Tesis Doctoral

Diseño de redes de comunicaciones mediante arquitecturas de p-ciclos y FIPP p-ciclos

Pecorari, Agustín

2016-10-19

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

Diseño de redes de comunicaciones mediante arquitecturas de p-ciclos y FIPP p-ciclos

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

Agustín Pecorari

Directora de tesis: Dra. Irene Loiseau

Consejera de estudios: Dra. Irene Loiseau

Buenos Aires, 2016

Fecha de defensa: 19 de octubre de 2016.

Diseño de redes de comunicaciones mediante arquitecturas de p-ciclos y FIPP p-ciclos

Las redes de telecomunicaciones se han convertido en infraestructura fundamental para las economías y las sociedades. Debido a las altísimas velocidades de transferencia de datos, la supervivencia de la red es de primordial importancia. En caso de una falla accidental, es imperativo que la red logre una rápida recuperación para minimizar la pérdida de datos. Las redes de telecomunicaciones supervivientes son aquellas que siguen funcionando a pesar de la ocurrencia de fallas. Y esto se logra redirigiendo el tráfico afectado hacia otros sectores de la red en los cuales se instaló capacidad extra con ese fin.

Al diseñar las redes de telecomunicaciones, el objetivo es que se pueda garantizar la protección del tráfico frente a ciertos tipos de fallas con el menor costo posible. Los especialistas desarrollaron inicialmente dos métodos: la protección basada en la restauración de la malla y las topologías basadas en anillos. La tecnología de p-ciclos fue propuesta a finales de la década de 1990 y se convirtió rápidamente en una técnica prometedora debido a que brinda los beneficios combinados de la velocidad de recuperación de la arquitectura de anillo y la eficiencia económica de la arquitectura de malla. Inicialmente se propusieron redes basadas en p-ciclos donde cada ciclo protege la red ante la falla de un vínculo que forma parte del ciclo o de uno que tiene sus dos extremos en éste (cuerda). Años mas tarde se desarrolló el concepto de FIPP p-ciclos (*failure independent path protecting p-cycles*), en el cual los p-ciclos protegen caminos entre dos de sus nodos.

Nuestro trabajo se centra en los problemas de alocación de capacidades de repuesto (*Spare Capacity Allocation Problem*, SCA) para p-ciclos y FIPP p-ciclos. Estos problemas son NP-difíciles. Evaluamos nuestros resultados utilizando topologías de redes reales (de EEUU y Europa) y artificiales (grafos completos de hasta 12 nodos, K_{12}).

Para el primer poblema (SCA) propusimos tres nuevos modelos de programación lineal entera (**PLE**) y entera mixta (**PLEM**) que implementamos sobre CPLEX (**Branch-and-Cut**), dos modelos de programación por restricciones (**CP**) sobre el motor CP de CPLEX, una metaheurística *Greedy Randomized Adaptive Search Procedures* (**GRASP**) con búsqueda local exacta y un algoritmo **Branch-and-Price** exacto. Con este último se obtuvieron excelentes resultados (menos de 1,5 % del óptimo) para las instancias artificiales (K_{12} tiene mas de 59 millones de ciclos posibles). Para las redes reales de hasta 27 nodos obtuvimos resultados de menos de 4,5 % del óptimo.

Para el segundo problema (FIPP) propusimos un nuevo modelo PLEM que implementamos sobre CPLEX (**Branch-and-Cut**), un modelo de CP, una metaheurística GRASP con búsqueda local exacta y un algoritmo *Branch-and-Price* exacto. En este caso, el algoritmo **Branch-and-Cut** del modelo PLEM resultó ser el más eficiente, obteniendo la solución óptima dentro de los 5 minutos para todas nuestras instancias.

Palabras clave: redes de telecomunicaciones supervivientes, p-ciclos, FIPP p-ciclos, *Branch-and-Price*.

P-cycle and FIPP p-cycle
Telecommunications Network Design

Telecommunications networks have become fundamental infrastructure for the economy and societies. Due to high speeds of data transfers, network survivability is of major importance. In case of an accidental failure, a fast recovery is imperative to minimize data loss. A telecommunication network is said survivable if it is still able to provide communication between sites it connects after certain component fails. Survivability is achived redirecting traffic to parts of the network where spare capacity have been installed for that porpouse.

The objective of survivable network design is to guarantee the protection of the traffic on some failure scenarios at minimum cost. Mesh restoration schemes were widely used in the 1970s and early 1980s. Ring based topologies were introduced in the late 80s based on self-healing rings (SHR) networks technology. In the late 1990s the p-cycle networking concept was proposed. This new technology is reported to simultaneously provide the switching speed and simplicity of rings with the much greater efficiency and flexibility for reconfiguration of a mesh network. A single unit capacity p-cycle is a cycle composed of one spare channel on each span it crosses. So a p-cycle provides one protection path for a failed span and it also protects spans that have both end nodes on the cycle but are not themselves on the cycle. In 2005, the FIPP (failure independent path protecting) p-cycle was proposed to protect paths.

Our work centers on the Spare Capacity Allocation Problems (SCA) for p-cycles and FIPP p-cycles. These problems are NP-hard. We evaluated our work on topologies of real networks (USA and Europe) and artificial ones (complete graphs up to 12 nodes, K_{12}).

For the first problem (SCA) we present three new mixed integer programming (**MIP**) models implemented on CPLEX (**Branch-and-Cut**), two constraint programming (**CP**) models implemented on the CPLEX CP engine, one **GRASP** metaheuristics algorithm with exact local search and one exact **Branch-and-Price** algorithm. With the last algorithm we obtained excelents results (less than 1.5 % optimality gap) on artificial networks

with more than 59 millions cycles. For the real networks instances with up to 27 nodes we obtained less than 4.5 % optimality gap.

