$

$$q_T \leq 1 \tag{4.41}$$



(a) Grafo y demandas, las faltantes son iguales a 0. $c_e = 1\ \forall e \in E$.



(b) Solución a la relajación lineal.

Fig. 4.4: Instancia y solución fraccionaria

Y se define $q_T \in \mathbb{R}_{\geq 0}$ ya que su coeficiente es negativo en la función objetivo y el problema es de minimización.

4.5. Nodos no factibles

Se mostrará en la sección 4.8 que se garantiza la inclusión de una solución factible en las columnas iniciales de modo que el Problema Maestro Restringido inicial será siempre factible. Sin embargo, es posible que luego de la inserción de una restricción de *branching* el problema lineal resultante sea infactible, en particular con 4.13 ó 4.16. En este caso no se puede descartar el nodo inmediatamente ya que puede ser factible agregando una o más columnas que no están presentes en la formulación restringida.

La propia generación de columnas provee un método para encontrar columnas que hagan factible el problema o bien que prueben que es efectivamente no factible, usando el hecho de que el dual de un problema infactible es no acotado, formalizado en el Lema de Farkas (ver en (Nemhauser & Wolsey, 1988)).

Teorema 3 (Lema de Farkas)**.** *Para* $A \in \mathbb{R}^{n \times m}$, $b \in \mathbb{R}^m$.

$$\{x \in \mathbb{R}_{\geq 0} : Ax \leq b\} \neq \emptyset \Leftrightarrow \nexists v \in \mathbb{R}^m_{\geq 0} \text{ tal que } vA \leq 0 \text{ y } vb > 0 \tag{4.42}$$

El vector v es un certificado de infactibilidad y se deriva según el algoritmo de resolución de Programación Lineal (Lübbecke, 2011; Andersen, 2001), además que se interpreta como

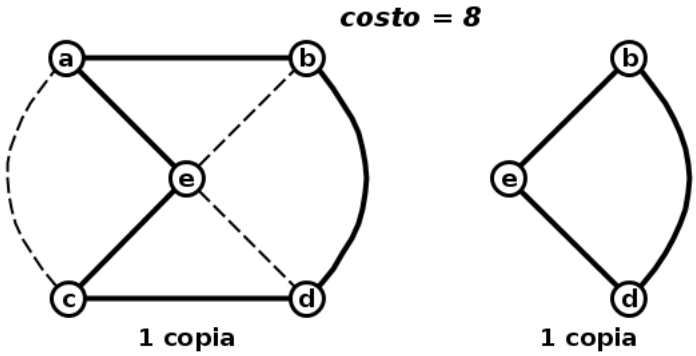


Fig. 4.5: Solución de 4.4a luego de incorporar cota inferior para par de ejes.

un rayo en el dual. Prueba que no es factible ya que no es posible satisfacer $vAx \leq vb$. No es único ya que se puede multiplicar por un escalar distinto de 0 manteniendo esta propiedad. El objetivo entonces es encontrar una columna a para agregar a A tal que $va > 0$, con lo cual el dual es acotado en la dirección de v. Esto significa que es el mismo problema de *pricing* en el cual los coeficientes de la función objetivo del primal se reemplazan por 0 y el vector v reemplaza al vector de variables duales.

$$\min_{x \in X} \{-v\ a(x)\} \tag{4.43}$$

En este documento utilizaremos para este método el nombre de *pricing* Farkas introducido en (Achterberg, 2009) en el contexto del *solver* utilizado para la implementación (capítulo 6).

4.6. Heurísticas primales

SCA es un problema relativamente fácil para encontrar soluciones factibles, sin embargo estas soluciones podrían ser de una calidad muy pobre. Encontrar soluciones primales teniendo como base el Problema Maestro Restringido es una tarea muy relacionada con las heurísticas propuestas por los autores que propusieron el problema SCA en las cuales construyen soluciones teniendo como base un conjunto de ciclos candidatos. En esta sección presentamos dos enfoques para esto, uno exacto (con tiempo acotado) y otro heurístico, incorporados como heurísticas primales dentro de nuestro algoritmo.

4.6.1. Problema Maestro Restringido como sub-MIP

Una heurística primal muy conocida para resolver MIPs como parte de un algoritmo *branch-and-cut* es RINS (*Relaxation Induced Neighborhood Search* (Danna et al., 2005)). RINS depende de una solución factible encontrada previamente y su objetivo es mejorarla. En resumen, construye un vecindario prometedor usando información contenida en la relajación lineal correspondiente al nodo que está siendo procesado. La exploración del vecindario se formula como un modelo MIP y se resuelve de forma recursiva. RINS está implementada y usada con éxito por *solvers* comerciales. Por supuesto tiene un costo de

ejecución relativamente alto, y por lo tanto el proceso de resolución tiene límite de tiempo y de nodos procesados. Además se ejecuta solamente en algunos pocos nodos seleccionados. Una configuración usual es, por ejemplo, a una profundida múltiplo de 20 en el árbol de búsqueda.

Sea $\bar{\lambda}$ la mejor solución entera encontrada hasta el momento. El procedimiento es el siguiente:

1. Fijar las variables que tienen el mismo valor en $\bar{\lambda}$ y en la relajación lineal.
2. Fijar el valor objetivo de $\bar{\lambda}$ como cota primal para el sub-MIP.
3. Resolver el sub-MIP con las variables restantes.

Los planos de corte encontrados previamente se incorporan para ser aprovechados por el sub-MIP. Por el contrario, las restricciones de branching se ignoran independientemente del nodo donde se ejecuta. Esto significa que no se restringe la solución encontrada a ser factible en un nodo particular del árbol.

RINS no puede aplicarse directamente en nuestro Problema Maestro Restringido porque tiene variables agregadas, lo que quiere decir que cada variable entera representa una cantidad exacta de copias de un ciclo en particular. En el caso de que todas las variables de la relajación lineal tengan valor cero o fraccionario (algo que es frecuente), ninguna de las variables distintas a cero se fijarán. Por otro lado, la mayoría de las variables serán excluidas por tener valor cero, que posiblemente coincide con su valor en la solución $\bar{\lambda}$. Esto aplica también para las variables que no están presentes en el Problema Maestro Restringido.

Sea λ^* la relajación lineal. Para aprovechar la información común entre $\bar{\lambda}$ y λ^*, establecemos el valor mínimo de las variables en

$$\lambda_p \geq \text{mín}\left(\lfloor {\lambda_p}^* \rfloor, \bar{\lambda}_p\right) \quad \forall p \in P$$

Esto se interpreta como que si tanto la mejor solución entera como la relajación lineal contienen un cierto número de copias de un ciclo p en particular, esa cantidad se fija como mínimo en el sub-MIP.[2] De este modo, esas copias de p están forzadas a estar en la solución de la heurística.

Debido a que las variables consideradas por el sub-MIP serían las que tienen valor distinto a cero en la mejor solución entera o la relajación lineal, la cantidad de variables a considerar es muy acotada. Esto es una limitación que tendría en cuenta pocos ciclos distintos para construir una solución. Para mitigar este inconveniente realizamos otra modificación al esquema de RINS que consiste en incluir un número limitado de variables adicionales del Problema Maestro Restringido con menos de 10 iteraciones fuera de la base de LP.

Aún manteniendo un número limitado de nodos a procesar y tiempo de ejecución general, se pueden obtener soluciones de muy buena calidad como se verá en el capítulo 6.

4.6.2. Heurísticas golosas

Implementamos dos heurísticas primales golosas para obtener soluciones factibles con bajo costo computacional. El objetivo principal es proveer a la heurística de la sección

[2] En la implementación realizamos esto actualizando las demandas según corresponde, sin modificar el dominio de las variables.

anterior con soluciones primales de calidad y eventualmente mejorarlas. Inicializamos ambas del mismo modo y utilizamos el mismo criterio para seleccionar los ciclos de forma iterativa. La diferencia está en el conjunto de ciclos candidatos, es decir, que toman en cuenta subconjuntos distintos de las variables del Problema Maestro Restringido.

El procedimiento común es el siguiente:

1. Dada una solución λ^* de la relajación lineal, establecemos como valor inicial $\lfloor {\lambda_p}^* \rfloor$ para cada variable λ_p del Problema Maestro Restringido.
2. Actualizamos las demandas correspondientemente.
3. Mientras las demandas no estén satisfechas, seleccionamos el ciclo que maximice la eficiencia *a priori* (AE_p) respecto a las demandas no cubiertas, lo incorporamos (incrementando la variable correspondiente) y actualizamos las demandas.

Este procedimiento se detalla en el algoritmo 2, con un procedimiento auxiliar en el algoritmo 3.

La eficiencia *a priori* es el criterio propuesto en (Grover & Doucette, 2002) que se define como:

$$AE_p = \frac{\sum_{e \in E} y_e^p d'_{e_p}}{C_p} \qquad \forall p \in P \tag{1.5}$$

donde d'_e es la demanda restante por cubrir del eje $e \in E$.

Evaluamos otros criterios en pruebas preliminares, como por ejemplo tomar el máximo entre y_e^p y d'_e en lugar del producto, pero obtuvimos mejores resultados con AE_p.

Como conjunto de ciclos candidatos tenemos dos alternativas:

1. Todas las columnas disponibles en el Problema Maestro Restringido.
2. Las variables con valor fraccionario en la solución a la relajación lineal del nodo procesado.

Cada una de ellas determina una heurística distinta, que aplicamos en cada nodo al finalizar el proceso de *pricing*. La segunda alternativa tiene un sesgo mayor hacia la solución de la relajación lineal que la primera.

4.7. Selección de nodos

Para la selección del siguiente nodo a procesar se tiene en cuenta la cota inferior obtenida previamente a la creación y una prioridad derivada de la variable utilizada para *branching*. En primer lugar, la selección se realiza con prioridad decreciente según el siguiente orden.

- Creado por *branching* en cantidad total de ciclos. (4.16-4.17).
- Creado por *branching* en vértice. (4.13-4.14).
- Creado por *branching* en conjunto de ejes $C \subseteq E$. (4.7-4.8). Dentro de estas tienen mayor prioridad las de menor tamaño $|C|$.
- *Branching* en variable de problema maestro. (4.3-4.4).

Entre los disponibles con mayor prioridad, se selecciona el nodo que tiene la mayor cota inferior resultante de la resolución de la relajación lineal en su nodo inmediatamente superior.

Algoritmo 2 Heurística primal con variables del PMR como ciclos candidatos

Require: λ^*: solución de relajación lineal, P_{cand}: ciclos candidatos

función HEURISTICAPRIMAL(λ^*, V, E, demandas[], P_{cand})
  para todo $p \in P_{cand}$ **hacer**
    $\lambda_p \leftarrow \lfloor {\lambda_p}^* \rfloor$
  fin para
  restantes $\leftarrow$ demandas
  para todo $p \in P_{cand}$ **hacer**
    ACTUALIZARDEMANDAS(E, restantes, p, λ_p)
  fin para
  mientras $\exists e \in E$: restantes[e] > 0 **hacer**
    $p \leftarrow \arg\max_{q \in P_{cand}} AE_q$
    $\lambda_p \leftarrow \lambda_p + 1$
    ACTUALIZARDEMANDAS(E, restantes, p, 1)
  fin mientras
  devolver λ
fin función

Algoritmo 3 Procedimiento auxiliar para actualizar demandas restantes dado un ciclo y una cantidad de copias

Require: c: ciclo, n: número de copias

procedure ACTUALIZARDEMANDAS(E, restantes[], c, n)
  para todo $e \in E : e$ *es eje de* c **hacer**
    restantes[e] $\leftarrow$ restantes[e] - n
  fin para
  para todo $e \in E : e$ *es cuerda de* c **hacer**
    restantes[e] $\leftarrow$ restantes[e] - $2 \times n$
  fin para
fin procedure

4.8. Columnas iniciales

Para que las columnas inciales del Problema Maestro Restringido garanticen que la relajación lineal sea factible alcanza con incluir las columnas correspondientes a alguna solución factible de SCA. El mismo algoritmo para el problema de *pricing* cuyo resultado es un ciclo, puede ser usado para generar de forma iterativa una solución para SCA, independientemente de si es exacto o heurístico.

Algoritmo 4 Columnas iniciales

```
1: función INICIAL(V, E, demandas[], costos[])
2:     S ← ∅
3:     restantes ← demandas
4:     mientras ∃e ∈ E : restantes[e] > 0 hacer
5:         C ← GENERARCOLUMNA(V, E, restantes, 0̄)
6:         S ← S ∪ {C}
7:         ACTUALIZARDEMANDAS(E, restantes, C, 1)
8:     fin mientras
9:     devolver S
10: fin función
```

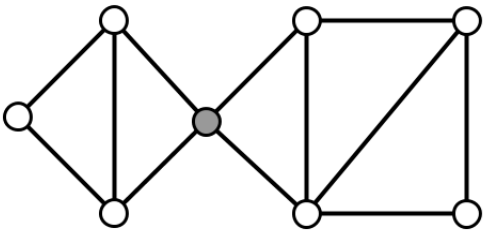
El algoritmo 4 detalla el procedimiento. La entrada para el problema de *pricing*, además del grafo que representa la red, incluye dos valores no negativos para cada eje: el costo y el valor de la variables duales π en el problema maestro. En cada iteración se utilizan las demandas no cubiertas como vector de π y se establece el costo nulo. El ciclo resultante se agrega a la solución y se usa para actualizar las demandas no cubiertas. Los motivos para el costo nulo son dos. En primer lugar, al ignorar los costos se maximiza la demanda cubierta por iteración y de esa forma se espera realizar menos iteraciones, y en segundo lugar la resolución sin tener en cuenta costo es, por lo general, más simple.

Como alternativa para realizar menos iteraciones, se puede asumir que al agregar cada ciclo a la solución, se incorpora la máxima cantidad de copias del mismo tal que la siguiente copia agregada no cubra más demandas. Esto puede hacerse simplemente modificando la actualización. En el caso particular de que el grafo sea Hamiltoniano, solo se ejecutaría una iteración y solamente habría una columna al inicio del algoritmo de generación de columnas.

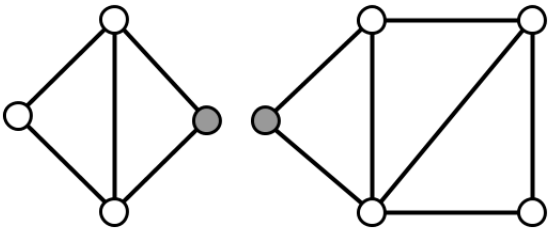
4.9. Preprocesamiento

Si el grafo de entrada es 2-eje-conexo (si se elimina un eje, el nuevo grafo es conexo) pero tiene uno o más puntos de articulación, se puede simplificar la instancia de entrada de SCA separando el grafo en las componentes conexas resultantes al eliminar los puntos de articulación. Estos vértices se incorporan nuevamente a cada componente conexa donde existan vértices adyacentes en el grafo original (figura 4.6). La solución para la instancia original es la unión de las soluciones individuales da cada instancia descompuesta, cada una de ellas un multiconjunto de ciclos.

Si un vértice tiene grado 2, entonces tiene dos ejes e_1 y e_2 incidentes que no son cuerdas de ningún ciclo. Las filas de la matriz correspondientes a e_1 y e_2 son iguales, y solamente es necesaria una de ellas tomando como lado derecho $\max(d_{e_1}, d_{e_2})$.



(a) Grafo de entrada con punto de articulación.



(b) Separado en instancias más simples.

Fig. 4.6: Instancia divida en 2 de menor tamaño

Generalizando el caso anterior, si el grafo de entrada tiene un camino con vértices intermedios de grado 2, este camino puede ser reemplazado por un camino de longitud 2. Sean $\{u, v, w_1, \ldots, w_k\} \subseteq V$ tal que $k > 1$, el grado de w_i es 2 para cada $i = 1, \ldots, k$ y $(u, w_1, \ldots, w_k, v)$ es un camino del grafo. El nuevo conjunto de vértices es $V \setminus \{w_2, \ldots, w_k\}$, colapsando los ejes eliminados en uno solo (w_1, v), para el cual se define demanda y costo.

$$d_{(w_1,v)} = \max\left(\{d_{(w_k,v)}\} \cup \{d_{(w_{i-1},w_i)} : i = 2, \ldots, k\}\right) \tag{4.44}$$

$$c_{(w_1,v)} = c_{(w_k,v)} + \sum_{i=2}^{k} c_{(w_{i-1},w_i)} \tag{4.45}$$

4.10. Resumen del algoritmo

El pseudocódigo 5 es el esquema general del algoritmo *branch-and-price-and-cut*.

El algoritmo de *pricing* se detalla en el siguiente capítulo. La cota inferior lb_i está disponible ($lb_i > -\infty$) en caso que no existan columnas con costo reducido negativo, o si se cuenta con la restricción 4.16. El parámetro α en caso de ser 1 deja inactiva esta característica.

Respecto a la separación de planos de corte, el orden a utilizar en el nodo raíz es el siguiente:

1. Demandas impares en eje 4.19.
2. Cota inferior para uso de vértice 4.30.
3. Conjunto de ejes 4.31.

Y en los demás nodos del árbol de búsqueda:

Algoritmo 5 Esquema general del algoritmo *branch-and-price-and-cut*

```
1: función SCA(V, E, demandas[], costos[])
2:     PREPROCESAMIENTO
3:     COLUMNASINICIALES
4:     LB ← −∞
5:     UB ← ∞
6:     mientras UB > LB y límite de tiempo y nodos no alcanzados hacer
7:         i ← SELECCIONAR NODO
8:         mientras i no finalizado hacer
9:             RESOLVERLP
10:            si i no factible entonces
11:                C ← PRICINGFARKAS
12:                si C ≠ ∅ entonces
13:                    agregar C a LP_i
14:                sino
15:                    marcar i finalizado (infactible)
16:                fin si
17:            sino
18:                (C, lb_i) ← PRICING
19:                si lb_i ≥ UB entonces
20:                    marcar i finalizado (poda)
21:                sino si C ≠ ∅ entonces
22:                    agregar C a LP_i
23:                    si lb_i ≥ UB × α + LB × (1 − α) entonces
24:                        BRANCHING
25:                        marcar i finalizado (early branch)
26:                    fin si
27:                sino si LP_i entera entonces
28:                    actualizar UB
29:                    marcar i finalizado (entera)
30:                sino
31:                    R ← SEPARACIÓN
32:                    si R ≠ ∅ entonces
33:                        agregar R a LP_i
34:                    sino
35:                        HEURISTICASPRIMALES
36:                        actualizar UB
37:                        BRANCHING
38:                        marcar i finalizado (branch)
39:                    fin si
40:                fin si
41:            fin si
42:        fin mientras
43:        actualizar LB
44:    fin mientras
45:    devolver (LB, UB)
46: fin función
```

- Restricción de *branching* en eje 4.25 y 4.24.
- Cota inferior para uso de vértice 4.30.

En cuanto a las heurísticas primales, se limita el uso de la heurística 4.6.1 solo a algunos nodos además de que se aplica en último lugar, mientras que las heurísticas golosas de 4.6.2 se aplican siempre. También se incluyen heurísticas simples de redondeo de variables para los casos en que la relajación lineal está "cerca" de ser entera. (Más sobre esto en el capítulo 6).

El *branching* se realiza de acuerdo a alguna de las reglas de la sección 4.3.5.

4.11. Comentarios sobre estabilización

Finalizamos este capítulo con algunas notas sobre estabilización dual, aunque este tema no será abordado para SCA en este trabajo. Por lo general, todo algoritmo de generación de columnas sufre de un efecto de lenta convergencia, llamado *tailing off* en (Gilmore & Gomory, 1963), que significa que se obtienen mejoras poco significativas en cada iteración a medida que se acerca al óptimo de la relajación lineal. El motivo principal señalado entre otros por (Lübbecke & Desrosiers, 2005) es la gran oscilación de las variables duales (en especial durante las primeras iteraciones).

Entre los métodos propuestos para mitigar este inconveniente, uno de ellos es el de (du Merle et al., 1999), que consiste en mantener un "centro de estabilidad" en el dual, penalizando en el problema primal las soluciones duales que se alejan mucho de dicho centro. Este centro se modifica a medida que la solución dual converge a un óptimo, e igualmente se modifica la penalización hasta hacerla nula. Si bien surgen algunos inconvenientes para la implementación, este procedimiento es general y puede ser aplicado a cualquier problema.

Otro método, dependiente del problema, es acotar el espacio del dual por medio de desigualdades válidas (nuevas variables en el primal). Respecto a esto fueron introducidos los conceptos de desigualdades *dual-óptimas*, que son desigualdades válidas para la cara óptima del poliedro dual, y desigualdades *dual-óptimas profundas* que pueden cortar soluciones óptimas duales a excepción de una. Esto fue reportado con éxito para problemas de *Cutting Stock* (Ben Amor et al., 2006), y luego generalizado en (Gschwind & Irnich, 2016).

Capítulo 5

PROBLEMA DE PRICING DE SCA

En este capítulo se presentarán todos los elementos necesarios para desarrollar un algoritmo *branch-and-cut* para el problema de *pricing* de SCA, que de ahora en más notaremos SCAPP, y también algunas heurísticas. Además serán revisadas todas las variantes presentadas en el capítulo 4.

La entrada del problema SCAPP es un grafo $G = (V, E)$ sin puentes y coeficientes no negativos c_e y π_e para cada eje $e \in E$, además de otros coeficientes adicionales derivados de restricciones de *branching* y planos de corte del problema maestro.

El algoritmo para resolver SCAPP debe decidir si existe un ciclo en G tal que el valor de la función objetivo (3.41) es menor que cero. En caso de una respuesta positiva, un ciclo que satisface esa condición es también parte de la salida. Aunque solo se requiere satisfacer factibilidad se formula como un problema de optimización, como es habitual en el contexto de generación de columnas.

5.1. Coeficientes de entrada

- Por cada eje $e \in E$:
  - $c_e \in \mathbb{R}_{\geq 0}$: es el costo del eje proveniente del problema maestro SCA.
  - $\pi_e \in \mathbb{R}_{\geq 0}$: es el valor de la variable dual correspondiente a la restricción de demanda 3.36 del eje e.
  - $\nu_e \in \mathbb{R}$: es el valor de la variable dual correspondiente a la restricción de *branching* 4.3 ó 4.4 del eje e. Si no existe se asume $\nu_e = 0$.
  - $\rho_e \in \mathbb{R}_{\geq 0}$: es el valor de la variable dual correspondiente a la desigualdad 4.19 ó 4.24 para el eje e. Si no existe se asume $\rho_e = 0$.
  - ${\rho'}_e \in \mathbb{R}_{\geq 0}$: es el valor de la variable dual correspondiente a la desigualdad 4.25 para el eje e. Si no existe se asume ${\rho'}_e = 0$.
- Por cada vértice $u \in V$:
  - $\mu_u \in \mathbb{R}$: es el valor de la variable dual correspondiente a una restricción de *branching* 4.13 ó 4.14, o desigualdad 4.30 para el vértice u. Si no existe se asume $\mu_u = 0$.
- Un coeficiente $\sigma \in \mathbb{R}$ que corresponde a la variable dual de la restricción de *branching* 4.16 ó 4.17.
- Para cada restricción de *branching* 4.7 y 4.8 del problema maestro, con el conjunto $C \subseteq E$ asociado, existe un coeficiente de entrada $\tau_C \in \mathbb{R}$.
- Para cada desigualdad 4.31 del problema maestro, con el conjunto $T \subseteq E$ asociado, existe un coeficiente $\omega_T \in \mathbb{R}_{\geq 0}$.

5.2. Formulación del problema

Para simplificar las modificaciones realizadas por el agregado de planos de corte en el problema maestro presentadas en el capítulo 4, se realizará una modificación de las variables definidas para la formulación (3.41)-(3.45) de SCAPP. Para cada eje $e \in E$, la variable binaria r_e introducida en la sección 4.4.1 se utiliza en la siguiente expresión para reemplazar a la variable y_e.

$$y_e = 2r_e - x_e \qquad \forall e \in E \tag{5.1}$$

El cambio implica modificar la función objetivo de acuerdo a la ecuación 5.1 y el reemplazo de algunas restricciones.

Por otro lado, definimos los siguientes conjuntos:

$$\mathcal{C} = \{C : C \in \mathcal{P}(E), \text{ existe una restricción 4.7 ó 4.8 para } C \text{ en el problema maestro}\} \tag{5.2}$$

$$\mathcal{T} = \{T : T \in \mathcal{P}(E), \text{ existe una restricción 4.31 para } T \text{ en el problema maestro}\} \tag{5.3}$$

Teniendo en cuenta todas las modificaciones introducidas en el capítulo 4, las variables del modelo para SCAPP son

- Para cada $u \in V$

$$z_u = \begin{cases} 1 & \text{si } u \text{ es un vértice del ciclo} \\ 0 & \text{caso contrario} \end{cases}$$

- Para cada $e \in E$

$$x_e = \begin{cases} 1 & \text{si } e \text{ es un eje del ciclo} \\ 0 & \text{caso contrario} \end{cases}$$

$$r_e = \begin{cases} 1 & \text{si } e \text{ es un eje o cuerda del ciclo} \\ 0 & \text{caso contrario} \end{cases}$$

$$r'_e = \begin{cases} 1 & \text{si } e \text{ es una cuerda del ciclo} \\ 0 & \text{caso contrario} \end{cases}$$

- Para cada $C \in \mathcal{C}$

$$t_C = \begin{cases} 1 & \text{si } \forall e \in C \text{ tal que } e \text{ es eje del ciclo} \\ 0 & \text{caso contrario} \end{cases}$$

- Para cada $T \in \mathcal{T}$

$$q_T = \begin{cases} 1 & \text{si } \exists e \in T \text{ tal que } e \text{ es eje del ciclo} \\ 0 & \text{caso contrario} \end{cases}$$

La formulación es la siguiente.

$$\min \quad \sum_{e\in E}(c_e+\pi_e-\nu_e)\,x_e-\sum_{e\in E}(2\pi_e+\rho_e)r_e-\sum_{e\in E}\rho'_e r'_e$$
$$-\sum_{u\in V}\mu_u z_u-\sum_{C\in\mathcal{C}}\tau_C t_C-\sum_{T\in\mathcal{T}}\omega_T q_T-\sigma \tag{5.4}$$

s.t.
$$\sum_{v\in N_G(u)} x_{(u,v)}=2\times z_u \qquad \forall u\in V \tag{5.5}$$
$$\sum_{\substack{(u',v')\in E\\ u'\in S\\ v'\in V\setminus S}} x_{(u',v')}\geq 2z_u+2z_v-2 \qquad \begin{array}{l}\forall S\subset V,|S|>1,\\ \forall u\in S,\ \forall v\in V\setminus S\end{array} \tag{5.6}$$
$$r_{(u,v)}\leq z_w \qquad \forall (u,v)\in E,\forall w\in\{u,v\} \tag{5.7}$$
$$r'_e\leq r_e-x_e \qquad \forall e\in E \tag{5.8}$$
$$t_C\leq x_e \qquad \forall C\in\mathcal{C},\forall e\in C,\ \text{si } \tau_C>0 \tag{5.9}$$
$$\left(\sum_{e\in C}x_e\right)-t_C\leq |C|-1 \qquad \forall C\in\mathcal{C},\ \text{si } \tau_C<0 \tag{5.10}$$
$$q_T\leq\sum_{e\in T}r_e \qquad \forall T\in\mathcal{T} \tag{5.11}$$
$$q_T\leq 1 \qquad \forall T\in\mathcal{T} \tag{5.12}$$
$$\sum_{v\in V}z_v\geq 3 \tag{5.13}$$
$$x_e\in\{0,1\} \qquad \forall e\in E \tag{5.14}$$
$$z_u\in\{0,1\} \qquad \forall u\in V \tag{5.15}$$
$$r_e,r'_e\in\mathbb{R}_{\geq 0} \qquad \forall e\in E \tag{5.16}$$
$$t_C\in\mathbb{R}_{\geq 0} \qquad \forall C\in\mathcal{C} \tag{5.17}$$
$$q_T\in\mathbb{R}_{\geq 0} \qquad \forall T\in\mathcal{T} \tag{5.18}$$

Si bien todas las variables son binarias, solo se necesita esta restricción en el modelo para las variables x y z. La función objetivo 5.4 incluye todas las variables duales asociadas a las restricciones introducidas por *branching* o planos de corte en el problema maestro.

La restricción 5.7 asegura que $r_{(u,v)}$ puede tener valor 1 si ambos nodos u y v son usados. No es necesario contar con una cota inferior ya que su coeficiente $-(2\pi_e+\rho_e)$ en la función objetivo (minimización) es siempre negativo.

Utilizamos la definición de r_e para definir r'_e para cada $e\in E$ con la restricción 5.8. Del mismo modo que antes, no hace falta acotar inferiormente la variable.

Las restricciones 5.9 y 5.10 que definen el conjunto C de ejes correspondientes a una restricción de *branching* del problema maestro pueden omitirse dependiendo del signo del coeficiente τ_C correspondiente.

Las desigualdades 5.11 y 5.12 acotan el valor de la variable q_T con coeficiente negativo en la función objetivo.

La restricción 5.13 asegura que el ciclo vacío $(x, z) = (\overline{0}, \overline{0})$ se excluye como solución. Esto es necesario debido al coeficiente constante σ cuyo signo puede ser positivo o negativo.

El objetivo de este problema es encontrar columnas con costo reducido negativo (función objetivo de esta formulación), por lo tanto las soluciones con valor mayor o igual a cero no son factibles para SCAPP y debería existir una restricción que lo determine así. Sin embargo, esto no se incluye como restricción ya que el solver comercial utilizado permite incluir una cota superior para el valor de la función objetivo, que se establece en cero.

5.3. Desigualdades válidas y separación

El modelo presentado en la sección 5.2 no es compacto ya que tiene una cantidad exponencial de restricciones para garantizar un único ciclo en la solución. La formulación inicial para el algoritmo *branch-and-cut* está compuesta por todas las restricciones a excepción de (5.6). Las restricciones de eliminación de *subtour* generalizadas (GSECs) son separadas en caso de ser violadas. Además se derivan otras desigualdades válidas de la estructura de ciclo (Laporte & Martín, 2007) definida por las variables (x, z).

5.3.1. Desigualdades lógicas

Si un eje (u, v) es parte de un ciclo, entonces sus nodos u y v también son parte del ciclo. Esto se representa en las siguientes desigualdades (usadas previamente en (Leifer & Rosenwein, 1994)).

$$x_{(u,v)} \leq z_w \qquad \forall (u, v) \in E, \forall w \in \{u, v\} \tag{5.19}$$

Existe un número polinomial $O(|E|)$ de estas desigualdades y por lo tanto se pueden separar por simple enumeración con bajo costo computacional.

5.3.2. Restricciones de eliminación de *subtour* generalizadas (GSECs)

Las restricciones (5.6) se incluyen a medida que son necesarias debido a que existe un número exponencial de ellas. Dada la relajación lineal (x^*, z^*), se construye un grafo auxiliar $G' = (V', E')$ con los valores mayores a cero (fraccionarios o enteros) de las variables binarias.

$$\begin{aligned} V' &= \{u \in V \text{ tal que } z^*_u > 0\} \\ E' &= \{e \in E \text{ tal que } x^*_e > 0\} \end{aligned} \tag{5.20}$$

En un primer paso, se separa G' en componentes conexas, ya que cada par de componentes conexas diferentes determina una GSEC violada. En caso de obtener una única componente conexa, se ejecuta un algoritmo de corte mínimo, para el cual se define un peso en cada eje $e \in E'$ igual al valor x^*_e de la variable correspondiente. Un corte de peso menor que 2 implica una GSEC violada. Así, se insertan todas las GSECs correspondientes a cortes mínimos. Implementamos para esta tarea el algoritmo de Karger (Karger, 1993), el cual es un conocido algoritmo randomizado para encontrar un corte mínimo en un grafo no dirigido con pesos en sus ejes. En cada paso del algoritmo se selecciona aleatoriamente un

eje de G' y sus vértices se colapsan en uno solo, obteniendo un nuevo grafo con un vértice menos. Cuando solo quedan dos vértices restantes el algoritmo finaliza, obteniendo dos vértices que representan un par de conjuntos V_1 y V_2 tal que $V_1 \cup V_2 = V'$ y $V_1 \cap V_2 = \emptyset$. El valor del corte es

$$c(V_1, V_2) = \sum_{\substack{(u,v)\in E' \\ u\in V1,\ v\in V2}} x^*_{(u,v)}$$

Si $c(V_1, V_2) < 2$ el corte corresponde a GSECs 5.6 violadas, con $S = V_1$ ó $S = V_2$. Todo el procedimiento se repite una cantidad determinada de veces. La figura 5.1 muestra un ejemplo de ejecución existosa en el cual el corte obtenido tiene valor menor a 2.

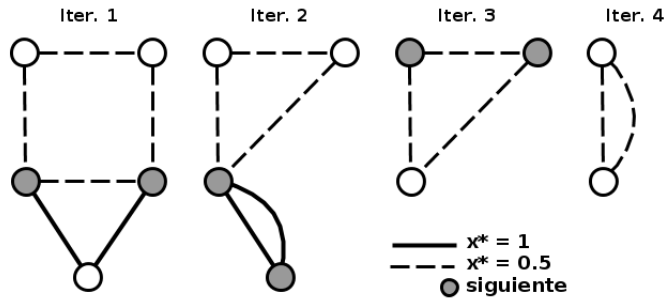


Fig. 5.1: Ejemplo de ejecución exitosa del algoritmo de Karger.

Debido a que es un algoritmo randomizado, puede fallar en encontrar todas las GSECs violadas y es necesario verificar cada solución primal para encontrarlas. (Estas se insertan como restricciones *lazy* en el solver comercial utilizado en la implementación).

5.3.3. Desigualdades peine

Una familia de desigualdades válidas conocidas para el problema de viajante de comercio son las llamadas *peine* (Grötschel & Padberg, 1979a,b), aplicadas también en (Balas, 1989). Involucran subconjuntos de V: un mango $H \subset V$ y t dientes $T_j \subset V$ para $j = 1, \ldots, j$. Si satisfacen

$$|T_j \cap H| \geq 1 \qquad j = 1, \ldots, j \tag{5.21}$$

$$|T_j \setminus H| \geq 1 \qquad j = 1, \ldots, j \tag{5.22}$$

$$T_i \cap T_j = \emptyset \qquad 1 \leq i < j \leq t \tag{5.23}$$

$$t \geq 3 \text{ e impar} \tag{5.24}$$

una desigualdad peine general es

$$\sum_{\substack{(u,v)\in E \\ \{u,v\}\subseteq H}} x_{(u,v)} + \sum_{j=1}^{t} \sum_{\substack{(u,v)\in E \\ \{u,v\}\subseteq T_j}} x_{(u,v)} \leq \sum_{w\in H} z_w + \sum_{j=1}^{t} |T_j| - \frac{3t+1}{2} \tag{5.25}$$

Si además satisfacen la condición siguiente se denominan peine simple

$$|T_j \cap H| = 1 \qquad j = 1, \ldots, j \tag{5.26}$$

Y las desigualdades peine simple se conocen como *2-matching* si

$$|T_j \setminus H| = 1 \qquad\qquad j = 1, \ldots, j \tag{5.27}$$

Reescribimos la desigualdad con $T = \{(u,v) : \{u,v\} = T_j \text{ para algún } j = 1, \ldots, t\}$.

$$\sum_{\substack{(u,v)\in E \\ \{u,v\}\subseteq H}} x_{(u,v)} + \sum_{(u,v)\in T} x_{(u,v)} \leq \sum_{w\in H} z_w + \frac{t-1}{2} \tag{5.28}$$

En nuestra implementación solo separamos desigualdades *2-matching* usando la heurística propuesta en (Fischetti et al., 1998). El procedimiento es el siguiente: dado el grafo G' definido en 5.20, se ejecuta el algoritmo de Kruskal (Kruskal, 1956) para árbol generador mínimo como soporte para obtener un H candidato en cada iteración. Cada vez que se agega un eje a la solución parcial de AGM, se establece H como la componente conexa de dicha solución que contiene a ese eje. Considerando el conjunto $\delta(H) = \{e_1, e_2, \ldots, e_p\}$ tal que cada eje de $\delta(H)$ tiene un vértice en H y otro en $V' \setminus H$ y $x^*_1 \geq x^*_2 \geq \cdots \geq x^*_p$, se establece $T = \{e_1, e_2, \ldots, e_t\}$ tal que $\sum_{i=1}^{t} x^*_i - \frac{t-1}{2}$ es máxima, con $3 \leq t \leq p$ y t impar. Si se verifica la desigualdad 5.28 con H y T no existe una desigualdad *2-matching* violada para H. Si ocurre el contrario, es necesario verificar que los dientes T sean disjuntos y eventualmente reparar los conjuntos. Mientras que existan dos ejes $e, e' \in T$ tal que comparten un vértice v, se reemplaza

- T por $T \setminus \{e, e'\}$.
- H por $H \setminus \{v\}$ si $v \in H$, ó por $H \cup \{v\}$ si $v \notin H$.

Por último, se debe verificar que luego de esta reparación $|T| > 1$, y si es el caso, se incorpora la desigualdad para H y T.

5.3.4. Desigualdades condicionales

Para ciertos valores de coeficientes de entrada, algunos subconjuntos de vértices podrían dar lugar solamente a ciclos con costo reducido mayor o igual a cero. A partir de la función objetivo se pueden derivar desigualdades válidas para descartar subconjuntos de V con esta característica. Estas desigualdades no son válidas para todas las instancias de entrada, pero en caso de serlo su uso es preferido al de las GSECs correspondientes.

Sea U un subconjunto de V ($U \subset V$). Definimos

$$E_U = \{(u,v) \in E \text{ tal que } \{u,v\} \subseteq U\} \tag{5.29}$$

$$cm(U) = \min_{\substack{E'\subseteq E_U \\ |E'|=|U|}} \sum_{e\in E'} (c_e + \pi_e - \nu_e) \tag{5.30}$$

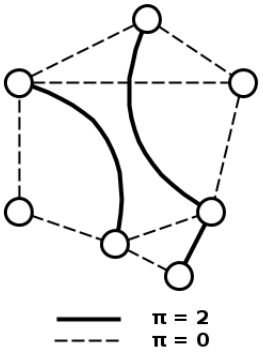
La restricción

$$1 + \sum_{e\in E_U} x_e \leq \sum_{u\in U} z_u \tag{5.31}$$

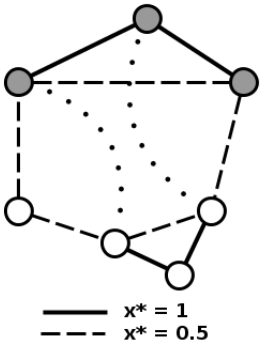
es una desigualdad válida si

$$cm(U) > \sum_{e\in E_U} (2\pi_e + \rho_e + \rho'_e) + \sum_{\substack{C\in\mathcal{C} \\ C\subseteq E_U}} \max(\tau_C, 0) + \sum_{\substack{T\in\mathcal{T} \\ T\cap E_U\neq\emptyset}} \omega_T + \sum_{u\in U} \mu_u + \sigma \tag{5.32}$$

Notar que es una condición suficiente pero no necesaria. $cm(U)$ es una cota inferior para el valor de la función objetivo asociado a los ejes de un posible ciclo incluido en U. El lado derecho de 5.32 es una cota superior para los valores positivos asociados a las variables duales del problema maestro (a excepción de ν).



(a) Grafo auxiliar con valores de π. $c_e = 1\ \forall e \in E$.



(b) Relajación lineal y separación.

Fig. 5.2: Ejemplo de desigualdad 5.32 violada.

Estas desigualdades son separadas junto con las GSECs del siguiente modo: por cada GSEC violada encontrada, se verifica la condición (5.32) y si resulta violada se agrega esta restricción en lugar de la GSEC correspondiente. (Se realiza un procedimiento similar en (Bérubé et al., 2009)). La figura 5.2 muestra un ejemplo simple solo con variables duales π (primeras iteraciones de generación de columnas), donde el subconjunto de vértices sombreado puede ser considerado como conjunto U. El mayor costo es calcular $cm(U)$ y por lo tanto se limita su aplicación para un tamaño de U acotado.

5.4. Comentarios finales del algoritmo

El orden de ejecución de los algoritmos de separación en nuestra implementación es el siguiente:

1. Desigualdades lógicas.

2. GSECs (con verificación de desigualdades condicionales).

3. Desigualdades *2-matching*.

Cada uno se aplica solamente si el anterior falló en la tarea de encontrar desigualdades válidas violadas.

Ya que el objetivo de este algoritmo es encontrar columnas de costo reducido negativo, las soluciones con valor de función objetivo ≥ 0 deben ser ignoradas. Como se mencionó previamente, se establece 0 como cota superior para podar el árbol de búsqueda. Además se verifica que el valor de la función objetivo sea estrictamente menor que cero antes de aceptar una solución.

Respecto al *branching*, las variables z correspondientes a vértices del grafo tienen prioridad sobre las variables x correspondientes a ejes. Este orden también fue propuesto en (Bérubé et al., 2009).

5.5. Heurísticas

Por cada instancia de SCA se deben resolver muchas instancias de SCAPP y por esto es conveniente contar con heurísticas para evitar utilizar el algoritmo exacto la mayor cantidad de veces posible. El enfoque fue puesto en resolver las instancias que surgen en el nodo raíz del algoritmo para SCA, y en particular para $\pi \geq 0$, $\rho \geq 0$, $\mu \geq 0$, $\omega_T = 0$. El motivo para esto es que la cantidad de iteraciones de *pricing* es mucho mayor en el nodo raíz que en el resto del árbol de búsqueda, además que las instancias de SCAPP son cada vez más difíciles de resolver a medida que la resolución de la relajación lineal del problema maestro correspondiente está más cerca de terminar.

En (Gendreau et al., 1995) se presenta una heurística para un problema de diseño de redes con anillos con función objetivo cuadrática, basada en el operador GENI presentado en (Gendreau et al., 1992) para el problema de viajante de comercio. Este problema es muy similar a SCAPP si solo se tienen en cuenta los coeficientes π. El operador GENI construye una solución iterativamente combinando una inserción con un operador 3-opt de mejora (Aarts & Lenstra, 2003). Los resultados reportados son muy satisfactorios, pero las instancias resueltas comprenden solamente grafos completos. Las instancias de SCAPP corresponden a redes de comunicaciones representadas por grafos generalmente ralos, y el algoritmo aplicado en (Gendreau et al., 1995) no obtiene los mismos resultados aplicado del mismo modo, ya que no es posible lograr una inserción en todas las iteraciones.

Respecto a los coeficientes de entrada, las instancias de SCAPP sobre las cuales se debe poner énfasis por ser derivadas de la generación de columnas tienen una cantidad relativamente baja de coeficientes $\pi_e > 0$ ($e \in E$) respecto a la cantidad total de ejes. Esta cantidad es aún menor para los demás coeficientes provenientes de variables duales del problema maestro. Observando las soluciones óptimas obtenidas con el algoritmo exacto, el ciclo correspondiente no siempre hace uso de todos los "premios" π, incluso en ocasiones solo toma una proporción muy pequeña. Se incluyen ejemplos en el apéndice C.

Desarrollamos dos heurísticas para resolver SCAPP teniendo en cuenta todo esto, para ser usadas en forma combinada. Una de ellas se basa en construir iterativamente ciclos, de forma parecida a GENI. La segunda busca obtener un ciclo lo más grande posible, quizá no factible, y repararlo. Además, las dos tienen características de la metaheurística GRASP (Feo & Resende, 1995; Resende & Ribeiro, 2010, 2016), para la cual incluimos un breve resumen a continuación.

5.5.1. GRASP (Greedy Randomized Adaptative Search Procedure)

GRASP es una metaheurística iterativa de dos fases: una construcción golosa y aleatoria, y una posterior búsqueda local. La mejor solución obtenida entre todas las iteraciones se guarda y devuelve como solución final. El esquema básico general es el presentado en el algoritmo 6.

Algoritmo 6 Esquema general de GRASP

```
1: función GRASP( V, E )
2:     f : función objetivo                                   ▷ Minimización
3:     mientras criterio de parada no satisfecho hacer
4:         Solución ← ∅                                       ▷ Construcción
5:         Inicializar conjunto de elementos candidatos
6:         Evaluar costo de cada elemento
7:         mientras Exista un elemento candidato hacer
8:             Construir lista restringida de candidatos (RCL)
9:             Seleccionar un elemento s de RCL al azar
10:            Agregar s a Solución
11:            Actualizar y reevaluar el conjunto de elementos candidatos
12:        fin mientras
13:        mientras Solución no es localmente óptima hacer    ▷ Búsqueda local
14:            Encontrar s' en vecindario de Solución con f(s') < f(Solución)
15:            Solución ← s'
16:        fin mientras
17:        Actualizar Mejor_Solución
18:    fin mientrasdevolver Mejor_Solución
19: fin función
```

En la primera fase se tiene en cuenta un conjunto de elementos que serán parte de la solución y se construye de forma iterativa agregando un elemento por cada iteración. Para ello, se evalúan todos los candidatos según una función de evaluación golosa, que usualmente representa un incremento en la función objetivo. Con los mejores elementos de acuerdo a esta función, se construye la Lista Restringida de Candidatos (RCL), cuyo tamaño es un parámetro del algoritmo. Luego se elige un elemento de la RCL al azar y se incorpora a la solución. Este procedimiento se repite hasta completar solución.

La segunda fase comprende una búsqueda local para mejorar la solución. La búsqueda local reemplaza de forma iterativa la solución por otra de su vecindario con mejor valor de función objetivo. El vecindario de una solución S es un conjunto de soluciones que pueden obtenerse al aplicar una modificación predefinida (operador) a una parte de S. Finaliza cuando no es posible encontrar un vecino con mejor valor, lo que significa que S es un óptimo local respecto al vecindario explorado.

Las dos etapas se repiten hasta alcanzar el criterio de parada, que puede ser un límite de iteraciones, tiempo o una cota para el valor de la función objetivo, entre otras cosas.

5.5.2. Elección de vértices iniciales

Ambas heurísticas introducidas para SCAPP tienen en común que eligen un vértice inicial para construir la solución. Esta selección es común a ambas y será presentada en

esta sección.

Se define una función puntaje $s(u)$ para cada vértice en función de los coeficientes con signo negativo en la función objetivo.

$$s(u) = \mu_u + \sum_{\substack{v \in V \\ (u,v) \in E}} \left(\pi_{(u,v)} + \rho_{(u,v)}\right) \quad \forall u \in V \tag{5.33}$$

Llamaremos vértices **significativos** al conjunto $V_s = \{v \in V,\ s(v) > 0\}$. Sea el grafo $G_s = (V_s, E_s)$ con $E_s = \{e \in E,\ \pi_e + \rho_e > 0\}$. En general, y por observación de las instancias de entrada, el grafo G_s no es conexo y además tiene pocos o ningún ciclo.

Se utilizará el conjunto de vértices iniciales $st(V) \subseteq V$ que cumple las siguientes condiciones:

- Cada vértice $u \in st(V)$ pertenece a una componente conexa distinta de G_s.
- Para cada vértice $u \in st(V)$ el valor de $s(u)$ es máximo en su componente conexa.

Además se limita el tamaño de $st(V)$ según un valor entero MAX_STARTS que es un parámetro de entrada. Al limitar el tamaño se eligen los vértices u de menor $s(u)$ para retirar de $st(V)$.

5.5.3. Heurística basada en inserciones

La idea general de esta heurística es similar a GENI: a partir de un ciclo inicial corto, por cada iteración se inserta un vértice y se intenta mejorar el valor de la función objetivo. A diferencia de GENI, no se espera lograr instertar todos los vértices al ciclo, ni siquiera todos los vértices significativos, sino que se busca obtener ciclos cortos con costo reducido negativo que no obtendría la segunda heurística, que se presenta en la sección 5.5.4.