For the second problem (FIPP) we present one new mixed integer programming (**MIP**) model implemented on CPLEX (**Branch-and-Cut**), one constraint programming (**CP**) implemented on the CPLEX CP engine, one **GRASP** metaheuristics algorithm with exact local search and one exact **Branch-and-Price** algorithm. In this case the **Branch-and-Cut** algorithm based on the MIP model is the most efficient, obtaining the optimal solutions within 5 minutes for all our instancies.

Keywords: survivable networks, p-cycles, FIPP p-cycles, Branch-and-Price.

A mis padres y a mi hija Camila.

Agradecimientos

Quiero agradecer en primer lugar a mi Directora, Irene Loiseau, por su paciencia y el valioso tiempo que me dedicó durante este proceso.

Además, quiero agradecer a los jurados de esta tesis por todo el tiempo dedicado a evaluar el trabajo. En especial a Isabel Méndez Díaz por la detallada revisión de los modelos matemáticos.

Otro agradecimiento especial es para Emanuel Delgadillo, Min Chih Lin, Verónica Becher, Juan José Miranda Bront, Daniel Negrotto, Pablo Factorovich y a todos los integrantes del Grupo de Investigación Operativa, Optimización Combinatoria y Grafos. Gracias por toda la ayuda y por los momentos compartidos en congresos y clases.

Agradezco a Soledad María Moreno por la detallada revisión de la redacción de esta tesis, a Francisco Boerr y Horacio Rojo por su apoyo, al equipo *Global Planning Tools* de *Verizon Business*, en especial a Venugopal Nandagopal y Thomas P. Newcomer por compartir sus conocimientos sobre telecomunicaciones.

Agradezco también a mis padres por haberme apoyado desde siempre con el estudio.

For the Snark's a peculiar creature, that won't
be caught in a commonplace way.
Do all that you know, and try all that you don't:
Not a chance must be wasted today!

Lewis Carroll, *The Hunting of the Snark*

Índice general

Índice de figuras

1. NOTACIÓN

B	Matriz de ciclos fundamentales del grafo G (μ x m).
C	Conjunto de ciclos fundamentales con respecto a un árbol arbitrario.
C_k^n	Coeficiente binomial: es el número de subconjuntos de k elementos escogidos de un conjunto con n elementos.
CS_j	Ciclo j generado.
D	Demanda más alta para una sola arista en todo el grafo G.
E	Conjunto de aristas.
E_r	Conjunto de aristas del camino de la demanda r.
$G = (V, E)$	Grafo no dirigido.
J	Cantidad máxima de ciclos CS_j a incluir en la solución.
K	Conjunto de todos los ciclos del grafo.
L	Número muy grande.
M	Matriz de incidencia del grafo G (n x m).
M^t	Matriz de incidencia transpuesta del grafo G (m x n).
P	Conjunto de ciclos elegibles, indexado por p.
PK	Conjunto de ciclos elegibles (indexado por p) y sus combinaciones de demandas compatibles (indexado por k).
Q	Cantidad máxima de repeticiones de ciclos.
V	Conjunto de nodos.
W	Conjunto de relaciones de demandas, indexado por r.

$\alpha = 1/\|V\|$	Coeficiente para garantizar voltajes compatibles.
Δ	Constante positiva grande (100 000).
λ_{c_p}	Cantidad de aristas del ciclo c_p.
μ	Dimensión del espacio de ciclos (*i.e.* número ciclomático de G).
∇	Constante positiva pequeña (0,0001).
$\partial_{m,n}$	Coeficiente igual a 1 si las relaciones de demandas m y n son «rivales». Esto significa que los caminos definidos para proteger las relaciones de demandas m y n no son mutuamente disjuntos (nodos o aristas).
Ω	Conjunto de ciclos prohibidos.
Ω'	Conjunto de *ciclos-relaciones de demandas* prohibidos.
$\Theta_{c_{pk}}$	Conjunto de relaciones de demandas del *ciclo-relaciones de demandas* c_pk.
π_e^p	Coeficiente igual a 1 si el ciclo p cruza la arista e, 0 si no.
c_k	Costo unitario del ciclo k.
$costo_e$	Costo de la arista e.
$costo^{pk}$	Costo del p-ciclo p (independiente de las demandas k).
d_e	Demanda en la arista e.
d_r	Cantidad de demandas unitarias en el conjunto de relaciones de demanda r.
e	Índice en el conjunto E de aristas del grafo $G = (V, E)$.
$e.origen$	Nodo origen del eje dirigido e.
$e.destino$	Nodo destino del eje dirigido e.
i	Índice sobre el conjunto V donde $i \in \{1, 2, \ldots, n\}$.
j	Índice sobre el conjunto de ciclos a generar, donde $j \in \{1, 2, \ldots, J\}$.
k	Índice sobre el conjunto V donde $k \in \{1, 2, \ldots, n\}$.
l	índice sobre el conjunto C donde $l \in \{1, 2, \ldots, \mu\}$.
$n = \|V\|$	Cantidad de nodos del grafo $G = (V, E)$.
$m = \|E\|$	Cantidad de aristas del grafo G ($\|E\|$).

p_k^e Coeficiente que codifica la relación de protección entre la arista e y el ciclo elegible k. $p_k^e \in \{0, 1, 2\}$. Vale 1 si la protección es *on-cycle* 2 si es una protección *straddle* y 0 en otro caso.

v Índice en el conjunto V de vértices del grafo $G = (V, E)$.

x_r^p Coeficiente igual a 1 si la relación de demanda r está en el p-ciclo p como *on-cycle*, 2 si es una cuerda y 0 en otro caso.

x_r^{pk} Coeficiente igual a 1 si la relación de demanda r está en el p-ciclo p como *on-cycle*, 2 si todo el camino es por aristas que son cuerdas del p-ciclo (*straddles*) y 0 en otro caso.