Dado un vértice inicial v_0 seleccionado aleatoriamente de $st(V)$, se elige el adyacente v_1 con máximo $s(v_1)$. Puede ser no significativo, y en ese caso es uno arbitrario. Se construye un ciclo inicial que contiene a v_0 y v_1 por medio de un recorrido BFS, con v_0 como raíz ignorando el eje (v_0, v_1), y finalizando cuando se alcanza v_1. El objetivo es simplemente encontrar un ciclo con la menor longitud posible, y por eso no se tienen en cuenta los costos.

A partir del ciclo inicial, se incrementa la longitud del ciclo insertando un vértice por vez. Se evalúan todas los inserciones posibles, cada una compuesta por un eje a eliminar del ciclo y un nuevo vértice fuera del ciclo adyacente a los vértices del eje eliminado, usando la misma función objetivo de SCAPP. Como resultado de esta evaluación se construye una RCL con un tamaño máximo predeterminado y se elige una de las inserciones de la RCL aleatoriamente. La inserción se realiza siempre que $|\text{RCL}| > 0$ aunque implique incrementar el valor de la función objetivo. Después de cada inserción se verifica si es posible mejorar la solución con un operador 2-opt (Aarts & Lenstra, 2003) (que solo debe tomar en cuenta los ejes nuevos). Esta etapa de construcción termina cuando la RCL es vacía, condición que puede darse antes de incorporar todos los vértices significativos al ciclo.

Luego de la etapa anterior, se eliminan los vértices no significativos del ciclo, siempre y cuando sea posible hacerlo sin incrementar el valor de la función objetivo. Esto puede darse si como consecuencia de quitar un vértice, una cuerda e del ciclo con $\pi_e > 0$ pasa a ser eje del ciclo. La verificación se realiza en un orden arbitrario y determinístico.

En cada paso de inserción y al final de la eliminación se verifica y actualiza, de ser necesario, un conjunto de mejores soluciones a ser devuelto por el algoritmo. Todo el proceso se repite una cantidad determinada de veces. El algoritmo 7 es el esquema de esta heurística.

El éxito de esta heurística depende de la posibilidad de realizar inserciones que involucren tres vértices adyacentes entre sí. Por ello es posible que los ciclos encontrados sean muy cortos, ya que las instancias de SCA son grafos ralos. Además, es claro que la topología de las redes tambien influye, ya que de no haber triángulos en el grafo no se podría hacer este tipo de inserciones. Es por esto que el objetivo es simplemente obtener ciclos cortos con costo reducido negativo que no podría encontrar la segunda heurística, que se presenta a continuación.

Algoritmo 7 Heurística de inserciones aleatorias para SCAPP

función HEURISTICA1(V, E, ITERS, RCL_size)
  Inicializar conjunto de soluciones Z
  para todo $i = 1, \ldots,$ITERS **hacer**
    $v_0 \leftarrow$RANDOM($st(V)$)
    $v_1 \leftarrow \arg\max_{v \in N(v_0)} s(v)$
    $p_{01} \leftarrow$ camino más corto de v_0 a v_1 en $(V, E \setminus \{(v_0, v_1)\})$
    Inicializar C con (v_0, v_1) y p_{01}
    repetir
      Inicializar RCL vacía ▷ Construir RCL
      para todo $v_l \in V \setminus C$, $(v_a, v_b) \in C$, v_l adyacente a v_a y v_b **hacer**
        Agregar inserción $(v_l,\ (v_a, v_b))$ a RCL según función objetivo de SCAPP y RCL_size
      fin para
      si |RCL| > 0 **entonces**
        $v_l,\ (v_a, v_b) \leftarrow$RANDOM(RCL) ▷ $v_l \notin C$, $(v_a, v_b) \in C$
        $C \leftarrow C \setminus \{(v_a, v_b)\} \cup \{(v_a, v_l), (v_l, v_b)\}$
        Aplicar 2-opt a C con función objetivo de SCAPP
        Actualizar Z con C
      fin si
    hasta |RCL| $= 0$
    ELIMINARNOSIGNIFICATIVOS(C)
    Actualizar Z con C
  fin para
  devolver Z
fin función

5.5.4. Heurística basada en el problema Steiner TSP

Como segunda heurística y con el objetivo de encontrar posibles ciclos grandes con costo reducido negativo, adaptamos el trabajo de (Interian & Ribeiro, 2017) que es un algoritmo GRASP para Steiner TSP. La entrada de Steiner TSP es un grafo con pesos en sus ejes y con un subconjunto de vértices $S \subseteq V$. La salida es un ciclo de peso mínimo que pasa por todos los vértices de S, pasando opcionalmente por los vértices de $V \setminus S$. La idea de ese trabajo es convertir la instancia de Steiner TSP en una de TSP con un grafo

Algoritmo 8 Procedimiento auxiliar para heurísticas

```
1: función eliminarNoSignificativos(C)
2:     para todo vértice v ∈ C hacer
3:         si s(v) = 0 entonces
4:             Evaluar función objetivo de SCAPP para C \ {v}
5:             si valor objetivo se reduce entonces
6:                 C ← C \ {v}
7:             fin si
8:         fin si
9:     fin para
10: fin función
```

completo usando solo los vértices de S, y tomando como peso el resultado del camino mínimo entre cada par de vértices de S en el grafo original. Con esta nueva instancia se desarrolla un algoritmo GRASP usando como fase de construcción la heurística de vecino más cercano (el vértice inicial es arbitrario), y como fase de búsqueda local un operador 2-opt. Además introducen una etapa de *Path Relinking*, pero esto último no lo incorporamos a nuestro algoritmo.

Para resolver SCAPP con un esquema similar, se toman las siguientes consideraciones.

- Definimos S como el conjunto de vértices significativos V_s, y K_S como el grafo completo cuyos vértices son los del conjunto S.
- Dado un conjunto $S' \subseteq S$, todos los ciclos que pasan por todos los vértices de S' tienen los mismos valores si se consideran solo los coeficientes μ y ρ. En el caso de π, depende de los ejes usados.
- Se establecen los pesos en el nuevo grafo K_S con el valor del camino mínimo correspondiente en el grafo original G, tomando como peso $c_e + \pi_e$ para cada $e \in E$. Intuitivamente, por usar ese eje para el ciclo se pierde el doble "premio" de protección por cuerda.
- Se elige un vértice inicial aleatoriamente entre los seleccionados $st(V)$.
- La fase de construcción es también por vecinos más cercanos, randomizada para un tamaño de RCL establecido como parámetro. Esto significa que en cada iteración se selecciona una cantidad preestablecida de vértices de S aún no incorporados al ciclo tal que la distancia al último vértice incorporado sea menor o igual que todos los no seleccionados. Luego se elige uno de ellos aleatoriamente y se inserta al ciclo.
- La construcción finaliza cuando se alcanza un tamaño $|S|-k$, para $k = 0, 1, \ldots, k_{max}$. El motivo es poder explorar ciclos que no necesariamente utilicen todos los vértices significativos.
- La búsqueda local es 2-opt. Al igual que en (Interian & Ribeiro, 2017), la búsqueda local aplicada en K_S implica eliminar dos ejes del ciclo e insertar dos nuevos ejes de la única forma posible para preservar un ciclo. Esta operación implica, en el grafo original G, eliminar dos caminos del ciclo e insertar otros dos.

- Tanto en la búsqueda local como durante la construcción, los caminos (en G) insertados podrían pasar por vértices que ya se encuentran en el ciclo. Esto incluye a los vértices significativos S ya que alguno de ellos podría ser parte del camino mínimo entre otros dos. Al mismo tiempo, al eliminar caminos se podrían retirar vértices repetidos del ciclo (en G).

Algoritmo 9 Heurística GRASP para SCAPP

```
1: función HEURISTICA2(V, E, c, π, μ, ρ, S, k_max, RCL_size, ITERS)
2:     Inicializar conjunto de soluciones Z
3:     Matriz W ← caminos mínimos en V con peso c + π
4:     Inicializar pesos(S) con matriz W
5:     para todo k = 0, ..., k_max hacer
6:         para todo i = 1, ...,ITERS hacer
7:             v_l ←RANDOM(st(V))
8:             C ← [v_l]                          ▷ v_l: último vértice agregado
9:             mientras |C| < |S| − k hacer                    ▷ fase construcción
10:                Inicializar RCL vacía
11:                para todo v ∈ S \ C hacer
12:                    Agregar v a RCL según peso de (v_l, v) y RCL_size
13:                fin para
14:                v_l ←RANDOM(RCL)
15:                C ← C + [v_l]
16:            fin mientras
17:            Aplicar 2-opt a C con pesos(S)              ▷ fase búsqueda local
18:            C_G ← ciclo original correspondiente a C en (V, E)
19:            REPARAR(C_G)
20:            si C_G es válido entonces
21:                ELIMINARNOSIGNIFICATIVOS(C_G)
22:                Actualizar Z con C_G
23:            fin si
24:        fin para
25:    fin para
26:    devolver Z
27: fin función
```

El algoritmo 9 muestra el procedimiento completo. Al final de la construcción de un ciclo es posible que el ciclo correspondiente en el grafo original sea inválido por utilizar un mismo vértice más de una vez. Por este motivo, se verifican y realizan algunos arreglos simples.

1. Por cada camino $P = (v_1, v_2, v_3, v_2, v_4)$.
   - Si $(v_1, v_3) \in E$ se reemplaza P por (v_1, v_3, v_2, v_4).
   - Si $(v_3, v_4) \in E$ se reemplaza P por (v_1, v_2, v_3, v_4).
   - Si ninguno existe, se elimina v_3 reemplazando P por (v_1, v_2, v_4).
2. Si existen dos caminos $P_1 = (v_1, v_2, v_3)$ y $P_2 = (v_4, v_2, v_5)$ en el ciclo, se verifica si existe alguno de los ejes (v_3, v_5), (v_1, v_4), (v_1, v_3) ó (v_4, v_5).

- (v_3, v_5) reemplaza a (v_2, v_3) y (v_2, v_5).
- (v_1, v_4) reemplaza a (v_1, v_2) y (v_2, v_4).
- (v_1, v_3) reemplaza a (v_1, v_2) y (v_2, v_3).
- (v_4, v_5) reemplaza a (v_2, v_4) y (v_2, v_5).

Si luego de estos arreglos el ciclo sigue siendo inválido, se descarta. Si resulta válido se eliminan los vértices no significativos del mismo modo que se hacía en la heurística presentada en la sección 5.5.3.

5.5.5. Conjunto de soluciones

Se utilizan ambas heurísticas para cada instancia de SCAPP. Para esto se inicializa un conjunto de soluciones (ciclos) que será la respuesta a la llamada desde el algoritmo de generación de columnas. Este conjunto tiene dos parámetros.

1. Tamaño máximo: la cantidad máxima de columnas que se agregan al problema maestro producto del *pricing* heurístico. Agregar más de una columna por iteración resulta en general beneficioso, siempre y cuando no se agreguen demasiadas columnas irrelevantes para la resolución de la relajación lineal.

2. Máximo *gap* entre soluciones: por el mismo motivo, las columnas con costo reducido muy bajo (en valor absoluto) respecto a las mejores encontradas pueden resultar irrelevantes y retrasar la resolución de la relajación lineal.

Capítulo 6

EXPERIMENTOS COMPUTACIONALES

6.1. Implementación

El algoritmo fue implementado en C++ con SCIP 4.0.1 (Maher et al., 2017). SCIP fue compilado con CPLEX 12.7.1 (CPLEX, 2017) como solver de programación lineal. El algoritmo exacto presentado en el capítulo 5 para el problema de *pricing* fue implementado con CPLEX. Esto incluye la heurística para columnas iniciales de la sección 4.8 ya que utiliza el mismo algoritmo. La heurística primal presentada en la sección 4.6.1 que resuelve SCA de forma exacta para un número acotado de columnas también fue implementada con CPLEX.

6.1.1. SCIP

SCIP es un solver libre para uso académico y de código abierto para Programación Entera Mixta (MIP), Programación No Lineal Entera Mixta (MINLP) y Programación por Restricciones (CP), además de ser un framework para implementar algoritmos *branch-and-price-and-cut* que permite un gran control del proceso de resolución y acceso a la información. El último uso mencionado es el que le damos para este trabajo.

SCIP está implementado como biblioteca estática o dinámica en C que provee además clases adaptadoras para su uso con C++. SCIP está implementado de manera tal que se puede controlar y extender su funcionalidad a través de *plugins*. Por defecto, SCIP tiene un conjunto importante de *plugins* para resolver MIPs, que pueden ser usados o no según la necesidad del usuario. Además se pueden implementar nuevos *plugins* de usuario para agregar, por ejemplo, reglas de *branching* o algoritmos de separación de desigualdades válidas, teniendo disponible siempre la información completa de variables, restricciones, nodos (de *branch-and-bound*), etc.. Esto es una diferencia importante respecto a otros solvers (como CPLEX para citar un ejemplo), en los cuales la funcionalidad adicional se implementa a través de *callbacks* con una interfaz limitada para el contexto específico. Cada tipo de *plugin* se corresponde con algún aspecto del algoritmo *branch-and-price-and-cut*:

- Separadores: corresponden a los algoritmos de planos de corte para ser usados en *branch-and-cut*. Todas las implementaciones de separadores incluidas en SCIP se desactivan para nuestro algoritmo ya que hace falta mantener la consistencia con la estructura del problema de *pricing*.
- *Pricers*: este es el tipo correspondiente a la generación de columnas y, por defecto, SCIP solo tiene un único *pricer* que no puede ser desactivado y que simplemente reconsidera variables previamente eliminadas por estar mucho tiempo fuera de la base de LP.
- Reglas de *Branching*: al igual que los separadores, todas las implementaciones por defecto se desactivan en nuestra implementación.

- Heurísticas primales: activamos solo dos heurísticas muy simples de redondeo de las incluidas en SCIP.
- Selectores de nodos: usamos las implementaciones incluidas por defecto en SCIP.
- Preproceso: para reducir filas y/o columnas, o ajustar cotas de variables. En este trabajo no los incluimos.

Otros *plugins* correspondientes a Programación por Restricciones (CP) tampoco son incluidos.

Al agregar estos *plugins* es posible establecer algunos parámetros comunes que establecen el comportamiento del algoritmo. Por ejemplo, se puede establecer la frecuencia (que es un entero positivo) que indica la profundidad en el árbol de búsqueda en la que se aplica el *plugin* (debe ser múltiplo de esa frecuencia), así como también un *offset* para esta frecuencia. Otro parámetro importante es la prioridad que determina el orden a aplicar *plugins* del mismo tipo. Cada iteración del solver correspondiente al procesamiento de un nodo aplica *pricers*, separadores, heurísticas (estas tres por rondas) y finalmente reglas de *branching* según los parámetros configurados y *plugins* incorporados.

6.2. Instancias de prueba

Contamos con 21 instancias de (Pecorari, 2016) que corresponden a redes ópticas de comunicación reales. Los grafos correspondientes tienen entre 6 y 27 vértices y desde 10 hasta 85 ejes. La demanda máxima de los ejes de estas instancias está entre 6 y 14, y los costos tienen un promedio de entre 400 y 500. Los nombres de las instancias (que aparecen en las tablas de este capítulo) son *VZ_US_PIP_0**NN***. El número **NN** simplemente es un número de instancia y no guarda relación con la dimensión. Las instancias con número de 16 a 21 consisten en grafos de más de un millón de ciclos. Las instancias con número de 1 a 15 corresponden a grafos más pequeños y solo se utilizarán en la comparación con resultados de trabajos previos.

Para extender nuestros experimentos generamos nuevas instancias de mayor tamaño que agrupamos en cuatro grupos de acuerdo a dos criterios:

- Grafo ralo o denso.
- Demanda baja o alta.

Para el caso de la demanda establecemos como **baja** a un máximo de 7, que implica al menos 4 ciclos en la solución, y **alta** a un máximo de 31 que implica 16 ciclos como mínimo. Lo referente a la densidad del grafo se indicará en el procedimiento de generación de instancias explicado más adelante. A parte de esto, generamos un quinto grupo especialmente para los experimentos de la sección 6.8.3, que consiste en grafos ralos con baja desviación en las demandas. En la tabla 6.1 indicamos los nombres de estas instancias que estarán a lo largo del presente capítulo. El valor de **N** en el nombre de la instancia corresponde a la cantidad de vértices del grafo.

A partir de una instancia de TSPLIB (Reinelt, 1991) con peso igual a la distancia euclidiana, y la cantidad de vértices deseada N, el procedimiento para la generación es el siguiente.

Tab. 6.1: Instancias generadas

Instancias	Grafo	Demanda
sparse-low-**N**	Ralo	de 1 a 7
sparse-high-**N**	Ralo	de 1 a 31
dense-low-**N**	Denso	de 1 a 7
dense-high-**N**	Denso	de 1 a 31
dem-**N**	Ralo	de 5 a 7

1. Se seleccionan N vértices del grafo de entrada para generar el conjunto de vértices V de la instancia de SCA.

2. Se agregan los ejes correspondientes a un árbol generador mínimo del grafo.

3. Se agregan los ejes correspondientes a un árbol generador mínimo del grafo, pero sin tener en cuenta los ejes agregados en el paso anterior.

4. Se agregan ejes por cada vértice u para garantizar un grado mínimo. La cantidad agregada $d(u)$ es aleatoria, con distribución uniforme, con mínimo y máximo grado como parámetros. Los vértices adyacentes se eligen aleatoriamente entre los D más cercanos ($D > d(u)$), siendo D el grado elegido más el grado mínimo sobre 2.

5. (*Solo para grafo denso*): Se agregan los ejes necesarios para incluir dos subgrafos $K_{\lceil N/4 \rceil}$ y $K_{\lceil N/5 \rceil}$ disjuntos. Para cada uno se elige un vértice de forma aleatoria y se completa el subgrafo K_n con los $n-1$ vértices más próximos al elegido.

6. Se agregan los ejes necesarios para garantizar la ausencia de puntos de articulación.

7. Se asignan las **demandas** de forma aleatoria uniforme (máximo y mínimo son parámetros) para cada eje agregado anteriormente.

8. Se asignan los **costos** a cada eje como el peso (la distancia en la instancia original) con una perturbación aleatoria uniforme (de $\pm 25\,\%$). Luego se normaliza para tener un costo promedio de 400, similar a las instancias reales de (Pecorari, 2016).

No se tuvieron en cuenta grafos completos entre las instancias de prueba para no escapar de la motivación del problema que son redes de comunicaciones cuya topología está definida de antemano y cuya capacidad fue previamente asignada. No es posible "completar" el grafo asumiendo que los enlaces faltantes se pueden agregar usando el camino más corto (considerando el costo) entre los vértices correspondientes que ya que esto carece de sentido en el problema real en el cual la falla ocurre en todos los canales del enlace (eje del grafo).

6.3. Entorno

Los experimentos fueron ejecutados en un procesador Intel Core i7-4790 8x 4GHz con 8 GB de memoria RAM con Linux Kernel 5.4 (distribución Manjaro 20.2.1).

6.4. Configuración del algoritmo

Implementamos nuestro algoritmo de modo que sea posible desactivar algunas de sus características y elegir las reglas de *branching*. Siempre y cuando no se especifique otra configuración de parámetros, para realizar los experimentos elegimos la siguiente configuración.

- Planos de corte: se verifican las desigualdades 4.19, 4.24, 4.25 y 4.30. En el caso de 4.19 solo se verifican en el nodo raíz. En esta configuración no se incluyen las desigualdades de la sección 4.4.4.
- Se utilizan las tres heurísticas primales en la sección 4.6 y dos heurísticas de SCIP, que redondean la solución de LP para intentar obtener una solución entera. Una de ellas lo hace sin tener en cuenta factibilidad, la otra sí tiene en cuenta factibilidad. La heurística más costosa (4.6.1) se utiliza con la prioridad mínima en nodos de profundidad múltiplo de 5. Las heurísticas golosas (4.6.2) se utilizan en todos los nodos hasta profundidad 5 (el motivo se explica más adelante, en la sección 6.9). Las heurísticas de redondeo de SCIP se ejecutan en todos los nodos.
- El problema de *pricing* se resuelve de forma exacta, con *gap* 0 % de criterio de parada. En cada iteración se agregan como máximo 5 columnas. Los motivos para incluir menos de 5 columnas pueden ser:
  1. El solver encontró menos de 5 soluciones factibles para la instancia del problema de *pricing*.
  2. Se descartaron soluciones un valor absoluto de función objetivo (costo reducido) menor al 90 % del mejor encontrado.
- Reglas de *branching*: las reglas aplicadas en orden de prioridad son las siguientes:
  1. En cantidad total de ciclos (4.16 y 4.17).
  2. En variables de vértices de la formulación original (4.13 y 4.14).
  3. En variables de ejes de la formulación original (4.3 y 4.4). (Incluye pares y 3-tuplas de ejes - 4.7 y 4.8).
  4. En variable simple del Problema Maestro Restringido. (4.1 y 4.2)

  Junto a la primera regla, se utiliza la cota 4.18 (indicada en la sección 4.3.6) para finalizar prematuramente el *pricing* donde corresponda. En el caso de *early branching* la condición es que la cota sea superior al promedio entre las cotas dual y primal globales. (Este 50 % se eligió arbitrariamente).
- Selección de nodos: no realizamos una implementación propia para la selección del siguiente nodo a procesar. La regla de selección elige el nodo con la mejor cota dual estimada (esto depende de su nodo padre). Sin embargo, como el *framework* da la posibilidad de establecer una prioridad para la selección de nodos, utilizamos esta característica asignando una prioridad de acuerdo a la restricción de *branching* aplicada al crear el nodo. El orden (de mayor a menor prioridad) es el siguiente:
  1. Cantidad de ciclos total. (4.16 y 4.17)

2. En variable de vértice. (4.13 y 4.14)
3. En variable de eje. (4.3 y 4.4)
4. En par de variables de ejes. (4.7 y 4.8 con conjunto de tamaño 2)
5. En tres variables de ejes. (4.7 y 4.8 con conjunto de tamaño 3)
6. *Branching* "débil". (4.1 y 4.2)

- Columnas iniciales: Para la construcción de las columnas iniciales utilizamos el mismo algoritmo de pricing para construir una solución factible de SCA sin tener en cuenta el costo de los ejes, como se explica en la sección 4.8. Cada ciclo de la solución se agrega como columna inicial y por lo tanto el Problema Maestro Restringido inicial es factible.

En todos los casos y salvo que se indique lo contrario, el tiempo de ejecución máximo es de 10 minutos. Los tiempos de ejecución se informan en todas las tablas en segundos.

También se estableció un máximo de 5000 nodos para procesar.

6.5. Comparación con trabajos previos

En primer lugar presentaremos los resultados obtenidos por nuestro algoritmo comparados con los resultados reportados en (Pecorari, 2016). Contamos solo con el mejor valor obtenido de función objetivo y el tiempo máximo de ejecución. Es importante aclarar que el hardware utilizado por (Pecorari, 2016) fue un procesador Intel Core i3 2.2GHz con 8GB de RAM corriendo en sistema operativo Windows 10 Pro, y en los casos que correspondían, se utilizó IBM ILOG CPLEX 12.6.

Los resultados de (Pecorari, 2016) comparan distintos algoritmos que incluyen heurísticas de la literatura, modelos de programación lineal entera mixta compactos de la literatura y presentados en aquel trabajo, algoritmos basados en Programación por Restricciones (CP), una heurística GRASP híbrida cuya fase de búsqueda local se plantea y resuelve como un problema de programación entera mixta, y un algoritmo de generación de columnas. De todos ellos se presentan los que obtuvieron los mejores resultados. Es importante aclarar que en el caso de los modelos MIP compactos se ajustó la cantidad de bloques idénticos (notada como J al igual que el presentado en el capítulo 3) de forma tal que sea igual a la cantidad de ciclos de la solución óptima correspondiente. Si no se conoce, se podría sobrestimar y perjudicar mucho el rendimiento debido a la simetría, o subestimar perdiendo la posibilidad de encontrar una solución óptima.

En la tabla 6.2 se presentan los resultados de (Pecorari, 2016) con 5 minutos de tiempo de ejecución máximo y en la tabla 6.3 con 30 minutos. Las columnas de ambas tablas son iguales. La columna MIP corresponde al mejor resultado de 4 modelos compactos, la columna GRASP corresponde a la heurísica GRASP híbrida y CG al algoritmo de generación de columnas. El trabajo de (Pecorari, 2016) también compara los resultados de otros algoritmos que no fueron incluidos por tener un rendimiento aún menor. Las tres columnas restantes corresponden al algoritmo *branch-and-price-and-cut* presentado en este trabajo. Se presenta el valor de la función objetivo obtenido, el gap entendido como

$$100 \times \frac{\text{cota primal} - \text{cota dual}}{\text{cota dual}}$$

Tab. 6.2: Comparación con (Pecorari, 2016) con 5 minutos de límite de tiempo

	MIP	GRASP	CG	BPC		
Instancia				Objetivo	Gap	Tiempo
VZ_US_PIP_001	32300	35750	32255	32240	**0**	0.34
VZ_US_PIP_002	37370	40900	37370	37370	**0**	0.33
VZ_US_PIP_003	35170	36730	35170	35170	**0**	0.1
VZ_US_PIP_004	39980	42390	39980	39980	**0**	0.28
VZ_US_PIP_005	24937	27044	25066	24937	**0**	0.28
VZ_US_PIP_006	26190	27430	26190	26190	**0**	0.31
VZ_US_PIP_007	30534	32787	30534	30534	**0**	0.1
VZ_US_PIP_008	32721	34089	32721	32721	**0**	1.51
VZ_US_PIP_009	37071	40094	37071	37071	**0**	0.35
VZ_US_PIP_010	44084	45984	44094	44084	**0**	0.2
VZ_US_PIP_011	42054	43572	42154	42054	**0**	0.18
VZ_US_PIP_012	35754	36754	35754	35754	**0**	0.23
VZ_US_PIP_013	29984	30792	29984	29984	**0**	0.03
VZ_US_PIP_014	14740	14740	14740	14740	**0**	0.01
VZ_US_PIP_015	14775	14775	14775	14775	**0**	0.01
VZ_US_PIP_016	36310	39400	36500	36310	**0**	0.56
VZ_US_PIP_017	42180	41880	38780	38780	**0**	30.75
VZ_US_PIP_018	58420	68350	58350	**58250**	**0**	25.93
VZ_US_PIP_019	169520	82570	72370	**72050**	**0**	16.41
VZ_US_PIP_020	-	60860	56470	**56170**	**0**	25.72
VZ_US_PIP_021	-	53990	47595	**46670**	**0**	205.14

y el tiempo de ejecución en segundos. Nuestro algoritmo obtiene las soluciones óptimas para todas las instancias (*gap* cero), y en cuestión de segundos para la mayoría de ellas. Las instancias **018** a **021** fueron resueltas de forma óptima, lo que no ocurre con los trabajos previos aún con los 30 minutos de tiempo de ejecución. Podemos decir así que el rendimiento del algoritmo *branch-and-price-and-cut* es superador respecto a los resultados reportados previamente.

6.6. Presentación de resultados

Las secciones siguientes contienen experiementos realizados principalmente con las instancias generadas de la sección 6.2. En este capítulo se incluyen los resultados de forma agregada, distinguiendo entre grupos distintos de instancias. En las tablas, junto al nombre del grupo de instancias, se incluye entre paréntesis la cantidad de instancias del grupo usadas para el experimento. Las columnas comunes a todas las tablas son:

- Mejor: Es la cantidad de veces que se encontró la mejor solución con la configuración de parámetros correspondiente, considerando solo los resultados de las configuraciones comparadas en el experimento.

- Gap: Promedio del *gap* del conjunto de resultados.

Tab. 6.3: Comparación con (Pecorari, 2016) con 30 minutos de límite de tiempo

	MIP	GRASP	CG	BPC		
Instancia				Objetivo	Gap	Tiempo
VZ_US_PIP_001	32240	34260	32255	32240	**0**	0.34
VZ_US_PIP_002	37370	39790	37370	37370	**0**	0.33
VZ_US_PIP_003	35170	36860	35170	35170	**0**	0.1
VZ_US_PIP_004	39980	43070	39980	39980	**0**	0.28
VZ_US_PIP_005	24937	25596	25038	24937	**0**	0.28
VZ_US_PIP_006	26190	27268	26190	26190	**0**	0.31
VZ_US_PIP_007	30534	32297	30534	30534	**0**	0.1
VZ_US_PIP_008	32736	34241	32721	32721	**0**	1.51
VZ_US_PIP_009	37071	40544	37071	37071	**0**	0.35
VZ_US_PIP_010	44084	45912	44084	44084	**0**	0.2
VZ_US_PIP_011	42054	43609	42054	42054	**0**	0.18
VZ_US_PIP_012	35754	37067	35754	35754	**0**	0.23
VZ_US_PIP_013	20984	30122	29984	29984	**0**	0.03
VZ_US_PIP_014	14740	14740	14740	14740	**0**	0.01
VZ_US_PIP_015	14775	14775	14775	14775	**0**	0.01
VZ_US_PIP_016	36310	38320	36500	36310	**0**	0.56
VZ_US_PIP_017	38860	41700	38780	38780	**0**	30.75
VZ_US_PIP_018	58420	67970	58350	**58250**	**0**	25.93
VZ_US_PIP_019	74440	82340	72370	**72050**	**0**	16.41
VZ_US_PIP_020	72110	59600	56470	**56170**	**0**	25.72
VZ_US_PIP_021	70710	52320	47430	**46670**	**0**	205.14

- Tiempo: Promedio del tiempo de ejecución en segundos del conjunto de resultados.

En cada sección se explicarán las columnas adicionales en caso de que sea necesario. Las tablas completas con los resultados desagregados se encuentran en el apéndice A.

6.7. Reglas de *branching*

6.7.1. Prioridad en variables correspondientes a vértices o a ejes

Comparamos el desempeño del algoritmo utilizando las distintas reglas de *branching* propuestas en la sección 4.3.5. Cabe recordar que las restricciones de *branching* incorporadas aplican a una variable correspondiente a un eje o vértice en la formulación original.

Es posible que en algún nodo no sea posible aplicar alguna de estas reglas, por lo que para el caso de ejes presentamos como alternativa la reglas 4.7-4.8 que toman en cuenta un subconjunto de ejes. Nuestra implementación solo considera conjuntos de hasta tres ejes, y en caso de no encontrar un subconjunto, se aplica el *branching* en alguna de las variables del Problema Maestro a pesar de las desventajas del desbalance en el árbol de búsqueda.

En caso de no poder encontrar restricciones de *branching* para una variable (original) de vértice, se procede del modo explicado en el paso anterior buscando aplicar *branching* en alguna variable (original) correspondiente a un eje.

La elección de la variable para el *branching* se hace de acuerdo a su infactibilidad en la relajación lineal. Para el caso de ejes tenemos dos alternativas, una es tomar la infactibilidad de todas las variables por igual y la otra es tomarla ponderada por el coeficiente de costo del eje. Como breve resumen, las reglas son las siguientes:

1. Prioridad en ejes. Máxima infactibilidad.
2. Prioridad en ejes. Máxima infactibilidad ponderada por costo del eje.
3. Prioridad en vértice. Luego regla 1.
4. Prioridad en vértice. Luego regla 2.

Tab. 6.4: Reglas de branching con prioridad en ejes.

	Solo infactibilidad			Ponderado por costo		
	Mejor	Gap	Tiempo	Mejor	Gap	Tiempo
Instancias						
VZ_US_PIP(6)	6	**0.23**	**428.05**	6	0.24	441.89
sparse-low(8)	2	1.12	552.94	**8**	**0.87**	**491.86**
sparse-high(8)	**7**	**0.09**	547.21	5	0.1	**541.89**
dense-low(8)	3	0.93	600.0	**7**	**0.82**	**586.08**
dense-high(8)	4	0.09	581.9	**7**	**0.06**	**442.24**

En la tabla 6.4 se comparan los resultados de las dos reglas con prioridad en ejes (reglas 1 y 2). (Los resultados completos se encuentran en la tabla A.1) Se puede ver una leve ventaja general al ponderar por costos, que se acentúa en los casos que las demandas de los ejes son bajas.

Tab. 6.5: Reglas de branching: prioridad en vértice vs prioridad en ejes.

	Primero en vértice			Branch en ejes		
	Mejor	Gap	Tiempo	Mejor	Gap	Tiempo
Instancias						
VZ_US_PIP(6)	6	**0.0**	**39.01**	6	0.23	428.05
sparse-low(8)	**7**	**0.61**	**461.54**	3	1.12	552.94
sparse-high(8)	6	**0.05**	**322.33**	**7**	0.09	547.21
dense-low(8)	**8**	**0.46**	**480.06**	2	0.93	600.0
dense-high(8)	**8**	**0.0**	**309.82**	3	0.09	581.9

Comparamos prioridad en vértice contra prioridad en ejes (reglas 1 y 3) en la tabla 6.5 (resultados completos en tabla A.2). Por lo visto en otros trabajos de la literatura (algoritmos *branch-and-cut*) para problemas similares esperamos ver una clara ventaja al establecer prioridad en vértices, lo que parece ocurrir efectivamente.

Por último, en la tabla 6.6 (completos en tabla A.3) vemos los resultados con prioridad en vértice y las dos distintas reglas de *branching* en eje como regla secundaria (reglas 3 y 4). Los resultados muestran un rendimiento similar, apenas a favor de la utilización de

la primera regla (no ponderar por costo) cuando no es posible aplicar *branching* en una variable original de vértice.

En vista de estos resultados, establecer la prioridad de *branching* en las variables originales correspondientes a vértices es determinante para obtener un mejor rendimiento del algoritmo.

Tab. 6.6: Branching: Prioridad en vértice. Resultados según branching en ejes como regla secundaria.

	Solo infactibilidad			Ponderado por costo		
	Mejor	Gap	Tiempo	Mejor	Gap	Tiempo
Instancias						
VZ_US_PIP(6)	6	0.0	**39.01**	6	0.0	47.2
sparse-low(8)	8	0.61	**461.54**	8	0.61	461.56
sparse-high(8)	7	0.05	322.33	**8**	0.05	**319.61**
dense-low(8)	**8**	0.46	**480.06**	7	0.46	481.88
dense-high(8)	**8**	**0.0**	309.82	6	0.01	**300.31**

Una observación importante que no puede apreciarse en las tablas anteriores es que en ningún caso fue necesario aplicar una regla de *branching* en tres ejes y por lo tanto tampoco una regla "débil" (una sola variable del problema maestro). Esto ocurrió no solo en los experimentos de esta sección sino que también en la totalidad de los experimentos realizados. A modo de ejemplo, se incluye la cantidad de restricciones de *branching* por cada tipo en la tabla 6.7. El valor reportado es la suma correspondiente a cada grupo de instancias. Los experimentos correspondientes son los mismo de la tabla 6.5.

Tab. 6.7: Reglas de branching: cantidad por tipo de restricción.

	Primero en vértice					Branch en ejes				
	Vértice	Eje	2 ejes	3 ejes	Débil	Vértice	Eje	2 ejes	3 ejes	Débil
Instancias										
VZ_US_PIP(6)	928	56	10	0	0	0	18876	880	0	0
sparse-low(8)	1161	2	6	0	0	0	1988	7	0	0
sparse-high(8)	2165	152	192	0	0	0	5421	230	0	0
dense-low(8)	819	45	10	0	0	0	1608	23	0	0
dense-high(8)	1160	118	140	0	0	0	4212	21	0	0

6.7.2. Cota superior para cantidad de ciclos

En esta sección presentamos los experimentos para evaluar la restricción de *branching* faltante. Esta no se trata de una regla en sí, sino que fue introducida para poder calcular con facilidad una cota inferior que permita, en los nodos que tengan la restricción de branching 4.16, finalizar el proceso de *pricing* prematuramente.

Los resultados se presentan en la tabla 6.8. Las primeras tres columnas corresponden a la configuración por defecto del algoritmo: utilizamos la cota inferior para finalizar prematuramente el *pricing*, ya sea porque es mayor que la cota primal o porque relizamos

early branching. Las siguientes tres columnas presentan los resultados al desactivar *early branching*, pero manteniendo el uso de la cota para finalizar la generación de columnas si es mayor o igual a la cota primal. Las últimas tres columnas muestran los resultados sin usar esta regla, que corresponden a los experimentos de prioridad de *branching* en vértice. Son pocos los casos donde observamos un benefecio al utilizar esta cota, que pueden encontrarse en la tabla completa A.4.

El motivo principal que observamos por el cual no se logra una mejora general en el rendimiento del algoritmo es que las cotas inferiores (relajaciones lineales) obtenidas al aplicar las restricciones de *branching* 4.16 y 4.17 son menores que las obtenidas al aplicar *branching* en variables correspondientes a vértices del grafo. La cantidad de nodos procesados por tiempo es similar en todos los casos, pero la cantidad total de nodos necesarios es mayor con 4.16 y 4.17 por el motivo mencionado anteriormente, de modo que no se logra aprovechar la finalización temprana de la generación de columnas.

Tab. 6.8: Branching: con o sin cota de cantidad de ciclos.

	Con early branch			Sin early branch			Regla vértice		
	Mejor	Gap	Tiempo	Mejor	Gap	Tiempo	Mejor	Gap	Tiempo
Instancias									
VZ_US_PIP(6)	6	0.0	50.75	6	0.0	54.13	6	0.0	**39.01**
sparse-low(10)	5	1.73	550.36	6	1.66	550.32	**7**	**1.65**	**541.39**
sparse-high(10)	**8**	**0.12**	467.44	6	0.14	457.57	7	0.13	**425.86**
dense-low(10)	6	0.98	525.68	6	0.98	525.53	**9**	**0.83**	**504.04**
dense-high(10)	8	**0.03**	458.46	7	0.04	459.81	**9**	**0.03**	**391.62**

6.8. Planos de corte

La configuración por defecto del algoritmo incluye planos de corte que son incorporados, en su mayoría, sin modificar la estructura del problema de *pricing*. En esta sección presentamos los resultados obtenidos al desactivar los algoritmos de separación propuestos, para verificar el beneficio de su aplicación en la práctica.

6.8.1. Planos de corte asociados a ejes

En primer lugar, observamos en la tabla 6.9 los experimentos correspondientes la los cortes asociados a ejes. Esto incluye tanto las cotas 4.19 (aprovechadas para las demandas impares) de la sección 4.4.1 como las desigualdades 4.24 y 4.25 para fortalecer las restricciones de *branching*, de la sección 4.4.2. Agregamos en la tabla la cantidad de cortes totales agregados en cada grupo de instancias. Esto se indica como **OD** para 4.19 y como **BR** para la suma de 4.24 y 4.25. En la tabla desagregada A.5 estas cantidades coresponden a las de cada instancia.

Si bien no se aprecia una gran cantidad de desigualdades aplicadas, observamos que el rendimiento del algoritmo se ve beneficiado en presencia de estas desigualdades. Cabe recordar que las desigualdades para demanda impar se aplican solo en el nodo raíz. Y por motivos obvios, las desigualades asociadas a restricciones de *branching* se aplican en los subnodos. Además, estas últimas requieren que previamente en la rama a la cual

Tab. 6.9: Resultados obtenidos al desactivar los cortes asociados a ejes.

	Con cortes					Sin cortes				
	Mejor	Gap	Tiempo	OD	BR	Mejor	Gap	Tiempo	OD	BR
Instancias										
VZ_US_PIP(6)	6	**0.0**	**50.75**	41	4	6	0.07	111.28	0	0
sparse-low(11)	**8**	**1.58**	**507.51**	86	0	5	1.87	548.06	0	0
sparse-high(11)	**9**	**0.11**	**436.41**	61	2	6	0.16	515.72	0	0
dense-low(11)	**7**	**0.89**	**532.44**	129	1	6	1.13	569.72	0	0
dense-high(11)	**10**	**0.03**	**429.02**	69	4	6	0.07	530.68	0	0

pertenecen se encuentre aplicada alguna regla de *branching* en ejes. Esto es un motivo por el cual no se encuentren muchas veces aplicadas, ya que la prioridad del *branching* está en los vértices.

6.8.2. Planos de corte asociados a vértices

Verificamos que las cotas de uso de vértices 4.30 presentadas en la sección 4.4.3 también influyen en el desempeño del algoritmo. Se presentan los resultados obtenidos en la tabla 6.10 y se incluye la siguiente información adicional por cada grupo de instancias (o por cada instancia en la tabla completa A.6):

INF es la cantidad de subnodos no factibles luego de aplicar alguna regla de *branching*. Esto fue probado por el algoritmo de *pricing*.

REP es la cantidad de subnodos que en principio resultaron no factibles luego de aplicar alguna regla de *branching*, pero luego se encontró una nueva columna con la cual resultó factible.

Tab. 6.10: Resultados obtenidos al desactivar los cortes asociados a vértices (cota inferior).

	Con cortes					Sin cortes				
	Mejor	Gap	Tiempo	REP	INF	Mejor	Gap	Tiempo	REP	INF
Instancias										
VZ_US_PIP(6)	6	0.0	**50.75**	0	0	6	0.0	58.07	1	10
sparse-low(11)	**8**	**1.58**	507.51	1	1	5	2.01	**507.48**	16	38
sparse-high(11)	**11**	**0.11**	**436.41**	0	0	6	0.2	449.07	0	29
dense-low(11)	**8**	**0.89**	**532.44**	16	2	6	1.06	535.47	40	37
dense-high(11)	10	0.03	**429.02**	0	6	10	0.03	429.67	1	32

Podemos ver en la tabla que aplicar estas desigualdades por lo general resulta beneficioso para el rendimiento del algoritmo. Además de eso vemos que, en presencia de estas desigualdades, es muy escasa la cantidad de nodos no factibles para los cuales hace falta ejecutar al menos una iteración de *pricing* Farkas para generar una columna o probar que el nodo es efectivamente no factible.

6.8.3. Planos de corte generalizados para ejes

Incorporamos las desigualdades de la sección 4.4.4 pero no obtuvimos resultados distintos con las instancias presentadas en la sección 6.2. Un posible motivo para esto es la gran desviación en los valores de las demandas. Por lo tanto generamos un nuevo grupo de instancias, con grafos ralos y demandas entre 5 y 7, usando el procedimiento explicado previamente en la sección 6.2.

Las desigualdades 4.31 fueron definidas de forma general para un conjunto $T \subseteq E$ de ejes, pero en nuestra implementación solo se consideran pares. Además, se verifican los pares de ejes asumiendo verdadera la hipótesis de la ausencia del ciclo que tiene a ambos ejes como cuerdas. Si se encuentra un par de ejes correspondientes a una desigualdad 4.31 posiblemente violada, se verifica que no exista el ciclo. Esto se puede hacer con el algoritmo para resolver el problema *pricing* ya que es un caso particular del mismo. Además, estas desigualdades solo se verifican en el nodo raíz del árbol de búsqueda.

Los resultados los experimentos realizados con estas instancias se presentan en la tabla 6.11. Corresponde a la configuración por defecto del algoritmo comparada cuando se incorporan las desigualdades 4.31. Se incluye también la cota inferior obtenida al final de la ejecución, que en todos los casos es más ajustada cuando se aplican los cortes. (La cota inferior al final del procesamiento del nodo raíz es siempre igual o más ajustada al aplicar los cortes). Esto lleva a obtener un mejor *gap* en general, aunque no siempre se obtiene una mejor solución para el mismo límite de tiempo de ejecución.

Tab. 6.11: Resultados obtenidos al activar los cortes asociados a pares de ejes.

	Con cortes				Sin cortes			
	Objetivo	Gap	Tiempo	Cota inferior	Objetivo	Gap	Tiempo	Cota inferior
Instancia								
dem45	**100880**	**0.53**	600.0	**100352.5**	101620	1.34	600.0	100275.83
dem46	106240	**0.48**	600.0	**105735.4**	106240	1.12	600.0	105065.96
dem47	**106670**	**0.59**	600.0	**106041.67**	106690	0.72	600.0	105928.67
dem48	109400	0.96	600.0	**108355.0**	**109320**	**0.94**	600.0	108307.22
dem49	**104550**	**0.31**	600.0	**104231.01**	104700	0.84	600.0	103823.98
dem50	112300	**0.33**	600.0	**111925.83**	**112270**	0.39	600.0	111836.67
dem51	114850	**0.48**	600.0	**114303.33**	**114790**	0.49	600.0	114226.4
dem52	106560	**1.28**	600.0	**105216.83**	**105980**	1.46	600.0	104455.83
dem53	121060	0.50	600.0	**120463.56**	**120970**	**0.48**	600.0	120397.17
dem54	**113220**	**0.29**	600.0	**112892.0**	113390	0.53	600.0	112791.11
dem55	115470	**1.79**	600.0	**113439.88**	**114910**	1.90	600.0	112765.42
dem56	123670	**1.16**	600.0	**122248.99**	**123510**	1.20	600.0	122045.65
dem57	129550	1.43	600.0	**127725.78**	**129350**	**1.39**	600.0	127574.33
dem58	**135850**	**1.13**	600.0	**134326.33**	136740	1.96	600.0	134111.24
dem59	135030	0.05	600.0	134960.0	**134970**	**0.01**	600.0	134960.0
dem60	134790	1.79	600.0	**132416.55**	**133530**	**0.87**	600.0	132376.45

6.9. Heurísticas primales

En la sección 4.6 presentamos dos heurísticas primales golosas y una heurística planteada como IP que es una adaptación de RINS a este algoritmo. Las heurísticas golosas tienen un doble objetivo, obtener soluciones enteras de la mejor calidad posible y proveer soluciones enteras para ser mejoradas por RINS. La heurística RINS (sección 4.6.1) es la resolución exacta para un conjunto muy acotado de variables, que toma en cuenta la mejor solución entera encontrada y la relajación lineal del nodo donde se ejecuta. La ejecución está limitada a 1000 nodos, 10 segundos de tiempo máximo de ejecución, y se resuelve con CPLEX con su configuración de parámetros por defecto.

A las heurísticas primales de la sección 4.6 agregamos dos heurísticas implementadas en SCIP que construyen una solución a partir de la solución a la relajación lineal. Una de ellas simplemente redondea las variables con valor fraccionario en orden arbitrario y luego verifica si la solución entera es factible. La otra hace lo mismo pero teniendo en cuenta la factibilidad de la solución al momento de redondear.

Se realizaron algunas pruebas preliminares con otras heurísticas implementadas por SCIP pero no se obtuvieron buenos resultados y por lo tanto fueron descartadas.

Los experimentos realizados consistieron en desactivar las heurísticas implementadas en este trabajo para evaluar su desempeño dentro del algoritmo. Si se desactiva RINS resulta más difícil encontrar soluciones enteras de calidad, como puede verse en la tabla 6.12 (completa A.7), por lo que es muy importante a pesar de su costo computacional. (El tiempo en las instancias de mayor tamaño fue de aproximadamente 7 segundos). Un punto importante es que RINS obtiene soluciones de muy buena calidad desde el nodo raíz, lo cual favorece significativamente el rendimiento global del algoritmo.

Tab. 6.12: Heurísticas primales (RINS).

	Todas			Sin RINS		
	Mejor	Gap	Tiempo	Mejor	Gap	Tiempo
Instancias						
sparse-low(6)	**6**	**2.77**	600.0	0	7.45	600.0
sparse-high(6)	**6**	**0.18**	**534.49**	1	2.29	554.04
dense-low(6)	**6**	**1.45**	**585.2**	0	4.67	600.0
dense-high(15)	**15**	**0.13**	**474.61**	4	2.44	516.34

RINS depende de la existencia de una solución factible, ya que si no existe al menos una no puede ejecutarse. Al desactivar las heurísticas golosas presentadas en la sección 4.6.2, son las heurísticas de redondeo las que aportan soluciones factibles de una forma rápida aunque de menor calidad, al menos en el nodo raíz. En la tabla 6.13 (completa en A.8) presentamos los resultados al desactivar las heurísticas golosas. Los resultados obtenidos son bastante similares con ambas configuraciones, lo que significa que las heurísticas golosas no siempre "guían" a RINS hacia las mejores soluciones.