2. INTRODUCCIÓN

2.1. Introducción

Las redes de telecomunicaciones metropolitanas, nacionales e internacionales basadas en fibra óptica son una de las maravillas del siglo 20 y se han convertido en infraestructura fundamental para las economías y las sociedades actuales y futuras. Al igual que muchas infraestructuras civiles básicas, como el agua, las carreteras, la energía y la salud pública, son casi invisibles para el común de las personas, especialmente cuando trabajan casi a la perfección. Pero surgen graves impactos económicos, personales y sociales si estos sistemas no funcionan, aunque sea temporalmente. Al igual que estas otras infraestructuras básicas, la red de transporte de fibra óptica es ahora de gran importancia para nuestra economía, estilos de vida, educación, entretenimiento, finanzas, etc. Los avances en informática, tecnología inalámbrica, movilidad, multimedia, televisión de alta definición, Internet, todo se detendría si no fuera por las capacidades de la red de transporte subyacente. El público a veces se pregunta; *¿Qué pasa con los teléfonos inalámbricos y celulares? ¿con ellos no necesitamos fibra?*, pero esto sólo se basa en el desconocimiento técnico de que cada llamada de teléfono celular requiere de transporte a través de fibra óptica por enlaces troncales, entre *switches* y estaciones base, para completar las llamadas. Del mismo modo, cada *digital subscriber line* (DSL) y cable módem de alta velocidad es también un usuario de la red troncal de transporte de fibra. Estas tecnologías de «acceso», a la que podemos añadir servidores y cajeros automáticos, son perfectamente conocidos por todos nosotros como usuarios, ya que estos sistemas están a la vista. Pero todos ellos se basan en la red de transporte que opera detrás de escena.[3]

Debido a las altísimas velocidades de transferencia de datos, la supervivencia de la red es de primordial importancia. En caso de una falla accidental, como el corte de un cable, es imperativo que la red logre una rápida recuperación para minimizar la pérdida de datos. Las redes de telecomunicaciones supervivientes son aquellas que siguen funcionando a pesar de fallas (cortes en las líneas de fibra óptica o fallas de los equipos). La supervivencia se logra redirigiendo el tráfico afectado hacia otros sectores de la red en los cuales se instaló capacidad extra con ese fin. Nuestro objetivo es diseñar las redes de forma tal que se pueda garantizar la protección total del tráfico frente a ciertos tipos de fallas, con el menor costo posible.

Para proteger y recuperar las redes de transporte, los especialistas desarrollaron e introdujeron dos métodos: la protección basada en anillos y la restauración de la malla. [30] La protección basada en anillo ofrece un tiempo de recuperación rápido a expensas de una mayor redundancia, mientras que la restauración de malla ofrece una mejor eficiencia a expensas de tiempos de recuperación más lentos. Los p-ciclos fueron propuestos a finales de la década de 1990, por Wayne D. Grover y D. Stamatelakis [3]. Se convirtieron rápidamente en una técnica prometedora para la recuperación de las redes de malla debido a los beneficios combinados de la velocidad de recuperación de los anillos y la eficiencia de las mallas.

2.2. Diseño de redes de telecomunicaciones

El diseño de redes de telecomunicaciones trata sobre la creación de un plan que sirva de orientación sobre cómo debería ser la red. Se debe elegir la estructura de la red, ubicar recursos y configurar parámetros de alto nivel. El administrador de la red puede ser un operador público o una corporación privada, puede tratarse de telefonía fija, móvil o un ISP (*Internet Service Provider*). Recomendamos el libro «Handbook of Optimization in Telecommunications» de Resende *et al.* [64], el cual detalla el estado del arte al año 2005.

La planificación y configuración de la red es considerada desde un punto de vista técnico; actividades como seleccionar segmentos de consumidores, decidir qué servicios ofrecer y la tecnología a utilizar se asumen finalizadas.

El diseño de redes tiene como objetivo acomodar un determinado tráfico o demanda. Para ello es necesario caracterizar esa demanda con un «modelo de tráfico», el cual determina los puntos iniciales y finales de las demandas, máximo *delay* tolerable y su volumen. Este último es medido en unidades que son múltiplos de bits por segundo por canal, por ejemplo: 64 kbits/s, E1/T1, OC-1 (51,84 Mbit/s), OC-3, OC-12, OC-24, OC-48, OC-192 y OC-768 (39.813,12 Mbit/s).

Las redes, en general, están divididas en partes que son tratadas diferentemente. Esto se debe a diversas funcionalidades, propiedades, topologías, geografía, o simplemente para que sean más sencillas de operar. Normalmente, las redes son separadas a su vez por niveles (*tiers*), siendo la más cercana al usuario final la de nivel 1, también llamada «red de acceso», mientras que la de nivel más alto es llamada *backbone* (columna vertebral). El tráfico, en esta última está compuesto por el de muchos usuarios. Es por este motivo que cada nivel debe tener un «modelo de tráfico» apropiado a sus necesidades. [64]

El ruteo puede ser estático, preconfigurado por el administrador de la red, o adaptativo, basado en las condiciones actuales de la red y es implementado por protocolos. Un protocolo es un conjunto de reglas que se implementa en los nodos, cada uno con información de ellos

mismos e intercambian información del estado de los otros nodos del sistema. En base a esos estados, tienen respuestas predefinidas. Un aspecto importante de los protocolos es que tienen una naturaleza distribuida en la cual los nodos tienen un acceso limitado de la infomación de la red. Un ejemplo de estos protocolos es el OSPF (*Open Shortest Path First*) utilizado en redes IP, que calcula el camino más corto, teniendo en cuenta las fallas actuales y la capacidad disponible. [64]

Una red de malla es una topología de red en la que todos los nodos cooperan en la distribución de los datos. [64]

En una red basada en anillos, los nodos están conectados a ciclos y por lo tanto solo transmiten información através de ellos. Estos ofrecen altos niveles de resiliencia a bajo costo, gracias a que es fácil tomar distintos caminos y vincularlos en forma de anillo con poca longitud extra de fibra. Los cables de comunicaciones submarinos son típicamente construidos en pares que funcionan como anillos autoreparables.