Las mejores soluciones enteras son, en su gran mayoría, encontradas por RINS. En la tabla 6.14 incluimos el porcentaje de veces que cada heurística encuentra la mejor solución que se devuelve como salida. (Las instancias son las mismas de la tabla 6.12). La diferencia entre las heurísticas golosas es el conjunto de columnas que se uiliza como conjunto de ciclos candidatos. En el primer caso se utilizan todas las columnas, mientras que en el

Tab. 6.13: Heurísticas primales (greedy).

	Todas			Sin greedy		
	Mejor	Gap	Tiempo	Mejor	Gap	Tiempo
Instancias						
sparse-low(13)	12	2.24	561.81	12	**2.22**	**561.79**
sparse-high(13)	7	**0.34**	500.58	**9**	0.36	**497.47**
dense-low(13)	**12**	**1.5**	568.43	10	1.61	**568.24**
dense-high(13)	10	**0.19**	559.37	10	0.2	**558.39**

segundo solo las variables con valor mayor a cero en la relajación lineal. Se observa que las heurísticas golosas implementadas encuentran soluciones mejores que las de redondeo en el nodo raíz, que luego son mejoradas por RINS. En los demás nodos es RINS la heurística que suele encontrar soluciones enteras con menor costo, aunque su ejecución se limita a una profundida múltiplo de 5. Observamos que a partir las primeras ejecuciones de RINS, las heurísticas golosas 4.6.2 no logran obtener mejores soluciones enteras. Por este motivo, la ejecución de las heurísticas golosas se limita hasta profundidad 5 en caso de usarse con RINS. En algunos casos, cuando son pocas las variables con valor fraccionario, redondear alcanza para mejorar la solución primal. Incluimos como ejemplo la salida de la ejecución de una instancia en el apéndice B junto a las referencias pertinentes.

Tab. 6.14: Heurísticas primales.

	Todas	Sin greedy	Sin RINS
Heurística			
RINS	94 %	94 %	0 %
Greedy 1	0 %	0 %	57 %
Greedy 2	0 %	0 %	25 %
Redondeo	6 %	6 %	18 %

6.10. Problema de *pricing*

6.10.1. Cantidad de columnas por iteración

Tab. 6.15: Resultados obtenidos según cantidad de columnas agregadas por iteración.

	Hasta 5 por iteración				1 por iteración			
	Mejor	Gap	Tiempo	NP	Mejor	Gap	Tiempo	NP
Instancias								
VZ_US_PIP(6)	6	0.0	50.75	2292	6	0.0	**49.03**	2068
sparse-high(12)	**10**	**0.11**	**450.04**	5210	8	0.2	456.94	3993
dense-high(12)	**12**	**0.06**	**443.27**	3425	6	0.13	514.74	3308

El modo más simple de acelerar la finalización de la generación de columnas es agregando más de una columna por iteración de *pricing*. Comparamos los resultados obtenidos si se agrega una columna por cada iteración respecto a un máximo de cinco columnas por iteración en la tabla 6.15 (completa en A.9). La columna **NP** muestra la cantidad total de nodos procesados del árbol de búsqueda.

Lo esperable es encontrar mejores resultados al agregar más de una columna por vez debido a que cada ejecución de *pricing* es muy costosa, si bien es posible que no todas las columnas generadas terminen entrando a la base. Vemos que esto en general ocurre, en particular para las instancias más grandes, en las que se alcanza el tiempo máximo de ejecucion y queda claro que se procesan más nodos para el mismo límite de tiempo.

6.10.2. Heurísticas

Implementamos dos heurísticas para resolver el problema de pricing. La primera de ellas (sección 5.5.3) a la que llamaremos heurística 1 o de inserciones, fue desarrollada con el objetivo resolver las instancias cuyas soluciones óptimas hacen uso de pocos coeficientes positivos provenientes de variables duales del problema maestro. La segunda heurística (sección 5.5.4) será nombrada heurística 2 o de Steiner TSP, y su objetivo es poder encontrar soluciones óptimas en los casos que éstas hagan uso de todos o casi todos los coeficientes.

Para evaluar su rendimiento individualemente guardamos las instancias de *pricing* de algunas ejecuciones del algoritmo para SCA. En la configuración por defecto solo se resuelve de forma exacta y por lo tanto contamos con el óptimo de cada instancia de *pricing*. Se utilizaron 5 de las instancias más grandes de las generadas para SCA y para contar la mayor cantidad posible de instancias de *pricing*, se estableció la configuración de la sección 6.10.1 en la que se agrega una sola columna por iteración de *pricing*. Se guardó también el valor de la solución óptima para poder comparar los costos reducidos obtenidos. Con cada instancia elegida se obtuvo aproximadamente 200 instancias factibles (es decir que existe una columna con costo reducido negativo) de *pricing* sin coeficientes correspondientes a planos de corte. Esto quiere decir que solo se tuvieron en cuenta valores positivos de los costos c y las duales π del problema maestro en esta primera evaluación.

Para ambas heurísticas se estableció el tamaño de la RCL variable de 2 a 5 y se ejecutaron 5 rondas con cada valor del parámetro. Estos parámetros fueron evaluados individualmente con las aproximadamente mil instancias generadas y no se observó ningua ventaja de elegir algún valor particular. Para la heurística 2 basada en Steiner TSP, se estableció en 3 el parámetro que indica la cantidad máxima de vértices no significativos que quedan fuera del ciclo. En este caso la experimentación mostró que incrementar este parámetro hasta 4 no aporta un beneficio significativo respecto al incremento del tiempo de ejecución. El parámetro de cantidad de iteraciones fue elegido de forma tal que el tiempo de ejecución sea acorde al tamaño de las instancias resueltas y significativamente menor que el tiempo de ejecución del algoritmo exacto para *pricing*.

En la tabla 6.16 se presenta el porcentaje de veces que ambas heurísticas encuentran alguna solución factible, juntas o separadas, según el número de iteración expresado como porcentaje del total. Esto combina los resultados de las 5 instancias de SCA. A medida que avanza el número de iteración, las instancias de *pricing* resultan más difíciles de resolver. La heurística 2 en particular no tiene un 100 % de éxito incluso en las primeras iteraciones. Un motivo de esto es que las reparaciones finales del algoritmo no son suficientes. Otro motivo más importante es que algunas instancias de *pricing* pueden no tener ciclos “grandes”

Tab. 6.16: Porcentaje de éxito de las heurísticas para encontrar columnas con costo reducido negativo.

Iteración (%)	Heurística 1	Heurística 2	Juntas
(0, 5]	100.0	98.0	100.0
(5, 10]	100.0	88.0	100.0
(10, 25]	100.0	82.0	100.0
(25, 50]	100.0	74.0	100.0
(50, 75]	88.0	60.8	92.4
(75, 100]	49.8	49.3	72.7

Tab. 6.17: Calidad de soluciones encontradas por las heurísticas como porcentaje respecto al óptimo.

Iteración (%)	Heurística 1	Heurística 2	Juntas
(0, 5]	79.8	88.9	91.1
(5, 10]	67.1	86.2	84.2
(10, 25]	64.3	82.4	78.6
(25, 50]	56.4	71.4	70.4
(50, 75]	38.2	63.7	57.4
(75, 100]	22.8	50.0	41.0

correspondientes a columnas de costo reducido negativo.

Si bien la heurística 2 tiene menor probabilidad de éxito por los motivos mencionados previamente, las soluciones encontradas tienen mejor calidad como puede verse en la tabla 6.17. El número reportado corresponde al promedio de $100 \times \frac{\text{costo red. de heurística}}{\text{costo red. óptimo}}$ tomando en cuenta solamente los casos en que la heurística correspondiente encontró una solución. Notar por ejemplo en el rango de iteraciones (50, 75] la heurística 2 tiene una tasa de éxito mucho menor pero las soluciones encontradas en esos casos de éxito por lo general son mejores. Nuevamente se observa que a medida que avanzan las iteraciones las instancias de *pricing* son más difíciles de resolver.

Para completar la evaluación individual de las heurísticas utilizamos las instancias de *pricing* obtenidas con planos de corte incorporados, es decir, con valores positivos de μ y ρ. En este caso, la cantidad de iteraciones de generación de columnas fue muy diferente entre las distintas instancias del problema maestro, siendo de solo 10 en un caso y hasta 100 en otro. Por lo tanto, presentamos todos los resultados en la tabla 6.18. La columnas tienen, respectivamente, el porcentaje de éxito de encontrar alguna solución factible y la calidad de la solución respecto al óptimo (solo para los casos de éxito).

Incorporamos las heurísticas como parte del algoritmo de generación de columnas, de forma que se ejecutan ambas en cada iteración de *pricing*, y si no se logra obtener una columna se ejecuta el algoritmo exacto. Para verificar su rendimiento solo evaluamos la ejecución en el nodo raíz, ya que la implementación de las heurísticas toma en cuenta solamente coeficientes correspondientes a variables duales de restricciones del problema y

Tab. 6.18: Resultados de las heurísticas de *pricing* para instancias con planos de corte en el problema maestro

	Éxito	Calidad
Heurística 1	59.3	39.8
Heurística 2	64.5	46.2
Juntas	81.4	47.7

los planos de corte usados en la configuración por defecto. Por otro lado, las heurísticas desarrolladas tienen un mejor rendimiento cuanto más alejado se está de resolver la relajación lineal, lo que ocurre al comenzar el procesamiento del nodo raíz. En cada subnodo luego del primer *branch*, la cantidad de iteraciones de *pricing* es significativamente menor.

Agregar muchas columnas irrelevantes para la resolución de la relajación lineal podría perjudicar el rendimiento del algoritmo, demorando la finalización de la generación de columnas. Además, ejecutar la heurística a partir de un cierto punto de la generación de columnas no aporta ningún beneficio. Por este motivo, se estableció un límite de ejecuciones fallidas, es decir, que luego de una cantidad determinada de iteraciones de *pricing* no necesariamente consecutivas en las cuales las heurísticas fallan en encontrar soluciones, las heurísticas no vuelven a ejecutarse y solo se resuelve de forma exacta. Este límite se restaura al momento de incorporar planos de corte.

Los experimentos los realizamos con las instancias generadas de más de 45 vértices, para las cuales el tiempo de ejecución promedio del algoritmo exacto para *pricing* supera los 30 segundos. Es importante recordar que cuando las instancias de *pricing* son más difíciles (próximo resolver el problema maestro) este tiempo aumenta. Con el grupo de instancias VZ_PIP_US (redes reales) incorporar las heurísticas *pricing* eleva ligeramente el tiempo de finalización de la generación de columnas ya que muchas de las columnas generadas no son relevantes para la resolución.

Vemos en las tablas 6.19 y 6.20 los resultados obtenidos con los planos de corte desactivados y activados respectivamente. Se compara el tiempo que demora en finalizar el proceso de generación de columnas con y sin las heurísticas. En las tablas se incluye además la siguiente información.

PI Iteraciones totales de *pricing*.

HI Iteraciones exitosas de heurísticas de *pricing*.

PC Columnas generadas en total, tanto por el algoritmo exacto como por las heurísticas.

PH Columnas generadas por heurísticas de *pricing*.

Las heurísticas primales fueron desactivadas para que no influyan en el tiempo de ejecución. Además se estableció el límite de nodos a procesar en uno para que ejecute solo el nodo raíz. Por cada iteración se incorporan a lo sumo 5 columnas con el mismo criterio explicado para la configuración por defecto.

El tiempo de cada grupo de instancias se reduce entre 17 % y 30 % sin coeficientes provenientes de planos de corte y entre 13 % y 25 % con los coeficientes ρ y μ. Los mejores resultados se obtienen con el grupo de instancias **dem** que consisten en grafos ralos. No se observa mucha diferencia entre los resultados de los otros grupos, aunque el que destaca como más difícil es el grupo “grafo ralo - demanda alta”.

Tab. 6.19: Heurísticas para pricing (sin cuts).

	Solo exacto					Con heurísticas				
Instancias	Tiempo	PI	HI	PC	HC	Tiempo	PI	HI	PC	HC
sparse-low(8)	62.36	1583	0	4766	0	**47.08**	1795	1110	4852	2407
sparse-high(8)	52.97	1351	0	4043	0	**44.01**	1636	990	4310	2069
dense-low(8)	68.14	1808	0	5348	0	**49.16**	2014	1313	5837	3304
dense-high(8)	97.29	1554	0	4654	0	**74.47**	1821	1213	5201	2955
dem(8)	92.33	697	0	2960	0	**65.34**	1601	1241	3958	2392

Tab. 6.20: Heurísticas para pricing (con cuts).

	Solo exacto					Con heurísticas				
Instancias	Tiempo	PI	HI	PC	HC	Tiempo	PI	HI	PC	HC
sparse-low(8)	143.65	2255	0	7398	0	**121.49**	2590	1339	7441	2720
sparse-high(8)	75.19	1497	0	4556	0	**65.28**	1793	1029	4765	2113
dense-low(8)	153.12	2506	0	8043	0	**116.64**	2870	1571	8541	3727
dense-high(8)	156.8	1711	0	5240	0	**126.67**	2018	1284	5708	3038
dem(8)	120.53	817	0	3433	0	**91.07**	1794	1338	4435	2508

Se puede observar que muchas iteraciones de heurísticas son exitosas aunque el tiempo de ejecución no se reduce proporcionalmente. Un resultado positivo es que la utilización de heurísticas no incrementa demasiado la cantidad de columnas incorporadas.

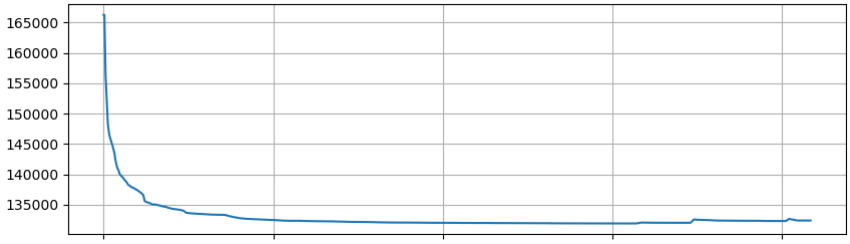
Los resultados completos se encuentran en las tablas A.10 y A.11. En la segunda, correspondiente a planos de corte incluidos en el problema maestro, se pueden ver tres instancias en las que la incorporación de heurísticas resultó perjudicial para el algoritmo.

En la figura 6.1 se presenta el valor de la reajación lineal respecto al tiempo transcurrido durante la resolución de la relajación lineal en el nodo raíz correspondiente a la instancia **dem-60**. Los leves incrementos del valor poco antes de la finalización se deben a la introducción de planos de corte.

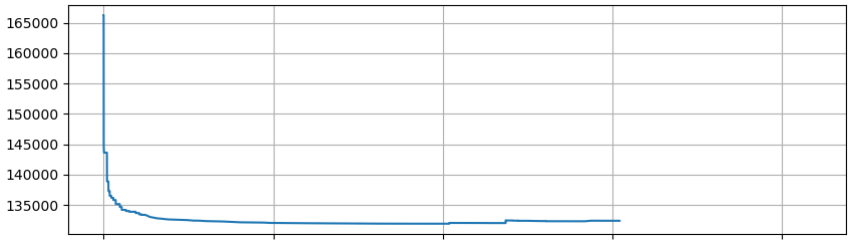
En general, puede decirse que la incorporación de heurísticas permite reducir el tiempo de la etapa inicial de la generación de columnas, pero es claro que aún queda trabajo para mejorar el rendimiento con las instancias cercanas a la finalización del proceso. Este también es el caso de los subnodos del árbol de búsqueda, donde además existen algunos coeficientes negativos provenientes de las variables duales del problema maestro correspondientes a restricciones de *branching*.

6.11. Discusión

En este capítulo presentamos los detalles de la experimentación en cuanto a implementación, entorno, conjuntos de instancias y pruebas realizadas. Comparamos nuestro algoritmo con trabajos previos, para lo cual utilizamos las instancias de (Pecorari, 2016) y los resultados presentados en ese trabajo, que incluye algoritmos propios y de la literatura.



(a) *Pricing* exacto.



(b) *Pricing* heurístico y exacto.

Fig. 6.1: Convergencia: valor de la relajación lineal respecto al tiempo de ejecución (la escala en el eje correspondiente al tiempo es la misma)

Logramos resolver de forma óptima las 4 instancias de (Pecorari, 2016) que no habían sido resueltas. El algoritmo desarrollado en este trabajo resulta eficiente para redes aun más grandes, como puede verse en la tabla 6.21 en las que se presentan resultados con 2 horas de tiempo máximo. Para este caso utilizamos la configuración de parámetros que dio mejores resultados, es decir la regla de *branching* con prioridad en vértice, e incorporamos heurísticas de pricing y los planos de corte para pares de ejes.

Aplicar *branching* en las variables de la formulación original resultó efectivo para lograr obtener soluciones de calidad más allá del nodo raíz. Tal como se ve en la literatura en problemas similares, establecer prioridad en vértices es mejor que en ejes. Además observamos que no fue necesario aplicar *branching* en más de un par de ejes ni en variables del problema maestro, aunque implementamos esos casos para garantizar que el algoritmo sea exacto. Esto significa además que las modificaciones realizadas en la práctica al problema de *pricing* fueron en la mayoría de los casos muy sencillas.

Evaluamos incorporar la posibilidad de hacer podas o *branching* prematuramente incorporando cotas para la cantidad de ciclos en la solución como restricciones de *branching*, para poder calcular cotas duales. Sin embargo, no se logró mejorar los tiempos de ejecución del algoritmo con esta característica.

Aunque la relajación lineal es muy ajustada debido a la descomposición, los planos de corte incorporados resultaron muy beneficiosos. Las modificaciones requeridas en el problema de *pricing* también fueron simples.

La heurística primal basada en RINS resulta importante para tener *gaps* pequeños desde el nodo raíz a pesar de su tiempo de ejecución. Desarrollamos otras heurísticas golosas que si bien obtienen buenas soluciones factibles, no resultaron necesarias para combinarlas con RINS en todos los casos.

Las heurísticas de *pricing* fueron útiles para resolver rápidamente las primeras itera-

Tab. 6.21: Resultados con 2 horas de tiempo límite.

Instancia	Objetivo	BPC Gap	Tiempo
dem56	123030	0.44	7200.0
dem57	129000	0.66	7200.0
dem58	135290	0.51	7200.0
dem59	134960	0.00	1147.74
dem60	133140	0.25	7200.0
dense-high-48	199930	0.00	1311.67
dense-high-49	193250	0.00	3194.22
dense-high-50	207600	0.23	7200.0
dense-high-51	205780	0.98	7200.0
dense-high-52	227560	0.01	7200.0
dense-low-48	47500	0.00	1388.72
dense-low-49	47600	0.48	7200.0
dense-low-50	47770	1.45	7200.0
dense-low-51	52820	0.22	7200.0
dense-low-52	54700	0.00	3111.26
sparse-high-48	260250	1.23	7200.0
sparse-high-49	269080	0.00	1215.82
sparse-high-50	272960	0.02	7200.0
sparse-high-51	271050	0.00	6404.71
sparse-high-52	282990	0.00	1705.27
sparse-low-48	61730	0.86	7200.0
sparse-low-49	66080	0.03	7200.0
sparse-low-50	59500	0.80	7200.0
sparse-low-51	60830	0.55	7200.0
sparse-low-52	66570	1.72	7200.0

ciones del proceso de generación de columnas en el nodo raíz. Sin embargo, las instancias de *pricing* más difíciles que son las últimas de la generación de columnas requieren un enfoque distinto.

Capítulo 7

CONCLUSIONES Y TRABAJO FUTURO

7.1. Conclusiones

En este trabajo abordamos el problema de Asignación de Capacidad de Reserva (SCA) con p-ciclos, el cual fue definido en el contexto de diseño de redes de telecomunicaciones supervivientes. Mostramos que el problema es $\mathcal{NP}$-hard y planteamos la descomposición de Dantzig-Wolfe para una formulación propuesta en este trabajo con restricciones de eliminación de *subtours* generalizadas. Mostramos que el problema de *pricing* también es $\mathcal{NP}$-hard pero con una estructura muy similar a otros problemas encontrados en la literatura que fueron resueltos exitosamente. Con esto desarrollamos un algoritmo *branch-and-price-and-cut* para resolver el problema abordado.

El enfoque utilizado para resolver el problema resultó muy superador al comparar con resultados obtenidos por trabajos previos. Las formulaciones compactas propuestas para SCA tienen el inconveniente de la simetría que representa una gran limitación para resolver instancias de tamaño mayor a los 20 vértices. Los algoritmos heurísticos que construyen una solución agregando ciclos de forma iterativa tampoco resultan del todo efectivos. Por el contrario, al utilizar generación de columnas no observamos una diferencia significativa en el tiempo necesario para resolver instancias con pocos o muchos ciclos (demandas bajas o demandas altas).

Las instancias VZ_US_PIP, correspondientes a redes de comunicaciones reales, fueron resueltas de forma óptima y con muy poco tiempo de ejecución. Como fue señalado en (Pecorari, 2016) y por lo visto en la literatura específica de telecomunicaciones, esto tiene interés práctico por el tamaño y topología de esas redes. Además resolvimos instancias generadas para la experimentación de tamaño significativamente mayor a las redes reales, obteniendo soluciones de muy buena calidad (*gap*) y en algunos casos de forma óptima.

Para mejorar el rendimiento del algoritmo propuesto observamos que es preciso resolver de forma más eficiente el problema de *pricing*. Las instancias derivadas de los valores de las variables duales del problema maestro son muy particulares y esto podría ser aprovechado para reducir el tiempo total del algoritmo *branch-and-price-and-cut*. Relacionado con esto, la lenta convergencia del proceso de generación de columnas también es un tema pendiente a resolver para mejorar este algoritmo.

Logramos incorporar desigualdades válidas relacionadas con la característica de los dos canales protegidos para las cuerdas de los ciclos, lo que a su vez fue también la característica innovadora desde el punto de vista tecnológico cuando fue introducido el problema en el contexto de diseño de redes supervivientes. Estas desigualdades dieron resultados muy positivos, además de que no existieron desventajas en cuanto a mantener la estructura del problema de *pricing*. Esto último aplica también al *branching*, ya que por los resultados obtenidos en la experimentación se logró evitar el desbalance en el árbol de búsqueda manteniendo el problema de *pricing* sin modificaciones significativas.

7.2. Trabajo futuro

El problema de Asignación de Capacidad de Reserva (SCA) con p-ciclos es la base a partir de la cual se definieron otros problemas de diseño de redes de telecomunicaciones con p-ciclos. Además de abordar las cuestiones que no fueron del todo resueltas en este trabajo para SCA, también podemos extender este algoritmo para resolver otros problemas relacionados. Por lo tanto, como trabajo futuro destacamos los siguientes puntos.

- Intensificar el trabajo en resolver el problema de *pricing* de forma más eficiente ya que es la la etapa del algoritmo que implica la mayor proporción del tiempo de ejecución y es clave para reducir el tiempo de ejecución total. Utilizar solamente heurísticas no parece ser el enfoque ideal, por lo que posiblemente se debería explorar un esquema híbrido que incluya programación lineal.

- Las reglas de branching aplicadas, vistas desde el punto de vista de las variables del problema original, son muy simples ya que se elige una variable según su valor fraccionario en la relajación lineal. Aplicar *strong branching* no sería viable porque para resolver cada relajación lineal hace falta resolver varias veces el problema de *pricing*. Sin embargo, podría hacerse de una forma limitada sin generar nuevas columnas al evaluar cada variable (original) para el *branching*, lo cual es una cuestión de implementación. Dados los tiempos necesarios para procesar cada nodo, se podría aplicar este enfoque hasta profundidades del árbol de búsqueda mayores de las que se usan en algoritmos *branch-and-cut*.

  Junto a esto, se podrían incorporar reglas de *branching* basadas en *pseudocostos*, aunque debería existir también una definición clara en los casos que el *branching* se hace en pares de variables (ejes).

- En este trabajo no abordamos de forma general el inconveniente de la lenta convergencia de la generación de columnas. Para esto sería necesario trabajar en los enfoques mencionados en la sección 4.11 sobre estabilización.

- Respecto a extender el algoritmo a otros problemas de diseño de redes con p-ciclos, la incorporación de límites de longitud de circunferencia o máxima cantidad de vértices en un ciclo es una modificación inmediata que tiene utilidad práctica. El cambio en la formulación serían nuevas restricciones que quedan como parte del problema de *pricing*.

  Lo más interesante sería posiblemente extender al problema JCP en el cual se debe decidir tanto la capacidad de trabajo (demandas de entrada en SCA) como la de reserva para todos los ejes. La asignación de capacidad de trabajo por sí sola puede verse como un problema de flujo, mientras que la asignación de capacidad de reserva es el problema tratado en este trabajo. Existen trabajos en la literatura para comparar resultados, aunque fue cambiada la definición del problema a grafos dirigidos y la protección de cuerdas no tiene el doble valor como en la definición original de (Grover & Stamatelakis, 1998).

Apéndice A

TABLAS DE RESULTADOS

En este apartado se presentan las tablas de resultados completos correspondientes al capítulo 6, para cada instancia utilizada en los experimentos.

Las columnas comunes en cada tabla son

- Objetivo: valor de función objetivo de la mejor solución entera encontrada.
- Gap: $100 \times \dfrac{\text{cota primal} - \text{cota dual}}{\text{cota dual}}$
- Tiempo: duración de la ejecución en segundos.

A continuación se reitera la descripción de las columnas adicionales presentadas en el capítulo 6.

En la tabla A.5 se indica la cantidad de planos de corte incorporados por cada instancia, **OD** para 4.19 y **BR** para 4.24 y 4.25.

En la tabla A.6 se tienen las siguientes columnas adicionales.

INF Cantidad de subnodos no factibles luego de aplicar alguna regla de *branching*.

REP Cantidad de subnodos que resultaron no factibles luego de aplicar alguna regla de *branching*, pero luego se encontró una nueva columna con la cual resultó factible.

En las tablas A.10 y A.11 correspondientes a heurísticas para SCAPP, las columnas son

PI Iteraciones totales de *pricing*.

HI Iteraciones exitosas de heurísticas de *pricing*.

PC Columnas generadas en total, tanto por el algoritmo exacto como por las heurísticas.

PH Columnas generadas por heurísticas de *pricing*.

Tab. A.1: Reglas de branching con prioridad en ejes.

	Solo infactibilidad			Ponderado por costo		
	Objetivo	Gap	Tiempo	Objetivo	Gap	Tiempo
Instancia						
VZ_US_PIP_016	36310	0.00	**0.82**	36310	0.00	1.0
VZ_US_PIP_017	38780	**0.08**	**511.23**	38780	0.14	557.72
VZ_US_PIP_018	58250	**0.27**	**349.99**	58250	0.36	423.99
VZ_US_PIP_019	72050	**0.42**	600.0	72050	0.44	600.0
VZ_US_PIP_020	56170	0.40	506.28	56170	**0.35**	**468.66**
VZ_US_PIP_021	46670	0.20	600.0	46670	**0.13**	600.0
dense-high-35	169720	0.00	455.18	169720	0.00	**99.31**
dense-high-36	178290	0.01	600.0	178290	**0.00**	**47.89**
dense-high-37	180180	0.05	600.0	**180120**	**0.00**	**390.71**
dense-high-38	189780	0.30	600.0	**189390**	**0.08**	600.0
dense-high-39	**187280**	**0.20**	600.0	187410	0.27	600.0
dense-high-40	171300	0.02	600.0	171300	**0.01**	600.0
dense-high-41	171220	0.05	600.0	**171210**	**0.04**	600.0
dense-high-42	189510	**0.06**	600.0	**189500**	0.07	600.0
dense-low-35	41010	0.49	600.0	**40990**	**0.39**	600.0
dense-low-36	**43210**	**1.50**	600.0	43250	1.62	600.0
dense-low-37	43540	**0.10**	600.0	43540	0.19	600.0
dense-low-38	45130	0.07	600.0	**45120**	**0.00**	**488.66**
dense-low-39	44730	**0.22**	600.0	44730	0.24	600.0
dense-low-40	41080	0.73	600.0	**40880**	**0.20**	600.0
dense-low-41	41740	2.59	600.0	**41570**	**2.27**	600.0
dense-low-42	46820	1.75	600.0	**46780**	**1.61**	600.0
sparse-high-35	**179290**	**0.01**	600.0	179300	0.03	600.0
sparse-high-36	192350	0.10	600.0	192350	**0.07**	600.0
sparse-high-37	196940	0.03	600.0	196940	**0.02**	600.0
sparse-high-38	204360	0.00	177.67	204360	0.00	**135.1**
sparse-high-39	207140	0.05	600.0	**207070**	**0.01**	600.0
sparse-high-40	**195720**	**0.23**	600.0	195810	0.31	600.0
sparse-high-41	212630	0.02	600.0	212630	0.02	600.0
sparse-high-42	**207160**	**0.24**	600.0	207390	0.36	600.0
sparse-low-35	43440	0.00	223.54	43440	0.00	**132.59**
sparse-low-36	46140	1.05	600.0	**46090**	**0.91**	600.0
sparse-low-37	47250	0.95	600.0	**47100**	**0.43**	600.0
sparse-low-38	49080	0.50	600.0	**49040**	**0.37**	600.0
sparse-low-39	48330	0.09	600.0	48330	**0.00**	**202.31**
sparse-low-40	48060	2.48	600.0	**47780**	**1.92**	600.0
sparse-low-41	52200	1.85	600.0	**52070**	**1.66**	600.0
sparse-low-42	50530	2.01	600.0	**50360**	**1.67**	600.0

Tab. A.2: Reglas de branching: prioridad en vértice vs prioridad en ejes.

	Primero en vértice			Branch en ejes		
	Objetivo	Gap	Tiempo	Objetivo	Gap	Tiempo
Instancia						
VZ_US_PIP_016	36310	0.00	**0.57**	36310	0.00	0.82
VZ_US_PIP_017	38780	**0.00**	**41.69**	38780	0.08	511.23
VZ_US_PIP_018	58250	**0.00**	**27.24**	58250	0.27	349.99
VZ_US_PIP_019	72050	**0.00**	**16.42**	72050	0.42	600.0
VZ_US_PIP_020	56170	**0.00**	**26.05**	56170	0.40	506.28
VZ_US_PIP_021	46670	**0.00**	**122.08**	46670	0.20	600.0
dense-high-35	169720	0.00	**162.75**	169720	0.00	455.18
dense-high-36	178290	**0.00**	**115.53**	178290	0.01	600.0
dense-high-37	**180120**	**0.00**	**96.31**	180180	0.05	600.0
dense-high-38	**189390**	**0.00**	**252.57**	189780	0.30	600.0
dense-high-39	**187130**	**0.00**	600.0	187280	0.20	600.0
dense-high-40	171300	**0.00**	**108.42**	171300	0.02	600.0
dense-high-41	**171210**	**0.00**	**542.97**	171220	0.05	600.0
dense-high-42	**189450**	**0.01**	600.0	189510	0.06	600.0
dense-low-35	**40930**	**0.02**	600.0	41010	0.49	600.0
dense-low-36	**43190**	**0.89**	600.0	43210	1.50	600.0
dense-low-37	43540	**0.00**	**218.41**	43540	0.10	600.0
dense-low-38	**45120**	**0.00**	**183.33**	45130	0.07	600.0
dense-low-39	44730	**0.03**	600.0	44730	0.22	600.0
dense-low-40	**40870**	**0.00**	**438.71**	41080	0.73	600.0
dense-low-41	**41350**	**1.51**	600.0	41740	2.59	600.0
dense-low-42	**46630**	**1.21**	600.0	46820	1.75	600.0
sparse-high-35	179290	**0.00**	**120.06**	179290	0.01	600.0
sparse-high-36	192350	**0.01**	600.0	192350	0.10	600.0
sparse-high-37	196940	**0.00**	**283.93**	196940	0.03	600.0
sparse-high-38	204360	0.00	**90.23**	204360	0.00	177.67
sparse-high-39	**207070**	**0.00**	**101.96**	207140	0.05	600.0
sparse-high-40	195820	**0.21**	600.0	**195720**	0.23	600.0
sparse-high-41	212630	**0.00**	**182.43**	212630	0.02	600.0
sparse-high-42	207210	**0.21**	600.0	**207160**	0.24	600.0
sparse-low-35	43440	0.00	**78.34**	43440	0.00	223.54
sparse-low-36	**45910**	**0.00**	**566.97**	46140	1.05	600.0
sparse-low-37	**47100**	**0.17**	600.0	47250	0.95	600.0
sparse-low-38	**48980**	**0.00**	**449.02**	49080	0.50	600.0
sparse-low-39	48330	**0.00**	**197.96**	48330	0.09	600.0
sparse-low-40	**47450**	**1.10**	600.0	48060	2.48	600.0
sparse-low-41	**52060**	**1.25**	600.0	52200	1.85	600.0
sparse-low-42	50700	2.38	600.0	**50530**	**2.01**	600.0

Tab. A.3: Branching: Prioridad en vértice. Resultados según branching en ejes como regla secundaria.

	Solo infactibilidad			Ponderado por costo		
	Objetivo	Gap	Tiempo	Objetivo	Gap	Tiempo
Instancia						
VZ_US_PIP_016	36310	0.00	0.57	36310	0.00	**0.54**
VZ_US_PIP_017	38780	0.00	**41.69**	38780	0.00	71.12
VZ_US_PIP_018	58250	0.00	**27.24**	58250	0.00	27.3
VZ_US_PIP_019	72050	0.00	16.42	72050	0.00	**15.79**
VZ_US_PIP_020	56170	0.00	26.05	56170	0.00	**25.69**
VZ_US_PIP_021	46670	0.00	**122.08**	46670	0.00	142.75
dense-high-35	169720	0.00	162.75	169720	0.00	**109.59**
dense-high-36	178290	0.00	115.53	178290	0.00	**96.87**
dense-high-37	180120	0.00	**96.31**	180120	0.00	107.64
dense-high-38	189390	0.00	252.57	189390	0.00	**249.26**
dense-high-39	**187130**	**0.00**	600.0	187140	0.01	600.0
dense-high-40	171300	0.00	108.42	171300	0.00	**89.21**
dense-high-41	171210	0.00	**542.97**	171210	0.00	549.87
dense-high-42	**189450**	**0.01**	600.0	189590	0.09	600.0
dense-low-35	**40930**	**0.02**	600.0	40940	0.04	600.0
dense-low-36	43190	0.89	600.0	43190	0.89	600.0
dense-low-37	43540	0.00	**218.41**	43540	0.00	219.37
dense-low-38	45120	0.00	**183.33**	45120	0.00	194.23
dense-low-39	44730	0.03	600.0	44730	0.03	600.0
dense-low-40	40870	0.00	**438.71**	40870	0.00	441.46
dense-low-41	41350	1.51	600.0	41350	1.51	600.0
dense-low-42	46630	1.21	600.0	46630	1.21	600.0
sparse-high-35	179290	0.00	**120.06**	179290	0.00	121.56
sparse-high-36	192350	0.01	600.0	192350	0.01	600.0
sparse-high-37	196940	0.00	**283.93**	196940	0.00	307.74
sparse-high-38	204360	0.00	90.23	204360	0.00	**46.5**
sparse-high-39	207070	0.00	101.96	207070	0.00	**98.59**
sparse-high-40	195820	0.21	600.0	**195730**	**0.16**	600.0
sparse-high-41	212630	0.00	**182.43**	212630	0.00	182.47
sparse-high-42	207210	0.21	600.0	207210	0.21	600.0
sparse-low-35	43440	0.00	78.34	43440	0.00	**71.89**
sparse-low-36	45910	0.00	**566.97**	45910	0.00	571.67
sparse-low-37	47100	0.17	600.0	47100	0.17	600.0
sparse-low-38	48980	0.00	**449.02**	48980	0.00	450.52
sparse-low-39	48330	0.00	**197.96**	48330	0.00	198.36
sparse-low-40	47450	1.10	600.0	47450	1.10	600.0
sparse-low-41	52060	1.25	600.0	52060	1.25	600.0
sparse-low-42	50700	2.38	600.0	50700	2.38	600.0

Tab. A.4: Branching: con o sin cota de cantidad de ciclos.

	Con early branch			Sin early branch			Regla vértice		
	Objetivo	Gap	Tiempo	Objetivo	Gap	Tiempo	Objetivo	Gap	Tiempo
Instancia									
VZ_US_PIP_016	36310	0.00	0.56	36310	0.00	**0.55**	36310	0.00	0.57
VZ_US_PIP_017	38780	0.00	**30.75**	38780	0.00	31.14	38780	0.00	41.69
VZ_US_PIP_018	58250	0.00	**25.93**	58250	0.00	26.42	58250	0.00	27.24
VZ_US_PIP_019	72050	0.00	16.41	72050	0.00	**16.23**	72050	0.00	16.42
VZ_US_PIP_020	56170	0.00	25.72	56170	0.00	**25.54**	56170	0.00	26.05
VZ_US_PIP_021	46670	0.00	205.14	46670	0.00	224.88	46670	0.00	**122.08**
dense-high-36	178290	0.00	199.87	178290	0.00	199.45	178290	0.00	**115.53**
dense-high-37	180120	0.00	112.89	180120	0.00	150.62	180120	0.00	**96.31**
dense-high-38	189390	0.02	600.0	189390	0.02	600.0	189390	**0.00**	**252.57**
dense-high-39	187160	0.02	600.0	187160	0.02	600.0	**187130**	**0.00**	600.0
dense-high-40	171300	0.00	151.45	171300	0.00	149.97	171300	0.00	**108.42**
dense-high-41	171210	0.00	600.0	171210	0.00	600.0	171210	0.00	**542.97**
dense-high-42	189490	0.05	600.0	189590	0.10	600.0	**189450**	**0.01**	600.0
dense-high-43	**189130**	**0.23**	600.0	189220	0.28	600.0	189350	0.33	600.0
dense-high-44	202510	0.00	520.34	202510	0.00	514.38	202510	0.00	**408.92**
dense-high-45	199450	0.00	600.0	199450	0.00	**583.72**	199450	0.00	591.52
dense-low-36	43210	0.92	600.0	43210	0.92	600.0	**43190**	**0.89**	600.0
dense-low-37	43540	0.00	267.18	43540	0.00	269.52	43540	0.00	**218.41**
dense-low-38	45120	0.00	278.46	45120	0.00	276.54	45120	0.00	**183.33**
dense-low-39	44730	0.16	600.0	44730	0.16	600.0	44730	**0.03**	600.0
dense-low-40	40870	0.00	511.18	40870	0.00	509.2	40870	0.00	**438.71**
dense-low-41	**41310**	**1.32**	600.0	**41310**	**1.32**	600.0	41350	1.51	600.0
dense-low-42	46630	1.27	600.0	46630	1.27	600.0	46630	**1.21**	600.0
dense-low-43	45470	0.74	600.0	45470	0.74	600.0	**45380**	**0.44**	600.0
dense-low-44	49950	2.43	600.0	49950	2.43	600.0	**49590**	**1.56**	600.0
dense-low-45	49090	2.92	600.0	49090	2.92	600.0	**48970**	**2.65**	600.0
sparse-high-36	192350	0.07	600.0	192350	0.07	600.0	192350	**0.01**	600.0
sparse-high-37	196940	0.00	566.81	196940	0.00	469.69	196940	0.00	**283.93**
sparse-high-38	204360	0.00	98.9	204360	0.00	98.67	204360	0.00	**90.23**
sparse-high-39	207070	0.00	201.73	207070	0.00	201.57	207070	0.00	**101.96**
sparse-high-40	**195690**	**0.18**	600.0	195830	0.22	600.0	195820	0.21	600.0
sparse-high-41	212630	0.00	206.95	212630	0.00	205.81	212630	0.00	**182.43**
sparse-high-42	207290	0.26	600.0	207420	0.32	600.0	**207210**	**0.21**	600.0
sparse-high-43	**211500**	**0.14**	600.0	211740	0.25	600.0	211610	0.20	600.0
sparse-high-44	**235970**	**0.37**	600.0	**235970**	**0.37**	600.0	236350	0.51	600.0
sparse-high-45	238080	0.15	600.0	238070	0.15	600.0	**238060**	**0.13**	600.0
sparse-low-36	46010	0.38	600.0	46010	0.39	600.0	**45910**	**0.00**	**566.97**
sparse-low-37	47100	0.27	600.0	47100	0.27	600.0	47100	**0.17**	600.0
sparse-low-38	48980	0.07	600.0	48980	0.08	600.0	48980	**0.00**	**449.02**
sparse-low-39	48330	0.00	103.55	48330	0.00	**103.2**	48330	0.00	197.96
sparse-low-40	47710	1.79	600.0	47710	1.79	600.0	**47450**	**1.10**	600.0
sparse-low-41	52200	1.77	600.0	**51800**	**0.97**	600.0	52060	1.25	600.0
sparse-low-42	**50210**	**1.41**	600.0	**50210**	**1.41**	600.0	50700	2.38	600.0
sparse-low-43	**52230**	**3.08**	600.0	**52230**	**3.08**	600.0	52730	4.05	600.0
sparse-low-44	58760	4.36	600.0	58760	4.36	600.0	**58400**	**3.65**	600.0
sparse-low-45	58940	4.21	600.0	58940	4.21	600.0	**58850**	**3.85**	600.0

Tab. A.5: Resultados obtenidos al desactivar los cortes asociados a ejes.

	Con cortes					Sin cortes				
Instancia	Objetivo	Gap	Tiempo	OD	BR	Objetivo	Gap	Tiempo	OD	BR
VZ_US_PIP_016	36310	0.00	0.56	4	0	36310	0.00	**0.55**	0	0
VZ_US_PIP_017	38780	0.00	**30.75**	4	2	38780	0.00	109.07	0	0
VZ_US_PIP_018	58250	0.00	**25.93**	8	0	58250	0.00	44.18	0	0
VZ_US_PIP_019	72050	0.00	**16.41**	7	1	72050	0.00	31.99	0	0
VZ_US_PIP_020	56170	**0.00**	**25.72**	7	0	56170	0.16	188.89	0	0
VZ_US_PIP_021	46670	**0.00**	**205.14**	11	1	46670	0.23	293.02	0	0
dense-high-35	169720	0.00	**134.64**	4	0	169720	0.00	600.0	0	0
dense-high-36	178290	**0.00**	**199.87**	5	0	178290	0.04	600.0	0	0
dense-high-37	180120	0.00	**112.89**	5	0	180120	0.00	424.22	0	0
dense-high-38	**189390**	**0.02**	600.0	6	0	189430	0.09	600.0	0	0
dense-high-39	187160	0.02	600.0	3	0	**187130**	**0.00**	**480.28**	0	0
dense-high-40	171300	0.00	151.45	7	0	171300	0.00	**132.96**	0	0
dense-high-41	171210	0.00	600.0	7	4	171210	0.00	600.0	0	0
dense-high-42	**189490**	**0.05**	600.0	8	0	189550	0.12	600.0	0	0
dense-high-43	**189130**	**0.23**	600.0	10	0	189240	0.36	600.0	0	0
dense-high-44	**202510**	**0.00**	**520.34**	8	0	202590	0.07	600.0	0	0
dense-high-45	**199450**	**0.00**	600.0	6	0	199540	0.07	600.0	0	0
dense-low-35	**40930**	**0.02**	600.0	12	1	40950	0.10	600.0	0	0
dense-low-36	43210	**0.92**	600.0	12	0	**43190**	1.37	600.0	0	0
dense-low-37	43540	0.00	267.18	12	0	43540	0.00	**266.89**	0	0
dense-low-38	**45120**	**0.00**	**278.46**	8	0	45170	0.27	600.0	0	0
dense-low-39	44730	0.16	600.0	10	0	44730	0.16	600.0	0	0
dense-low-40	**40870**	**0.00**	**511.18**	12	0	41110	0.79	600.0	0	0
dense-low-41	**41310**	**1.32**	600.0	11	0	41680	2.66	600.0	0	0
dense-low-42	**46630**	**1.27**	600.0	14	0	46730	1.66	600.0	0	0
dense-low-43	45470	**0.74**	600.0	14	0	**45410**	0.75	600.0	0	0
dense-low-44	49950	2.43	600.0	14	0	**49740**	**2.33**	600.0	0	0
dense-low-45	49090	2.92	600.0	10	0	**48770**	**2.29**	600.0	0	0
sparse-high-35	179290	**0.00**	**126.08**	5	0	179290	0.02	600.0	0	0
sparse-high-36	**192350**	**0.07**	600.0	5	0	192360	0.11	600.0	0	0
sparse-high-37	196940	**0.00**	**566.81**	3	1	196940	0.01	600.0	0	0
sparse-high-38	204360	0.00	**98.9**	6	1	204360	0.00	138.23	0	0
sparse-high-39	207070	0.00	201.73	2	0	207070	0.00	**134.7**	0	0
sparse-high-40	195690	0.18	600.0	9	0	**195530**	**0.10**	600.0	0	0
sparse-high-41	**212630**	**0.00**	**206.95**	9	0	212650	0.02	600.0	0	0
sparse-high-42	**207290**	**0.26**	600.0	5	0	207500	0.37	600.0	0	0
sparse-high-43	**211500**	**0.14**	600.0	8	0	211640	0.22	600.0	0	0
sparse-high-44	**235970**	**0.37**	600.0	6	0	236460	0.77	600.0	0	0
sparse-high-45	238080	0.15	600.0	3	0	**237940**	0.15	600.0	0	0
sparse-low-35	43440	0.00	79.07	6	0	43440	0.00	**48.61**	0	0
sparse-low-36	**46010**	**0.38**	600.0	8	0	46290	1.55	600.0	0	0
sparse-low-37	**47100**	**0.27**	600.0	9	0	47210	0.64	600.0	0	0
sparse-low-38	**48980**	**0.07**	600.0	8	0	49130	0.57	600.0	0	0
sparse-low-39	48330	0.00	**103.55**	4	0	48330	0.00	580.1	0	0
sparse-low-40	**47710**	**1.79**	600.0	13	0	47900	2.37	600.0	0	0
sparse-low-41	52200	1.77	600.0	5	0	**51930**	**1.33**	600.0	0	0
sparse-low-42	**50210**	**1.41**	600.0	9	0	50620	2.38	600.0	0	0
sparse-low-43	**52230**	**3.08**	600.0	13	0	52740	4.54	600.0	0	0
sparse-low-44	58760	4.36	600.0	5	0	**58650**	4.36	600.0	0	0
sparse-low-45	58940	4.21	600.0	6	0	**58180**	**2.86**	600.0	0	0

Tab. A.6: Resultados obtenidos al desactivar los cortes asociados a vértices (cota inferior).