La calidad de una red se mide con varios indicadores, muchos de ellos contradictorios entre sí. Por lo tanto, una red puede ser buena para algunas aplicaciones y puede no serlo para otras. Los diseñadores deben equilibrar esos indicadores en función de sus objetivos. [64]

Los principales indicadores son:

- Performance (volumen, *delay*, *jitter* -variaciones en el *delay*-, cobertura, disponibilidad).
- Redundancia, resiliencia, supervivencia.
- Costo.
- Capacidad para adaptarse a los cambios en los patrones de tráfico.

A continuación describiremos las distintas topologías más utilizadas.

2.2.1. Topologías basadas en mallas

Cuando se utiliza una técnica de enrutamiento para diseñar una red de malla, el mensaje se propaga a lo largo de un camino, viajando de nodo a nodo hasta que alcanza el destino. Para asegurar la disponibilidad de todos sus caminos, una red de enrutamiento debe permitir conexiones continuas y la reconfiguración de los caminos rotos, utilizando protocolos de auto-reparación. Una red de malla cuyos nodos están todos conectados entre sí es una red totalmente conectada.

Los protocolos basados en *mallas* tienen múltiples caminos entre cualquier par de nodos fuente y destino. Cuando la topología de la red cambia frecuentemente, estos protocolos son más robustos debido a que generalmente disponen de varios caminos. Además, hacen un muy buen uso de las capacidades de repuesto. Por otro lado, la velocidad de restauración es menor que en otros paradigmas debido a la necesidad de señalización y de toma de decisiones de ruteo más complejas. Para más detalles, ver Skorin-Kapov *et al.* [64], capítulo 20, pág. 538.

2.2.2. Topologías basadas en anillos

Henningson *et al.* [64] explican que, al aplicar los métodos tradicionales para el diseño de redes de telecomunicaciones, se obtienen estructuras tipo árbol. Sin embargo, debido a la creciente importancia de estas redes, su supervivencia es crucial. No es aceptable que partes de la red queden completamente incapaces de comunicarse si un solo vínculo se rompiese. Por lo tanto, las redes de telecomunicaciones deben ser diseñadas cumpliendo ciertos requisitos de supervivencia.

En los sistemas de telecomunicaciones son comunes los anillos auto-reparables o SHR (*self-healing ring*). Estos consisten en dos anillos superpuestos: un anillo trabaja comunicando los nodos y otro anillo superpuesto pero en el sentido inverso provee redundancia. Los estándares tecnológicos utilizados en las redes troncales SDH (*Synchronous Digital*

Hierarchy) y SONET en los EEUU (*Synchronous Optical Networking*) son normalmente configurados como anillos auto-reparables.

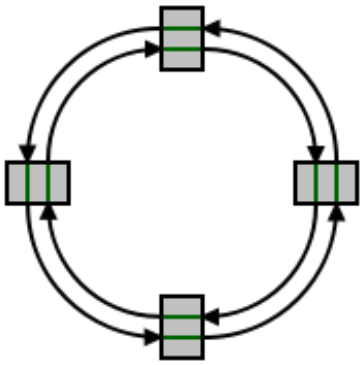

Fig. 2.1: Anillo auto-reparable intacto

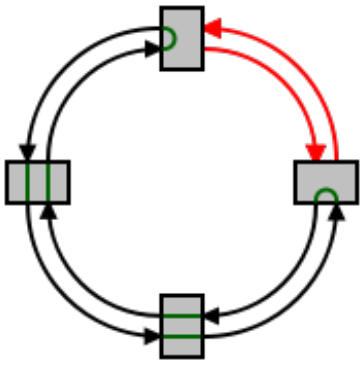

Fig. 2.2: Anillo auto-reparable con una falla

Los anillos SDH pueden ser unidireccionales o bidireccionales. En el caso bidireccional, el tráfico puede ser enviado en cualquiera de los dos sentidos. Así, la mitad de las fibras son utilizadas para enviar el tráfico y la otra mitad son utilizadas solo en caso de falla. Para cada paquete de datos, se debe determinar en qué sentido conviene enviarlo. Los objetivos son minimizar la capacidad a utilizar y balancear la carga. En este caso, el problema de ruteo se debe resolver para cada anillo. [64]

En el caso unidireccional, cada anillo utiliza dos fibras, una para enviar todo el tráfico en una dirección y la otra solo para protección en el sentido contrario. La capacidad requerida

es, en este caso, igual a la suma de toda la demanda para ambas fibras; a esto se lo llama protección dedicada. Esta solución es más cara debido a que requiere más capacidad en todos los segmentos del anillo, pero es más fácil de administrar, ya que no necesita especificar el ruteo. [64]

Cuando se utilizan anillos en una red, se requiere de muchos distintos. Normalmente la estructura física de la red (*i.e.* las fibras y los nodos) viene dada sin los anillos lógicos y la demanda está especificada por pares de nodos. Se trata entonces de determinar qué nodos deben conformar qué anillos, cuáles son las fibras que deben ir en cada uno y cómo rutear el tráfico en estos para minimizar el costo total. Este último incluye el costo que depende de la longitud total de cada anillo, el asociado al ruteo y el del equipamiento en los nodos. [64]

Normalmente, para facilitar la administración de la red, solo es posible cambiar tráfico de un anillo a otro en algunos pocos nodos especiales, llamados nodos de tránsito. Si tenemos una red con dos nodos de tránsito y queremos que sea 2-conexa, debemos pedir que al menos un anillo pase por los dos nodos de tránsito y que todos los anillos estén conectados por lo menos a uno de los nodos de tránsito. En ese caso, si ocurre una falla en un vínculo, todo el tráfico puede ser enviado utilizando el otro camino. [64]

Si hay una falla en alguno de los nodos (exceptuando los nodos de tránsito), todo el tráfico (salvo el tráfico con origen o destino en el nodo de la falla) puede ser enviado de la misma forma descrita en el párrafo anterior. Sin embargo, si hay una falla en alguno de los nodos de tránsito, todo el tráfico que use este nodo es interrumpido. Si agregamos el requisito de 2-conectividad respecto de los nodos, *i.e.* de requerir que para cada par de nodos de la red existan dos caminos disjuntos en el sentido de los nodos, entonces el tráfico puede ser enviado a pesar de una falla en un nodo de tránsito.

La velocidad de restauración ante una falla es mucho mayor en los anillos que en las topologías malladas.

El equipo utilizado para conectar un anillo a un nodo es un OADM (*Optical Add Drop Multiplexer*), el cual es capaz de concentrar la demanda en un flujo mayor y subirlo al anillo. Esto permite que el tráfico principal pase por el anillo y parte del tráfico sea subido (*added*) o bajado (*dropped*) entre el anillo y el nodo.

2.2.2.1. Diseño de redes basadas en anillos mediante generación de ciclos

Supongamos que queremos generar un conjunto de anillos (ciclos de un grafo) y luego utilizar un método de optimización para determinar cuales utilizar y como rutear las demandas. [64]

La cantidad de anillos posibles es enorme para redes grandes, pudiendo llegar hasta mil millones de posibles anillos en redes reales de entre 50-100 nodos. Si el grafo que representa la red es un grafo completo, por ejemplo de 20 nodos (relativamente pequeño), entonces la cantidad de ciclos es 1.745 E +17. Sabemos que la cantidad de ciclos en un grafo completo está dado por la fórmula 2.1, donde C_k^n es el coeficiente binomial (es decir, es el número de subconjuntos de k elementos escogidos de un conjunto con n elementos) y n la cantidad de nodos del grafo.

$$\sum_{k=3}^{n}(k-1)!\frac{C_k^n}{2} \tag{2.1}$$

Así mismo, si ponemos un límite a la cantidad máxima de nodos en el anillo, la cantidad de ciclos sigue siendo potencialmente enorme.

La principal idea de este método es generar anillos de forma iterativa según se los necesite, dado que es imposible evaluar todos los anillos al mismo tiempo. [64]

Se genera un conjunto inicial de anillos y un modelo de programación lineal entera (PLE) determina la mejor forma de rutear las demandas utilizando solo esos anillos. Luego,

nuevos anillos son generados utilizando técnicas de optimización más sofisticadas (*branch-and-price*). Se calcula un premio por cada arista basado en la solución anterior, y estos premios son utilizados para generar un nuevo anillo que mejora la solución actual si se lo incorpora al modelo de optimización junto con los anillos anteriores. El objetivo del problema de generación de ciclos es encontrar uno que maximice los premios recolectados y minimice el costo de usar los vínculos.

2.2.3. P-ciclos

El esquema de protección basado en p-ciclos es una técnica para proteger una red mallada de un fallo en una arista, con la ventaja de tener simultáneamente la velocidad de recuperación de los anillos auto-reparables y la eficiencia en el uso de la capacidad de repuesto de las redes malladas. Posee similitudes conceptuales con el esquema de protección de ruta de repuesto compartida (SBPP, *shared backup path protection*).

Los p-ciclos fueron propuestos por Wayne D. Grover y D. Stamatelakis a fines de los '90. El nombre p-ciclo significa ciclo de protección pre-configurado, pre-conectado.

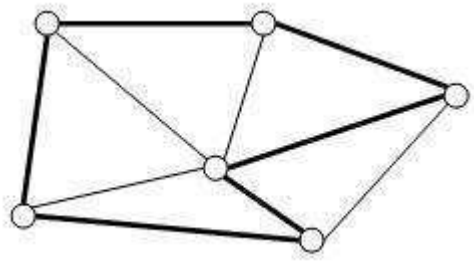

Fig. 2.3: P-ciclo formado sobre red de malla

Los p-ciclos se implementan sobre redes malladas. La idea es utilizar la capacidad de repuesto de éstas, de forma tal que formen anillos lógicos sobre la red mallada, como se muestra en la figura 2.3. Asumiendo que utilizamos la tecnología de anillo *bi-directional line switched ring* (BLSR), entonces solo los dos nodos adyacentes están involucrados en caso de una falla de enlace para redireccionar el tráfico hacia el ciclo pre-configurado. En

la figura 2.4 se muestra el camino de protección resultante para el caso de una falla «en ciclo».

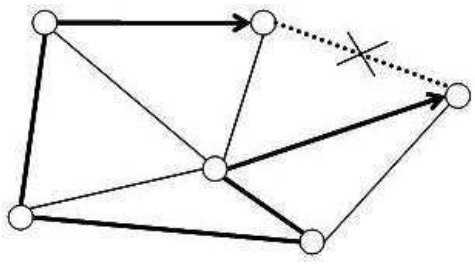

Fig. 2.4: Falla en ciclo y camino de protección

Una de las principales diferencias entre un esquema de anillo auto-reparable y el esquema de p-ciclo es la capacidad de este último para proteger doblemente los vínculos que no están en él (pero cuyos nodos origen y destino si lo están, *i.e.* cuerdas del ciclo). La figura 2.5 muestra los dos caminos de protección resultantes para el caso de una falla «en cuerda». La capacidad de proteger dos canales por cada canal de repuesto permite lograr una eficiencia, con respecto a la capacidad, similar a una malla. Esta característica le da al p-ciclo eficiencia adicional en relación con los esquemas basados en anillos. [42] Una característica que se suele pasar por alto de los p-ciclos es que el ruteo puede ser elegido libremente sobre la red y no se limita a seguir rutas restringidas a la forma del anillo. [3]