	Con cortes					Sin cortes				
	Objetivo	Gap	Tiempo	REP	INF	Objetivo	Gap	Tiempo	REP	INF
Instancia										
VZ_US_PIP_016	36310	0.00	**0.56**	0	0	36310	0.00	0.58	0	1
VZ_US_PIP_017	38780	0.00	**30.75**	0	0	38780	0.00	62.23	0	6
VZ_US_PIP_018	58250	0.00	**25.93**	0	0	58250	0.00	26.01	0	1
VZ_US_PIP_019	72050	0.00	**16.41**	0	0	72050	0.00	21.51	1	2
VZ_US_PIP_020	56170	0.00	25.72	0	0	56170	0.00	**25.57**	0	0
VZ_US_PIP_021	46670	0.00	**205.14**	0	0	46670	0.00	212.54	0	0
dense-high-35	169720	0.00	**134.64**	0	0	169720	0.00	147.32	0	4
dense-high-36	178290	0.00	199.87	0	5	178290	0.00	**197.58**	1	3
dense-high-37	180120	0.00	112.89	0	0	180120	0.00	**101.27**	0	2
dense-high-38	189390	0.02	600.0	0	0	189390	0.02	600.0	0	0
dense-high-39	187160	**0.02**	600.0	0	0	**187130**	0.03	600.0	0	15
dense-high-40	171300	0.00	**151.45**	0	0	171300	0.00	160.03	0	1
dense-high-41	171210	0.00	600.0	0	0	171210	0.00	**522.14**	0	3
dense-high-42	**189490**	**0.05**	600.0	0	1	189560	0.09	600.0	0	3
dense-high-43	189130	0.23	600.0	0	0	189130	0.23	600.0	0	0
dense-high-44	202510	0.00	**520.34**	0	0	202510	0.00	600.0	0	1
dense-high-45	199450	0.00	600.0	0	0	199450	0.00	**597.99**	0	0
dense-low-35	**40930**	**0.02**	600.0	0	0	40950	0.17	600.0	0	6
dense-low-36	**43210**	**0.92**	600.0	0	0	43300	1.13	600.0	0	3
dense-low-37	43540	0.00	267.18	8	1	43540	0.00	**253.48**	11	2
dense-low-38	45120	0.00	278.46	0	0	45120	0.00	**236.69**	0	1
dense-low-39	**44730**	**0.16**	600.0	0	0	44830	0.75	600.0	22	9
dense-low-40	**40870**	**0.00**	**511.18**	0	0	41070	0.67	600.0	0	2
dense-low-41	41310	**1.32**	600.0	0	1	41310	1.34	600.0	0	1
dense-low-42	46630	1.27	600.0	0	0	**46590**	**1.18**	600.0	0	0
dense-low-43	45470	**0.74**	600.0	0	0	**45460**	0.77	600.0	0	0
dense-low-44	**49950**	**2.43**	600.0	2	0	50020	2.89	600.0	1	13
dense-low-45	49090	2.92	600.0	6	0	**49000**	**2.72**	600.0	6	0
sparse-high-35	179290	0.00	**126.08**	0	0	179290	0.00	233.51	0	2
sparse-high-36	**192350**	**0.07**	600.0	0	0	192360	0.09	600.0	0	3
sparse-high-37	196940	**0.00**	**566.81**	0	0	196940	0.01	600.0	0	2
sparse-high-38	204360	0.00	98.9	0	0	204360	0.00	**98.68**	0	0
sparse-high-39	207070	0.00	**201.73**	0	0	207070	0.00	212.94	0	2
sparse-high-40	**195690**	**0.18**	600.0	0	0	195730	0.21	600.0	0	3
sparse-high-41	212630	0.00	206.95	0	0	212630	0.00	**194.59**	0	2
sparse-high-42	**207290**	**0.26**	600.0	0	0	207450	0.37	600.0	0	7
sparse-high-43	**211500**	**0.14**	600.0	0	0	212910	0.81	600.0	0	0
sparse-high-44	**235970**	**0.37**	600.0	0	0	236210	0.49	600.0	0	3
sparse-high-45	238080	**0.15**	600.0	0	0	238080	0.17	600.0	0	5
sparse-low-35	43440	0.00	79.07	0	0	43440	0.00	**78.8**	0	0
sparse-low-36	**46010**	**0.38**	600.0	0	1	46150	0.95	600.0	0	3
sparse-low-37	**47100**	**0.27**	600.0	0	0	47140	0.36	600.0	0	0
sparse-low-38	**48980**	**0.07**	600.0	0	0	49020	0.49	600.0	11	9
sparse-low-39	48330	0.00	103.55	0	0	48330	0.00	**103.47**	0	0
sparse-low-40	**47710**	**1.79**	600.0	0	0	48200	3.35	600.0	0	6
sparse-low-41	52200	**1.77**	600.0	0	0	**52100**	1.82	600.0	2	2
sparse-low-42	**50210**	**1.41**	600.0	0	0	50700	2.50	600.0	0	2
sparse-low-43	**52230**	**3.08**	600.0	0	0	52560	3.97	600.0	0	2
sparse-low-44	58760	4.36	600.0	0	0	**58210**	**3.45**	600.0	1	2
sparse-low-45	58940	**4.21**	600.0	1	0	**58520**	5.24	600.0	2	12

Tab. A.7: Heurísticas primales (RINS).

	Todas			Sin RINS		
	Objetivo	Gap	Tiempo	Objetivo	Gap	Tiempo
Instancia						
dense-high-35	169720	0.00	**134.64**	169720	0.00	222.04
dense-high-36	178290	0.00	**199.87**	178290	0.00	561.78
dense-high-37	180120	0.00	**112.89**	180120	0.00	135.96
dense-high-38	**189390**	**0.02**	600.0	189490	0.10	600.0
dense-high-39	**187160**	**0.02**	600.0	187230	0.07	600.0
dense-high-40	171300	0.00	**151.45**	171300	0.00	225.26
dense-high-41	**171210**	**0.00**	600.0	174330	1.85	600.0
dense-high-42	**189490**	**0.05**	600.0	193670	2.28	600.0
dense-high-43	**189130**	**0.23**	600.0	195660	3.69	600.0
dense-high-44	**202510**	**0.00**	**520.34**	213590	5.47	600.0
dense-high-45	**199450**	**0.00**	600.0	205920	3.26	600.0
dense-high-46	**202530**	**0.34**	600.0	208750	3.42	600.0
dense-high-47	**216320**	**0.36**	600.0	227480	5.53	600.0
dense-high-48	**201270**	**0.72**	600.0	212480	6.33	600.0
dense-high-49	**193480**	**0.16**	600.0	202060	4.60	600.0
dense-low-40	**40870**	**0.00**	**511.18**	41940	2.70	600.0
dense-low-41	**41310**	**1.32**	600.0	42860	5.15	600.0
dense-low-42	**46630**	**1.27**	600.0	47210	2.55	600.0
dense-low-43	**45470**	**0.74**	600.0	46400	2.87	600.0
dense-low-44	**49950**	**2.43**	600.0	52540	7.74	600.0
dense-low-45	**49090**	**2.92**	600.0	51050	7.01	600.0
sparse-high-40	**195690**	**0.18**	600.0	199050	1.92	600.0
sparse-high-41	212630	0.00	**206.95**	212630	0.00	324.22
sparse-high-42	**207290**	**0.26**	600.0	212770	2.92	600.0
sparse-high-43	**211500**	**0.14**	600.0	216650	2.58	600.0
sparse-high-44	**235970**	**0.37**	600.0	245990	4.63	600.0
sparse-high-45	**238080**	**0.15**	600.0	241780	1.71	600.0
sparse-low-40	**47710**	**1.79**	600.0	49330	5.25	600.0
sparse-low-41	**52200**	**1.77**	600.0	55170	7.51	600.0
sparse-low-42	**50210**	**1.41**	600.0	52400	5.88	600.0
sparse-low-43	**52230**	**3.08**	600.0	54010	6.60	600.0
sparse-low-44	**58760**	**4.36**	600.0	61710	9.60	600.0
sparse-low-45	**58940**	**4.21**	600.0	62150	9.88	600.0

Tab. A.8: Heurísticas primales (greedy).

	Todas			Sin greedy		
	Objetivo	Gap	Tiempo	Objetivo	Gap	Tiempo
Instancia						
dense-high-38	189390	0.02	600.0	189390	0.02	600.0
dense-high-39	187160	0.02	600.0	187160	0.02	600.0
dense-high-40	171300	0.00	151.45	171300	0.00	**151.29**
dense-high-41	171210	0.00	600.0	171210	0.00	600.0
dense-high-42	189490	0.05	600.0	**189450**	**0.03**	600.0
dense-high-43	**189130**	**0.23**	600.0	189340	0.35	600.0
dense-high-44	202510	0.00	520.34	202510	0.00	**519.48**
dense-high-45	199450	0.00	600.0	199450	0.00	**588.25**
dense-high-46	**202530**	**0.34**	600.0	202870	0.51	600.0
dense-high-47	216320	0.36	600.0	**215990**	**0.20**	600.0
dense-high-48	201270	0.72	600.0	201270	0.72	600.0
dense-high-49	**193480**	**0.16**	600.0	193740	0.28	600.0
dense-high-50	208240	0.57	600.0	**208120**	**0.51**	600.0
dense-low-38	45120	0.00	278.46	45120	0.00	**275.71**
dense-low-39	44730	0.16	600.0	44730	0.16	600.0
dense-low-40	40870	0.00	**511.18**	40870	0.00	511.37
dense-low-41	41310	1.32	600.0	41310	1.32	600.0
dense-low-42	**46630**	**1.27**	600.0	46800	1.65	600.0
dense-low-43	45470	0.74	600.0	**45360**	**0.53**	600.0
dense-low-44	49950	2.43	600.0	49950	2.43	600.0
dense-low-45	49090	2.92	600.0	49090	2.92	600.0
dense-low-46	**50440**	**2.22**	600.0	50500	2.34	600.0
dense-low-47	52460	1.31	600.0	52460	1.31	600.0
dense-low-48	47660	0.41	600.0	47660	0.41	600.0
dense-low-49	**48860**	**3.48**	600.0	49430	4.68	600.0
dense-low-50	48430	3.21	600.0	48430	3.21	600.0
sparse-high-38	204360	0.00	98.9	204360	0.00	**71.76**
sparse-high-39	207070	0.00	201.73	207070	0.00	**188.83**
sparse-high-40	195690	0.18	600.0	**195650**	**0.12**	600.0
sparse-high-41	212630	0.00	206.95	212630	0.00	**206.49**
sparse-high-42	207290	0.26	600.0	**207250**	**0.24**	600.0
sparse-high-43	**211500**	**0.14**	600.0	212010	0.38	600.0
sparse-high-44	235970	0.37	600.0	**235780**	**0.29**	600.0
sparse-high-45	**238080**	**0.15**	600.0	238140	0.17	600.0
sparse-high-46	239900	0.14	600.0	**239610**	**0.02**	600.0
sparse-high-47	**247590**	**0.38**	600.0	248320	0.68	600.0
sparse-high-48	263360	2.48	600.0	**263230**	**2.43**	600.0
sparse-high-49	269270	0.08	600.0	**269190**	**0.05**	600.0
sparse-high-50	**273320**	**0.18**	600.0	273490	0.24	600.0
sparse-low-38	48980	0.07	600.0	48980	0.07	600.0
sparse-low-39	48330	0.00	103.55	48330	0.00	**103.27**
sparse-low-40	47710	1.79	600.0	47710	1.79	600.0
sparse-low-41	52200	1.77	600.0	52200	1.77	600.0
sparse-low-42	50210	1.41	600.0	50210	1.41	600.0
sparse-low-43	52230	3.08	600.0	52230	3.08	600.0
sparse-low-44	58760	4.36	600.0	58760	4.36	600.0
sparse-low-45	**58940**	**4.21**	600.0	59370	4.97	600.0
sparse-low-46	57300	2.03	600.0	57300	2.03	600.0
sparse-low-47	61670	4.33	600.0	61670	4.33	600.0
sparse-low-48	62060	1.73	600.0	62060	1.73	600.0
sparse-low-49	66410	0.78	600.0	66410	0.78	600.0
sparse-low-50	60930	3.61	600.0	**60310**	**2.54**	600.0

Tab. A.9: Resultados obtenidos según cantidad de columnas agregadas por iteración.

	Hasta 5 por iteración				1 por iteración			
	Objetivo	Gap	Tiempo	NP	Objetivo	Gap	Tiempo	NP
Instancia								
VZ_US_PIP_016	36310	0.00	**0.56**	5	36310	0.00	0.83	7
VZ_US_PIP_017	38780	0.00	**30.75**	305	38780	0.00	37.42	365
VZ_US_PIP_018	58250	0.00	**25.93**	293	58250	0.00	34.38	409
VZ_US_PIP_019	72050	0.00	**16.41**	109	72050	0.00	17.32	85
VZ_US_PIP_020	56170	0.00	**25.72**	205	56170	0.00	28.0	195
VZ_US_PIP_021	46670	0.00	205.14	1375	46670	0.00	**176.24**	1007
dense-high-35	169720	0.00	**134.64**	36	169720	0.00	600.0	326
dense-high-36	178290	0.00	**199.87**	210	178290	0.00	274.74	259
dense-high-37	180120	0.00	**112.89**	130	180120	0.00	289.21	282
dense-high-38	189390	0.02	600.0	709	189390	**0.01**	600.0	673
dense-high-39	**187160**	**0.02**	600.0	576	187270	0.09	600.0	468
dense-high-40	171300	0.00	**151.45**	92	171300	0.00	215.41	68
dense-high-41	171210	0.00	600.0	658	171210	0.00	**597.53**	561
dense-high-42	**189490**	**0.05**	600.0	238	189580	0.11	600.0	190
dense-high-43	**189130**	**0.23**	600.0	113	189380	0.38	600.0	86
dense-high-44	**202510**	**0.00**	**520.34**	106	203070	0.28	600.0	32
dense-high-45	**199450**	**0.00**	600.0	369	199530	0.05	600.0	241
dense-high-46	**202530**	**0.34**	600.0	188	203180	0.67	600.0	122
sparse-high-35	179290	0.00	**126.08**	215	179290	0.00	148.0	234
sparse-high-36	192350	0.07	600.0	1132	192350	**0.06**	600.0	994
sparse-high-37	196940	**0.00**	**566.81**	1647	196940	0.01	600.0	1477
sparse-high-38	204360	0.00	98.9	201	204360	0.00	**86.55**	91
sparse-high-39	207070	0.00	201.73	420	207070	0.00	**140.49**	192
sparse-high-40	195690	0.18	600.0	235	**195530**	**0.13**	600.0	161
sparse-high-41	212630	0.00	**206.95**	73	212630	0.00	308.26	89
sparse-high-42	**207290**	**0.26**	600.0	376	207680	0.45	600.0	267
sparse-high-43	**211500**	**0.14**	600.0	78	212940	0.83	600.0	21
sparse-high-44	**235970**	**0.37**	600.0	132	236360	0.54	600.0	57
sparse-high-45	238080	0.15	600.0	220	**238060**	0.15	600.0	124
sparse-high-46	**239900**	**0.14**	600.0	481	240070	0.21	600.0	286

Tab. A.10: Heurísticas para pricing (sin cuts).

	Solo exacto					Con heurísticas				
	Tiempo	PI	HI	PC	HC	Tiempo	PI	HI	PC	HC
Instancia										
dem53	78.86	73	0	331	0	**56.37**	164	137	403	282
dem54	38.93	89	0	351	0	**21.86**	245	209	538	392
dem55	65.18	85	0	348	0	**39.71**	189	151	457	294
dem56	40.12	93	0	367	0	**27.55**	267	216	600	415
dem57	78.07	86	0	374	0	**64.19**	215	177	501	339
dem58	111.05	73	0	308	0	**91.78**	138	90	405	180
dem59	131.08	94	0	406	0	**91.67**	183	121	513	231
dem60	195.38	104	0	475	0	**129.59**	200	140	541	259
dense-high-45	31.92	173	0	500	0	**23.9**	289	218	693	467
dense-high-46	36.79	171	0	488	0	**22.07**	178	127	498	326
dense-high-47	89.64	160	0	572	0	**58.1**	288	204	766	434
dense-high-48	93.92	203	0	615	0	**69.39**	192	115	631	330
dense-high-49	56.21	206	0	598	0	**42.38**	190	116	617	344
dense-high-50	57.05	222	0	643	0	**36.21**	220	156	650	403
dense-high-51	330.8	181	0	579	0	**293.09**	204	116	649	292
dense-high-52	81.97	238	0	659	0	**50.63**	260	161	697	359
dense-low-45	31.39	209	0	551	0	**23.98**	256	155	633	345
dense-low-46	33.76	210	0	595	0	**24.22**	244	166	647	392
dense-low-47	58.7	179	0	587	0	**40.24**	296	202	812	471
dense-low-48	105.62	257	0	773	0	**64.91**	259	182	748	473
dense-low-49	90.22	231	0	690	0	**70.22**	245	162	750	423
dense-low-50	83.39	259	0	783	0	**56.24**	238	166	751	465
dense-low-51	55.97	200	0	587	0	**51.45**	217	120	700	337
dense-low-52	86.1	263	0	782	0	**62.0**	259	160	796	398
sparse-high-45	30.39	124	0	391	0	**24.21**	162	106	407	225
sparse-high-46	28.06	166	0	454	0	**18.5**	197	132	468	269
sparse-high-47	34.6	139	0	454	0	**31.51**	178	113	479	233
sparse-high-48	130.39	209	0	707	0	**111.62**	261	173	704	344
sparse-high-49	47.46	188	0	559	0	**40.24**	219	102	620	224
sparse-high-50	37.44	174	0	461	0	**29.38**	198	117	502	243
sparse-high-51	55.71	174	0	547	0	**55.23**	224	114	646	261
sparse-high-52	59.73	177	0	470	0	**41.4**	197	133	484	270
sparse-low-45	35.53	154	0	476	0	**34.02**	208	136	534	270
sparse-low-46	35.61	188	0	542	0	**22.8**	195	135	522	309
sparse-low-47	51.96	165	0	495	0	**37.28**	185	126	487	269
sparse-low-48	69.4	220	0	667	0	**68.73**	218	100	687	256
sparse-low-49	37.79	208	0	592	0	**25.45**	241	120	577	252
sparse-low-50	62.63	237	0	731	0	**57.7**	304	205	833	440
sparse-low-51	116.39	181	0	560	0	**69.93**	199	143	534	303
sparse-low-52	89.6	230	0	703	0	**60.76**	245	145	678	308

Tab. A.11: Heurísticas para pricing (con cuts).

	Solo exacto					Con heurísticas				
	Tiempo	PI	HI	PC	HC	Tiempo	PI	HI	PC	HC
Instancia										
dem53	119.49	96	0	428	0	**76.7**	189	151	463	299
dem54	45.45	99	0	388	0	**29.09**	262	216	580	401
dem55	76.96	97	0	399	0	**53.91**	214	166	514	310
dem56	51.98	110	0	419	0	**35.62**	279	219	636	421
dem57	120.04	104	0	451	0	**108.69**	257	204	585	375
dem58	128.45	77	0	317	0	**109.93**	146	93	419	183
dem59	170.86	110	0	467	0	**135.31**	208	126	606	236
dem60	251.02	124	0	564	0	**179.34**	239	163	632	283
dense-high-45	37.66	187	0	550	0	**28.37**	309	228	735	479
dense-high-46	48.59	183	0	529	0	**30.9**	190	129	527	328
dense-high-47	130.43	176	0	635	0	**96.18**	303	206	823	436
dense-high-48	230.44	251	0	781	0	**158.01**	268	161	779	384
dense-high-49	**84.99**	225	0	671	0	94.33	219	118	696	346
dense-high-50	69.44	239	0	709	0	**49.39**	243	162	730	410
dense-high-51	555.63	198	0	654	0	**493.78**	216	117	691	294
dense-high-52	97.21	252	0	711	0	**62.38**	270	163	727	361
dense-low-45	47.58	267	0	748	0	**41.44**	346	195	864	407
dense-low-46	75.44	275	0	819	0	**62.48**	338	206	898	446
dense-low-47	146.89	246	0	856	0	**94.04**	362	218	1026	500
dense-low-48	166.07	335	0	1093	0	**154.19**	358	196	1097	503
dense-low-49	255.19	374	0	1288	0	**233.88**	418	210	1350	500
dense-low-50	257.22	353	0	1152	0	**134.36**	330	177	1101	500
dense-low-51	107.64	273	0	854	0	**90.58**	319	160	988	393
dense-low-52	168.96	383	0	1233	0	**122.18**	399	209	1217	478
sparse-high-45	48.37	144	0	438	0	**36.69**	180	109	462	229
sparse-high-46	30.44	173	0	474	0	**22.75**	211	135	502	272
sparse-high-47	41.64	148	0	484	0	**38.0**	193	120	514	240
sparse-high-48	202.27	230	0	795	0	**180.99**	284	179	780	351
sparse-high-49	57.81	203	0	605	0	**49.81**	233	105	651	227
sparse-high-50	57.07	206	0	582	0	**39.9**	224	127	572	255
sparse-high-51	**85.9**	200	0	648	0	86.44	249	118	731	266
sparse-high-52	78.06	193	0	530	0	**67.65**	219	136	553	273
sparse-low-45	**95.03**	242	0	798	0	104.4	326	179	879	324
sparse-low-46	56.77	230	0	692	0	**43.16**	251	155	686	338
sparse-low-47	110.62	229	0	767	0	**89.5**	258	144	746	296
sparse-low-48	151.17	333	0	1133	0	**148.57**	344	134	1129	299
sparse-low-49	69.15	283	0	873	0	**54.02**	328	147	844	290
sparse-low-50	124.69	337	0	1138	0	**114.77**	422	247	1222	500
sparse-low-51	366.88	293	0	994	0	**267.29**	319	167	910	336
sparse-low-52	174.92	308	0	1003	0	**150.24**	342	166	1025	337

Apéndice B

EJEMPLO DE SALIDA

A modo de ejemplo incluimos la salida de una ejecución del algoritmo implementado con SCIP.

El primer símbolo a la izquierda indica que se encontró una nueva solución entera con valor de función objetivo menor a las anteriores y el modo por el cual fue encontrada:

r, R son las heurísticas primales simples de redondeo incluidas en SCIP

es la heurística primal golosa con todas las comlumnas como ciclos candidatos

$ es la heurística primal golosa que considera solo las variables con valor fraccionario

^ es la heurística RINS

La columna `node` es el número de nodo siendo procesado y `left` la cantidad de nodos que quedan por procesar. De ese modo se puede identificar el fin del procesamiento del nodo raíz con el par `node` $= 1$, `left` $= 2$.