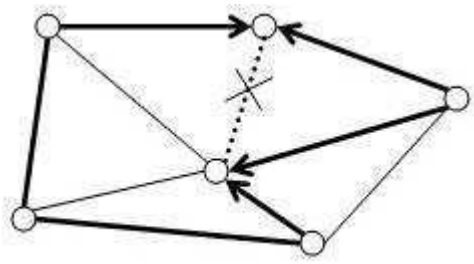

Fig. 2.5: Falla en cuerda y caminos de protección

En caso de falla en un vínculo, los nodos adyacentes detectan el tráfico interrumpido y lo redirigen automáticamente dentro del p-ciclo que protege ese vínculo, sin necesidad de señalización adicional. De esta manera, los p-ciclos logran velocidades de restauración

similares a las de los anillos auto-reparables. Nótese que la protección de un vínculo determinado, dependiendo de la cantidad de unidades a proteger, puede proveerse por más de un p-ciclo. La protección de la red da lugar a la superposición de varios p-ciclos distintos y, por lo tanto, se puede lograr una gran eficiencia con respecto a las capacidades de repuesto similar a las topologías malladas. Los p-ciclos corresponden a esquemas que comparten los vínculos de protección [36].

Resumiendo, los p-ciclos protegen una unidad de demanda en las aristas que están en el ciclo y dos unidades en las aristas que no están en el ciclo pero cuyos nodos sí (*i.e.* cuerdas del ciclo). En el primer caso se protege una unidad, porque, como en los anillos auto-reparables, se redirige el tráfico en el sentido inverso sobre el ciclo. En el segundo, se protegen dos unidades, porque todo el ciclo restante queda intacto y aporta dos caminos. Es importante aclarar que los p-ciclos son ciclos simples y no necesariamente cubren todo el grafo.

Las ventajas de las topologías basadas en p-ciclos son las siguientes:

- Tiempos de restauración muy cortos (similar al de los anillos auto-reparables).
- Planificación de capacidades simple y directa.
- Baja redundancia.
- Equipamiento simple y barato (ejemplo: OADMs, *optical add-drop multiplexers*).
- Eficientes para múltiples clases de servicios.
- Ruteo por camino más corto.
- Caminos de protección pre-reservados.
- Basados en ciclos.
- Protegen de fallas en cuerdas.

- La acción de protección es local.
- Se comparten las capacidades de protección.

2.2.3.1. Tipos de p-ciclos

Existen variaciones del esquema de p-ciclos dependiendo de lo que se busca proteger. Los vistos hasta ahora se encuadran dentro del tipo «p-ciclos de vínculos», dado que su objetivo es proteger el tráfico de una falla en un vínculo. Pero existen otros tipos, como por ejemplo los llamados «p-ciclos de nodos», que buscan proteger el tráfico de una falla en un nodo y los «FIPP p-ciclos», que buscan proteger caminos completos de demandas.

P-ciclos de nodos: para proteger un nodo intermedio de una ruta p, un p-ciclo puede ser construido de forma tal que dos vínculos adyacentes l y l' de un nodo v y sobre la ruta p sean protegidos por el mismo p-ciclo. En caso de una falla en dicho nodo intermedio, los dos nodos más cercanos (adyacentes) en p pueden detectar la falla y enviar el tráfico automáticamente por el resto del p-ciclo como en el caso del «p-ciclo de vínculos». [36]

FIPP (failure-independent path-protecting) p-ciclos: en 2005, Grover *et al.* [27] crean un nuevo tipo de esquema de protección basado en p-ciclos. Se trata de una extensión del concepto básico de p-ciclo orientado a un esquema de protección de rutas. Este esquema protege las rutas (de principio a fin y no solo segmentos) de fallas de nodo o de vínculo. En 2012, Jaumard *et al.* [36] notan que para que los «FIPP p-ciclos» protejan el 100 % de las fallas de nodos, es necesario exigir, adicionalmente, que las rutas sean disjuntas en el sentido de los nodos para cada «FIPP p-ciclo».

2.2.3.2. Eficiencia de los p-ciclos

En [64], Grover *et al.* proponen indicadores de la calidad de un ciclo. Sea:

c_k Costo unitario del ciclo k.

d_e Demanda en la arista e.

p_k^e Coeficiente que codifica la relación de protección entre la arista e y el ciclo elegible k. $p_k^e \in \{0, 1, 2\}$. Vale 1 si la protección es *on-cycle* 2 si es una protección *straddle* y 0 en otro caso.

Puntaje topológico: es la cantidad total de protecciones del ciclo sin considerar las demandas.

$$TS_k = \sum_{e \in E} p_k^e \tag{2.2}$$

Eficiencia a priori: es el puntaje topológico dividido por el costo.

$$AE_k = TS_k / c_k \tag{2.3}$$

La AE_k refleja la eficiencia potencial que podría llegar a lograr ese p-ciclo si todas las cuerdas estuvieran demandadas por dos canales cada una y todas las aristas del ciclo por una.

Estos indicadores tienen dos aspectos: por un lado, ayudan a entender de forma intuitiva qué ciclos son buenos y ayudan a entender por qué una solución óptima no restringida tiende a tener muchos ciclos grandes, incluso Hamiltonianos; por otro lado, sirven de base lógica para construir heurísticas.

Si el ruteo es dado y solo nos interesa ubicar los p-ciclos, entonces podemos aproximar la eficiencia con un indicador que considere esta situación:

Eficiencia para una instancia: es la cantidad de demandas que cubre para una determinada instancia (ruteo) dividido por el costo.

$$E_k^d = \frac{\sum_{e \in E} d_e p_k^e}{c_k} \tag{2.4}$$

Una alternativa similar sería:

$$E_k'^d = \frac{\sum_{e \in E} min(d_e, p_k^e)}{c_k} \tag{2.5}$$

Los p-ciclos pueden lograr una excelente eficiencia en la práctica. Pueden alcanzar la cota mínima largamente reconocida de $1/(\bar{d} - 1)$ donde $\bar{d} = \frac{2|E|}{|V|}$ es el grado promedio. En [64] Grover *et al.* muestran un ejemplo de esta afirmación.

2.3. Consideraciones preliminares e hipótesis de trabajo

A lo largo del presente trabajo nos ocuparemos únicamente de redes de telecomunicaciones tipo *backbone*. Por este motivo, las representaremos mediante grafos simples, no dirigidos, 2-conexos (sin puentes).

Llamaremos *demanda* al tráfico requerido entre un par de nodos, adyacentes o no. Existen dos tipos de problemas en el diseño de redes: decidir por dónde debe ir la demanda (ruteo) y decidir dónde poner las capacidades de repuesto. Estos problemas se pueden enfrentar de forma conjunta o secuencial (primero ruteo de demandas y luego la ubicación de las capacidades de repuesto).

Consideraremos costos en las aristas de los grafos, pero no en los nodos. Esto es una muy buena aproximación de lo que ocurre en la realidad, ya que los costos de los vínculos son proporcionales a la capacidad a instalar y a la distancia por cubrir. Los costos de los nodos se pueden considerar fijos, porque el diferencial (asociado a los nodos) entre los distintos ruteos es despreciable.

Solo nos ocuparemos de escenarios únicos de demandas, es decir que, en nuestros proble-

mas, las demandas son determinísticas. Esta hipótesis también es realista, ya que se suele planificar la red para satisfacer picos de demanda (minoristas) o para capacidad contratada (mayoristas). En las redes que estamos considerando, las demandas se deben considerar discretas (cantidad de canales, OCn en general).

Las fallas pueden presentarse en una arista o en un nodo del grafo. Vamos a considerar escenarios de una sola falla a la vez. Varias fallas simultáneas son posibles, pero mucho menos probables. Vale aclarar que, por ejemplo, si decidimos proteger la red ante una falla en arista, el objetivo es proteger a la red de la falla en cualquier arista.

Los problemas que consideramos aquí no tienen en cuenta la naturaleza modular y no-lineal del costo de incrementar capacidades según los equipos disponibles actualmente. Esto se podría hacer fácilmente modificando los modelos tratados como se describe en Grover *et al.* [64], pág 395.

En el capítulo 3 mostramos los modelos que proponemos para resolver el problema de alocación de capacidades de repuesto en p-ciclos (SCA). Se trata de dos modelos de programación lineal entera mixta (PLEM) y uno de programación lineal entera (PLE) implementados sobre el *branch-and-cut* de CPLEX, dos de programación por restricciones (CP) implementadas sobre el motor de CP de CPLEX, un algoritmo GRASP con búsqueda local exacta y un algoritmo *branch-and-price* exacto. En el capítulo 4 mostramos nuestras propuestas para resolver el problema FIPP. En este caso proponemos un modelo de PLEM (resuelto mediante el algoritmo *branch-and-cut* de CPLEX), uno de programación por restricciones implementado sobre el motor de CP de CPLEX, un algoritmo GRASP también con búsqueda local exacta (utilizando el algoritmo *branch-and-cut* de CPLEX) y un algoritmo *branch-and-price* exacto. En el capítulo final 5 analizamos nuestras conclusiones sobre todos ellos.

3. PROBLEMA DE ALOCACIÓN DE CAPACIDADES DE REPUESTO EN P-CICLOS (SCA)

En el problema **SCA** (*Spare Capacity Allocation*) se trata de cubrir las demandas de cada arista en un grafo con p-ciclos (ciclos simples que cubren una unidad de demanda en sus aristas y dos unidades en sus cuerdas).

3.1. Complejidad del SCA

En 2004, Schupke [72] prueba que la versión de decisión del problema SCA es NP-completo porque se puede reducir el problema de ciclo Hamiltoniano a una instancia del SCA con demandas y costos unitarios para todas las aristas. La idea es que, en ese caso, un ciclo Hamiltoniano domina todas las otras soluciones posibles y, por lo tanto, SCA debería devolver un único ciclo si el grafo es Hamiltoniano y más de uno en caso contrario.

3.2. Dominación de ciclos

Enunciaremos algunas relaciones triviales de dominación entre ciclos y grupos de ciclos que utilizaremos en los modelos y algoritmos subsiguientes. Algunas de estas relaciones se establecen independientemente de las demandas y otras con respecto a éstas.

Proposición 3.2.1. *Si dos ciclos comparten una única arista, y ningún otro nodo además de los dos de dicha arista, entonces el ciclo resultante al combinarlos, mediante la operación de unión disjunta (adición módulo 2, para la cual utilizamos el símbolo $\oplus$) es un ciclo válido*

y domina los ciclos originales. Esto es así, porque el ciclo resultante es más barato y protege en igual cantidad las aristas on-cycle y las aristas cuerda (straddles). Además, en general, puede proteger más aristas straddle que los ciclos originales.

En las figuras 3.1 y 3.2 podemos ver dos ciclos que son dominados por su combinación como se muestra en la figura 3.3. Las aristas más gruesas indican que pertenecen al ciclo (*on-cycle*) y las aristas punteadas indican las cuerdas (*straddles*).