Al final se incluye información adicional a la salida de SCIP sobre iteraciones de *pricing* y planos de corte aplicados.

```
************************************************************
INSTANCE NAME: dense-high-41.in
************************************************************
INIT: 16 initial columns computed in : 0.219127 seconds.
presolving:
presolving (1 rounds: 1 fast, 1 medium, 1 exhaustive):
 0 deleted vars, 0 deleted constraints, 0 added constraints, 0 tightened bounds, 0 added holes, 0 changed sides, 0 changed coefficients
 0 implications, 0 cliques
presolved problem has 16 variables (0 bin, 16 int, 0 impl, 0 cont) and 150 constraints
    150 constraints of type <linear>
Presolving Time: 0.00

 time | node  | left  |LP iter|LP it/n| mem |mdpt |frac |vars |cons |cols |rows |cuts |confs|strbr|  dualbound   | primalbound  |  gap
r 0.0s|     1 |     0 |    10 |     - |1408k|   0 |   4 |  16 | 150 |  16 | 150 |   0 |   0 |   0 |      --      | 2.465500e+05 |    Inf
  0.0s|     1 |     0 |    16 |     - |1408k|   0 |   2 |  19 | 150 |  19 | 150 |   0 |   0 |   0 |      --      | 2.465500e+05 |    Inf
r 0.0s|     1 |     0 |    16 |     - |1408k|   0 |   2 |  19 | 150 |  19 | 150 |   0 |   0 |   0 |      --      | 2.187700e+05 |    Inf
  5.5s|     1 |     0 |  1412 |     - |7890k|   0 |  37 | 222 | 150 | 222 | 150 |   0 |   0 |   0 |      --      | 2.187700e+05 |    Inf
 14.5s|     1 |     0 |  1984 |     - |  12M|   0 |  37 | 378 | 150 | 378 | 150 |   0 |   0 |   0 | 1.698357e+05 | 2.187700e+05 |  28.81%
R14.5s|     1 |     0 |  1984 |     - |  12M|   0 |  37 | 378 | 150 | 378 | 150 |   0 |   0 |   0 | 1.698357e+05 | 2.186300e+05 |  28.73%
 14.6s|     1 |     0 |  2081 |     - |  12M|   0 |  41 | 381 | 150 | 381 | 157 |   7 |   0 |   0 | 1.698357e+05 | 2.186300e+05 |  28.73%
 19.2s|     1 |     0 |  2313 |     - |  14M|   0 |  37 | 441 | 150 | 441 | 157 |   7 |   0 |   0 | 1.706687e+05 | 2.186300e+05 |  28.10%
R19.2s|     1 |     0 |  2313 |     - |  14M|   0 |  37 | 441 | 150 | 441 | 157 |   7 |   0 |   0 | 1.706687e+05 | 2.177100e+05 |  27.56%
 19.4s|     1 |     0 |  2384 |     - |  15M|   0 |  40 | 442 | 150 | 442 | 159 |   9 |   0 |   0 | 1.706687e+05 | 2.177100e+05 |  27.56%
 20.6s|     1 |     0 |  2439 |     - |  15M|   0 |  36 | 449 | 150 | 449 | 159 |   9 |   0 |   0 | 1.710833e+05 | 2.177100e+05 |  27.25%
R20.6s|     1 |     0 |  2439 |     - |  15M|   0 |  36 | 449 | 150 | 449 | 159 |   9 |   0 |   0 | 1.710833e+05 | 2.170000e+05 |  26.84%
#20.6s|     1 |     0 |  2439 |     - |  15M|   0 |  36 | 449 | 150 | 449 | 159 |   9 |   0 |   0 | 1.710833e+05 | 1.941400e+05 |  13.48%
$20.6s|     1 |     0 |  2439 |     - |  15M|   0 |  36 | 449 | 150 | 449 | 159 |   9 |   0 |   0 | 1.710833e+05 | 1.883700e+05 |  10.10%
^21.9s|     1 |     0 |  2439 |     - |  15M|   0 |  36 | 449 | 150 | 449 | 159 |   9 |   0 |   0 | 1.710833e+05 | 1.719900e+05 |   0.53%
 time | node  | left  |LP iter|LP it/n| mem |mdpt |frac |vars |cons |cols |rows |cuts |confs|strbr|  dualbound   | primalbound  |  gap
 22.4s|     1 |     0 |  2439 |     - |  15M|   0 |  36 | 449 | 150 | 449 | 159 |   9 |   0 |   0 | 1.710833e+05 | 1.719900e+05 |   0.53%
 22.4s|     1 |     2 |  2439 |     - |  15M|   0 |  36 | 449 | 150 | 449 | 159 |   9 |   0 |   0 | 1.710833e+05 | 1.719900e+05 |   0.53%
^39.2s|     7 |     6 |  3693 | 209.0 |  22M|   5 |  42 | 600 | 150 | 600 | 164 |   9 |   0 |   0 | 1.710833e+05 | 1.714700e+05 |   0.23%
^58.6s|    23 |    20 |  5477 | 138.1 |  26M|   5 |  41 | 674 | 150 | 657 | 164 |   9 |   0 |   0 | 1.711396e+05 | 1.713800e+05 |   0.14%
^77.1s|    33 |    28 |  6496 | 126.8 |  32M|   8 |  37 | 764 | 150 | 641 | 164 |   9 |   0 |   0 | 1.711425e+05 | 1.713400e+05 |   0.12%
^88.9s|    47 |    40 |  7586 | 111.9 |  33M|   9 |  39 | 785 | 150 | 732 | 164 |   9 |   0 |   0 | 1.711450e+05 | 1.713300e+05 |   0.11%
^93.8s|    51 |    44 |  7902 | 109.3 |  35M|   9 |  39 | 798 | 150 | 628 | 164 |   9 |   0 |   0 | 1.711500e+05 | 1.713000e+05 |   0.09%
^ 122s|    79 |    63 |  9546 |  91.1 |  39M|  10 |  42 | 846 | 150 | 649 | 164 |   9 |   0 |   0 | 1.711582e+05 | 1.712700e+05 |   0.07%
^ 131s|    88 |    64 | 10326 |  90.7 |  42M|  10 |  42 | 872 | 150 | 622 | 164 |   9 |   0 |   0 | 1.711600e+05 | 1.712600e+05 |   0.06%
  144s|   100 |    72 | 11264 |  89.1 |  44M|  10 |  39 | 912 | 150 | 559 | 165 |   9 |   0 |   0 | 1.711630e+05 | 1.712600e+05 |   0.06%
  239s|   200 |   124 | 19691 |  86.7 |  71M|  13 |  40 |1163 | 150 | 783 | 166 |   9 |   0 |   0 | 1.711809e+05 | 1.712600e+05 |   0.05%
^ 268s|   241 |   138 | 22306 |  82.8 |  79M|  13 |  46 |1180 | 150 | 802 | 169 |   9 |   0 |   0 | 1.711867e+05 | 1.712400e+05 |   0.03%
^ 283s|   263 |   146 | 23831 |  81.6 |  83M|  13 |  25 |1186 | 150 | 831 | 169 |   9 |   0 |   0 | 1.711873e+05 | 1.712300e+05 |   0.02%
^ 298s|   280 |   149 | 25750 |  83.6 |  87M|  13 |  29 |1215 | 150 | 843 | 169 |   9 |   0 |   0 | 1.711873e+05 | 1.712200e+05 |   0.02%
  311s|   300 |   153 | 27349 |  83.3 |  90M|  14 |  36 |1215 | 150 | 697 | 169 |  10 |   0 |   0 | 1.711915e+05 | 1.712200e+05 |   0.02%
 time | node  | left  |LP iter|LP it/n| mem |mdpt |frac |vars |cons |cols |rows |cuts |confs|strbr|  dualbound   | primalbound  |  gap
^ 334s|   326 |   139 | 29574 |  83.5 |  92M|  16 |  44 |1236 | 150 | 937 | 174 |  10 |   0 |   0 | 1.711933e+05 | 1.712100e+05 |   0.01%
```

```
  398s|   400 |   143 | 35244 |  82.2 | 109M|  16 |  45 |1333 | 150 | 721 | 166 |  10 |   0 |   0 | 1.711965e+05 | 1.712100e+05 |   0.01%
  480s|   500 |   139 | 42897 |  81.1 | 123M|  17 |  41 |1416 | 150 |1089 | 173 |  10 |   0 |   0 | 1.712007e+05 | 1.712100e+05 |   0.01%
  554s|   600 |    91 | 50265 |  79.8 | 136M|  18 |  45 |1490 | 150 | 755 | 170 |  12 |   0 |   0 | 1.712029e+05 | 1.712100e+05 |   0.00%

SCIP Status        : solving was interrupted [time limit reached]
Solving Time (sec) : 600.33
Solving Nodes      : 658
Primal Bound       : +1.71210000000000e+05 (233 solutions)
Dual Bound         : +1.71206421052632e+05
Gap                : 0.00 %
Complementary Information.
Pricer iterations (root): 176
Pricer iterations (non root): 1123
Pricer iterations (TOTAL): 1299
Priced columns (root): 433
Priced columns (non root): 1115
Priced columns (TOTAL): 1548
Infeasibility repairs: 0
Infeasibility proofs: 0
Early branches: 3
Early stops (cutoff): 13
Cuts (odd demand): 7
Cuts (vertex LB): 2
Cuts (branch up): 0
Cuts (branch down): 4
Cuts (branch): 4
Cuts (TOTAL): 13
Branch (cycle bound): 4
Branch (vertex): 367
Branch (edge): 21
Branch (2 edge): 0
Branch (3 edge): 0
Branch (weak): 0
```

Apéndice C

EJEMPLO DE INSTANCIAS

La figura C.1 corresponde a la instancia VZ_US_PIP_14. Se presentan las demandas de entrada para cada eje. Se omiten los costos por simplicidad.

Las figuras C.2 y C.3 corresponden a dos intancias de *pricing* que surgen al resolver VZ_US_PIP_14. Se incluyen los valores de las variables duales π del problema maestro cuyo valor es mayor a cero. En cada una de ellas se muestra el ciclo correspondiente a la solución óptima (menor costo reducido, negativo en ambas).

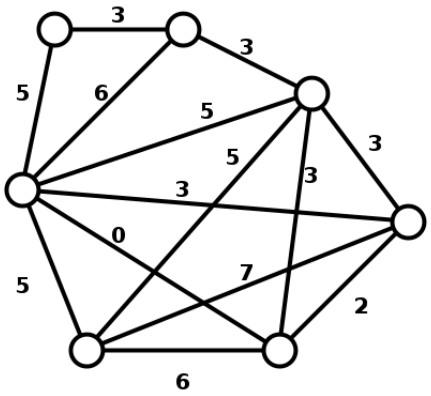


Fig. C.1: Instancia y demandas

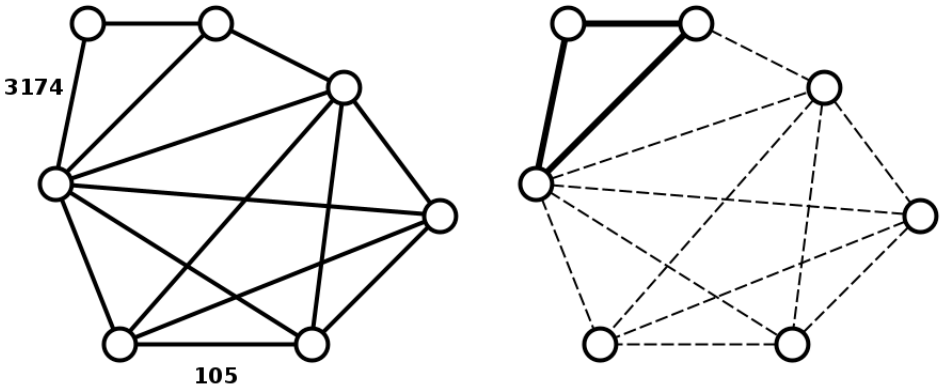


Fig. C.2: Instancia de pricing y su solución

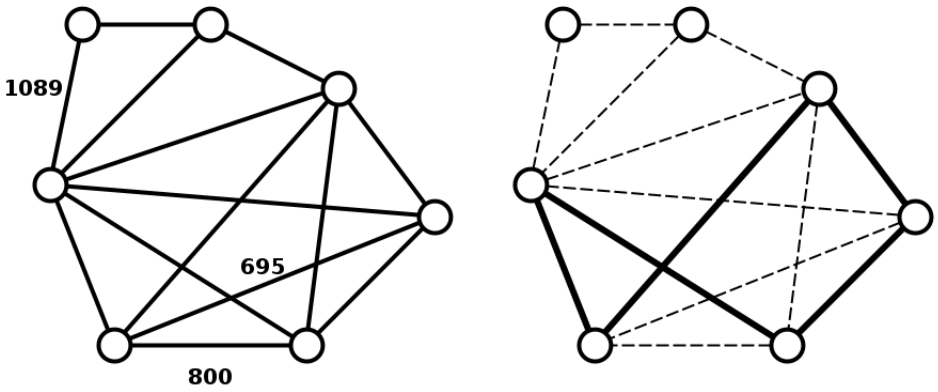


Fig. C.3: Instancia de pricing y su solución

Bibliografía

Aarts, E., & Lenstra, J. K. (2003). *Local Search in Combinatorial Optimization*. Princeton University Press.
URL http://www.jstor.org/stable/j.ctv346t9c

Achterberg, T. (2009). SCIP: solving constraint integer programs. *Mathematical Programming Computation*, *1*, 1–41.

Andersen, E. D. (2001). Certificates of primal or dual infeasibility in linear programming. *Computational Optimization and Applications*, *20*(2), 171–183.

Asthana, R., Singh, Y., & Grover, W. (2010). P-cycles: An overview. *Communications Surveys Tutorials, IEEE*, *12*, 97 – 111.

Atamtürk, A., & Rajan, D. (2008). Partition inequalities for capacitated survivable network design based on directed p-cycles. *Discrete Optimization*, *5*(2), 415–433.
URL https://doi.org/10.1016/j.disopt.2007.08.002

Balas, E. (1989). The prize collecting traveling salesman problem. *Networks*, *19*(6), 621–636.

Ben Amor, H., Desrosiers, J., & Valério de Carvalho, J. M. (2006). Dual-optimal inequalities for stabilized column generation. *Operations Research*, *54*(3), 454–463.
URL http://dx.doi.org/10.1287/opre.1060.0278

Bérubé, J., Gendreau, M., & Potvin, J. (2009). A branch-and-cut algorithm for the undirected prize collecting traveling salesman problem. *Networks*, *54*(1), 56–67.
URL https://doi.org/10.1002/net.20307

CPLEX, I. I. (2017). V12.7: User's manual for cplex. *International Business Machines Corporation*.

Danna, E., Rothberg, E., & Pape, C. L. (2005). Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, *102*(1), 71–90.
URL https://doi.org/10.1007/s10107-004-0518-7

Dantzig, G. B., & Wolfe, P. (1960). Decomposition principle for linear programs. *Operations Research*, *8*(1), 101–111.
URL https://doi.org/10.1287/opre.8.1.101

Delgadillo, R. E. (2013). Modelos y algoritmos para diseño de redes de comunicaciones con requisitos de supervivencia. Tesis de Licenciatura. Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires.

Desaulniers, G., Desrosiers, J., & Spoorendonk, S. (2011). Cutting planes for branch-and-price algorithms. *Networks*, *58*(4), 301–310.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/net.20471

Doucette, J., & Grover, W. D. (2002). Capacity design studies of span-restorable mesh transport networks with shared-risk link group (SRLG) effects. In N. Ghani, & K. M. Sivalingam (Eds.) *OptiComm 2002: Optical Networking and Communications*, vol. 4874, (pp. 25 – 38). International Society for Optics and Photonics, SPIE.
URL https://doi.org/10.1117/12.475308

Doucette, J., He, D., Grover, W. D., & Yang, O. (2003). Algorithmic approaches for efficient enumeration of candidate p-cycles and capacitated p-cycle network design. In *Fourth International Workshop on Design of Reliable Communication Networks*, (pp. 212 – 220).

Drid, H., Brochier, N., Rouzic, E., & Ghani, N. (2012). P-cycle design for mixed-line rate optical networks. In *2012 16th International Conference on Optical Network Design and Modelling (ONDM)*, (pp. 1–4).

du Merle, O., Villeneuve, D., Desrosiers, J., & Hansen, P. (1999). Stabilized column generation. *Discrete Mathematics*, *194*(1), 229 – 237.
URL http://www.sciencedirect.com/science/article/pii/S0012365X98002131

Feo, T., & Resende, M. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, *6*, 109–133.

Fischetti, M., González, J. J. S., & Toth, P. (1998). Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, *10*, 133–148.

Gavish, B., & Graves, S. C. (1978). The Travelling Salesman Problem and Related Problems. *Operations Research Center Working Paper;OR 078-78*.

Gendreau, M., Hertz, A., & Laporte, G. (1992). New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, *40*(6), 1086–1094.
URL https://doi.org/10.1287/opre.40.6.1086

Gendreau, M., Labbé, M., & Laporte, G. (1995). Efficient heuristics for the design of ring networks. *Telecommunication Systems*, *4*(1), 177–188.
URL https://doi.org/10.1007/BF02110085

Gendreau, M., Laporte, G., & Semet, F. (1998). A branch-and-cut algorithm for the undirected selective traveling salesman problem. *Networks*, *32*(4), 263–273.
URL https://doi.org/10.1002/(SICI)1097-0037(199812)32:4<263::AID-NET3>3.0.CO;2-Q

Gilmore, P., & Gomory, R. (1963). A linear programming approach to the cutting stock problem-part ii. *Operations Research*, *11*(6), 863–888.
URL http://www.jstor.org/stable/167827

Grötschel, M., & Padberg, M. W. (1979a). On the symmetric travelling salesman problem i: Inequalities. *Mathematical Programming*, *16*(1), 265–280.
URL https://doi.org/10.1007/BF01582116

Grötschel, M., & Padberg, M. W. (1979b). On the symmetric travelling salesman problem ii: Lifting theorems and facets. *Mathematical Programming*, *16*(1), 281–302.
URL https://doi.org/10.1007/BF01582117

Grover, W., & Doucette, J. (2002). Advances in optical network design with p-cycles: Joint optimization and pre-selection of candidate p-cycles. In *Proceedings of the IEEE-LEOS Summer Topical Meeting on All Optical Networking*, (pp. WA2–49).

Grover, W. D., Doucette, J., Kodian, A., Leung, D., Sack, A., Clouqueur, M., & Shen, G. (2006). Design of survivable networks based on p-cycles. In M. G. Resende, & P. Pardalos (Eds.) *Handbook of Optimization in Telecommunications*. Springer Science.

Grover, W. D., & Stamatelakis, D. (1998). Cycle oriented distributed preconfiguration: ring-like speed with mesh-like capacity for self-planning network restoration. *IEEE ICC '98*, *1*, 537–543.

Grover, W. D., & Stamatelakis, D. (2000). Bridging the Ring-Mesh Dichotomy With P-Cycles. *Proceedings of Design of Reliable Communication Networks (DRCN 2000), Munich, Germany, April 9-12, 2000, Session 5*.

Gschwind, T., & Irnich, S. (2016). Dual inequalities for stabilized column generation revisited. *INFORMS Journal on Computing*, *28*(1), 175–194.
URL `https://doi.org/10.1287/ijoc.2015.0670-`

Interian, R., & Ribeiro, C. C. (2017). A GRASP heuristic using path-relinking and restarts for the Steiner traveling salesman problem. *International Transactions in Operational Research*, *24*(6), 1307–1323.
URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.12419`

Karger, D. R. (1993). Global min-cuts in rnc, and other ramifications of a simple min-out algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, (pp. 21 – 30). Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.
URL `http://dl.acm.org/citation.cfm?id=313559.313605`

Kodian, A., Sack, A., & Grover, W. D. (2004). p-Cycle Network Design with Hop Limits and Circumference Limits. *Proceedings of the First International Conference on Broadband Networks. (BroadNets 2004)*..

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, *7*(1), 48–50.

Laporte, G., & Martín, I. R. (2007). Locating a cycle in a transportation or a telecommunications network. *Networks*, *50*(1), 92–108.
URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/net.20170`

Leifer, A. C., & Rosenwein, M. B. (1994). Strong linear programming relaxations for the orienteering problem. *European Journal of Operational Research*, *73*(3), 517–523.

Liu, C., & Ruan, L. (2004). Finding good candidate cycles for efficient p-cycle network design. *13th international Conference on Computer Communication and Networks(ICCCN 2004)*, (pp. 321 – 326).

Lo, K., Habibi, D., Phung, Q. V., Rassau, A., & Nguyen, H. N. (2007). Qrp02-6: Efficient p-cycles design by heuristic p-cycle selection and refinement for survivable wdm mesh networks. In *IEEE Globecom*, (pp. 1 – 5).

Lübbecke, M., & Desrosiers, J. (2004). Selected topics in column generation. *Operations Research*, *53*, 1007–1023.

Lübbecke, M. E. (2011). Column generation. *Wiley encyclopedia of operations research and management science*.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470400531.eorms0158

Lübbecke, M. E., & Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, *53*(6), 1007–1023.
URL https://doi.org/10.1287/opre.1050.0234

Maher, S. J., Fischer, T., Gally, T., Gamrath, G., Gleixner, A., Gottwald, R. L., Hendel, G., Koch, T., Lübbecke, M. E., Miltenberger, M., Müller, B., Pfetsch, M. E., Puchert, C., Rehfeldt, D., Schenker, S., Schwarz, R., Serrano, F., Shinano, Y., Weninger, D., Witt, J. T., & Witzig, J. (2017). The SCIP Optimization Suite 4.0. Tech. Rep. 17-12, ZIB, Takustr.7, 14195 Berlin.

Mak, V., & Thomadsen, T. (2006). Polyhedral combinatorics of the cardinality constrained quadratic knapsack problem and the quadratic selective travelling salesman problem. *Journal of Combinatorial Optimization*, *11*(4), 421–434.
URL https://doi.org/10.1007/s10878-006-8462-5

Nemhauser, G. L., & Wolsey, L. A. (1988). *Integer and Combinatorial Optimization*. John Wiley & Sons, Ltd.

Panayiotou, T., Ellinas, G., & Antoniades, N. (2016). p-cycle-based protection of multicast connections in metropolitan area optical networks with physical layer impairments constraints. *Optical Switching and Networking*, *19*(P2), 66–77.
URL http://dx.doi.org/10.1016/j.osn.2015.03.007

Pecorari, A. (2016). *Diseño de redes de comunicaciones mediante arquitecturas de p-ciclos y FIPP p-ciclos*. Ph.D. thesis, Universidad de Buenos Aires, Facultad de Ciencias Exactas y Naturales, Departamento de Computación.

Rajan, D., & Atamtürk, A. (2003). Survivable network design: Routing of flows and slacks. In G. Anandalingam, & S. Raghavan (Eds.) *Telecommunications Network Design and Management*, (pp. 65–81). Boston, MA: Springer US.
URL https://doi.org/10.1007/978-1-4757-3762-2_4

Rajan, D., & Atamtürk, A. (2004). A directed cycle-based column-and-cut generation method for capacitated survivable network design. *Networks*, *43*(4), 201–211.
URL https://doi.org/10.1002/net.20004

Reinelt, G. (1991). TSPLIB—A Traveling Salesman Problem Library. *INFORMS Journal on Computing*, *3*(4), 376–384.
URL https://ideas.repec.org/a/inm/orijoc/v3y1991i4p376-384.html

Resende, M. G., & Pardalos, P. (Eds.) (2006). *Handbook of Optimization in Telecommunications*. Springer Science.

Resende, M. G., & Ribeiro, C. C. (2010). Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In M. Gendreau, & J.-Y. Potvin (Eds.) *Handbook of Metaheuristics*, (pp. 283–319). Boston, MA: Springer US.
URL https://doi.org/10.1007/978-1-4419-1665-5_10

Resende, M. G., & Ribeiro, C. C. (Eds.) (2016). *Optimization by GRASP*. Springer Science.

Schupke, D. A. (2004). An ILP for optimal p-cycle selection without candidate cycle enumeration. *8th Conference on Optical Network Design and Modelling ONDM'04*.

Smutnicki, A., & Walkowiak, K. (2012). p-cycle based multicast protection - A new ILP formulation. In *2012 2nd Baltic Congress on Future Internet Communications*, (pp. 268–274).

Thomassen, C. (1997). On the complexity of finding a minimum cycle cover of a graph. *SIAM Journal on Computing*, *26*(3), 675–677.
URL https://doi.org/10.1137/S0097539794267255

Vanderbeck, F. (2000). On Dantzig-Wolfe Decomposition in Integer Programming and ways to Perform Branching in a Branch-and-Price Algorithm. *Operations Research*, *48*(1), 111–128.
URL https://pubsonline.informs.org/doi/abs/10.1287/opre.48.1.111.12453

Vanderbeck, F. (2005). Implementing mixed integer column generation. In G. Desaulniers, J. Desrosiers, & M. M. Solomon (Eds.) *Column Generation*, (pp. 331–358). Boston, MA: Springer US.
URL https://doi.org/10.1007/0-387-25486-2_12

Vanderbeck, F. (2011). Branching in branch-and-price: a generic scheme. *Mathematical Programming*, *130*, 249–294.

Vanderbeck, F., & Savelsbergh, M. W. (2006). A generic view of dantzig–wolfe decomposition in mixed integer programming. *Operations Research Letters*, *34*(3), 296–306.
URL https://www.sciencedirect.com/science/article/pii/S0167637705000659

Vanderbeck, F., & Wolsey, L. (2010). Reformulation and decomposition of integer programs. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, & L. A. Wolsey (Eds.) *50 Years of Integer Programming*, (pp. 431–502). Springer.

Wolsey, L. A. (1988). *Integer Programming*. John Wiley & Sons, Ltd.

Wu, B., Yeung, K., & Ho, P.-H. (2010). ILP formulations for p-Cycle design without candidate cycle enumeration. *IEEE/ACM Transactions on Networking*, *18*(1), 284–295.
URL http://dx.doi.org/10.1109/TNET.2009.2025769

Zhang, H., & Yang, O. (2002). Finding protection cycles in DWDM networks. In *IEEE International Conference on Communications*, vol. 5, (pp. 2756–2760).