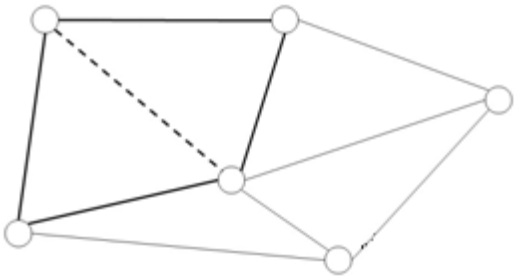

Fig. 3.1: Ciclo original 1

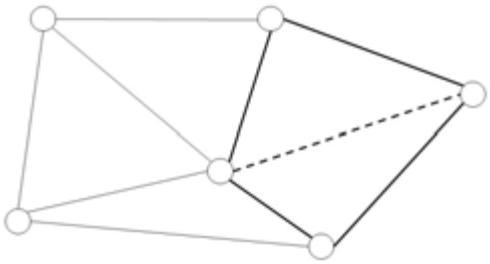

Fig. 3.2: Ciclo original 2

3.3. Bibliografía sobre SCA

A continuación mencionamos algunas referencias bibliográficas respecto a un problema relacionado a SCA que es el de diseño de redes basadas en anillos simples (sin protecciones en las cuerdas, como los p-ciclos). Por ejemplo, Henningsson *et al.* [64] presentan un algoritmo de generación de columnas. Particularmente, estos autores exponen un modelo que

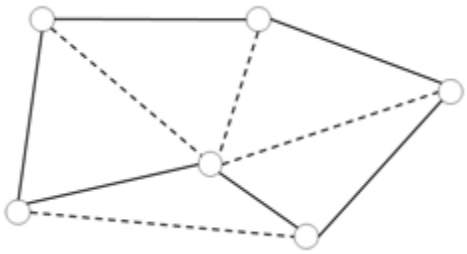

Fig. 3.3: Ciclo resultante de la combinación del ciclo 1 y del 2

llaman *Modelo de Generación de Anillos* para resolver el subproblema de *pricing*. Si el ciclo tuviese que incluir a todos los nodos, se trataría del *Traveling Salesman Problem* (TSP). Sin embargo este no es el caso, lo que nos lleva a una variante del *Traveling Salesman Subtour Problem*. Para más detalles de este problema se referencia a Gensch [22], Mittenthal y Noon [50], Henningsson y Holmberg [31] y Westerlund [82]. Stidsen y Thomadsen [80] llaman a este problema *Quadratic Selective Travelling Salesman problem*.

En 2002, Rajan y Atamtürk [60], trabajaron sobre el problema *Joint Capacity Allocation* (JCA), el cual es similar a SCA, pero incluyendo las decisiones de ruteo. Desarrollaron un algoritmo de generación de columnas basado en variables llamadas *cycle-slack* que determinan la cantidad de capacidad de repuesto asignada a un ciclo en particular. Utilizando este tipo de formulación (Ryan Foster), evitan la pérdida de eficiencia cuando en un esquema de generación de columnas se destruye la estructura del problema de pricing. Además, los autores demuestran que el problema de *pricing* es *NP-hard*. Éste es exactamente igual a nuestro problema de *pricing* del esquema de generación de columnas propuesto en la sección 3.7.

En 2004, Stidsen y Thomadsen [80] desarrollaron un esquema de generación de columnas heurístico para el problema JCA. Estos autores dividieron el problema de *pricing* en dos: uno que determina las rutas y otro los ciclos. Mak y Thomadsen [48] realizaron un estudio poliedral del problema de *pricing*.

Respecto de SCA, Grover *et al.* [64] plantean un modelo sencillo que resuelven median-

te un algoritmo *Branch-and-Cut* y que veremos en la próxima sección 3.4. Este modelo requiere conocer los ciclos, los autores optaron por no incluirlos todos, sino solo aquellos que tienen indicadores promisorios. Los indicadores utilizados son los que se describieron en el capítulo anterior.

En 2004 Schupke [72] formuló un modelo de programación lineal entera sin enumeración de ciclos. En ese modelo, los ciclos son definidos por medio de restricciones de flujos y generan como máximo una cantidad predeterminada de J ciclos, llamados CS_j. Esta técnica se asegura de generar a lo sumo un ciclo en cada CS_j. Para eso necesita de restricciones de eliminación de *subtours*. El modelo permite identificar las cuerdas (*straddles*) y tiene una gran cantidad de variables, pero es de orden polinomial. El modelo tiene $J(|V|2+|V||E|+4|E|+|V|+1)$ variables y $J(|V|2+|V||E|+9|E|+|V|+1)+|E|$ restricciones.

En 2008, Wu *et al.* [84] formularon tres nuevos modelos basados en el trabajo de Schupke, considerando también un máximo de J ciclos. Uno de esos modelos es de conservación de flujo, utilizando una técnica de flujo no dirigido. Este modelo tiene $J|E|2+J|E||V|+J|V|$ variables y $J|E|(|E|+|V|+2)+J|V|+|E|$ restricciones. Otro modelo, que generó los mejores resultados en sus pruebas, llamado «de exclusión de ciclos», está basado en la idea de voltaje eléctrico para asegurar un único ciclo para cada CS_j. Este modelo tiene $3J(|E|+|V|)$ variables y $4J|E|+2J|V|+|E|+J$ restricciones.

A continuación detallamos el modelo «de exclusión de ciclos»:

Coeficientes:

$\alpha = 1/\|V\|$	Coeficiente para garantizar voltajes compatibles.
$costo_e$	Costo de la arista e.
d_e	Demanda en la arista e.

Variables:

x_{uv}^j Variable binaria. Toma valor 1 si el ciclo pasa por la arista $e = uv$ en CS_j, 0 en caso contrario.

ζ_u^j Variable binaria. Toma valor 1 si el ciclo pasa por el nodo u en CS_j, 0 en caso contrario.

χ_{uv}^j Variable binaria. Toma valor 1 si la arista $e = uv$ es protegida en CS_j, 0 en caso contrario.

z_u^j Variable binaria. Toma valor 1 si el nodo u es nodo raíz en CS_j, 0 en caso contrario.

$volt_u^j$ Variable continua. Es el valor del voltaje del nodo u en CS_j.

Formulación:

$$\min \sum_{j\in\{1...J\}} \sum_{e=uv\in E} c_e(x_{uv}^j + x_{vu}^j) \tag{3.1}$$

sujeto a

$$x_{uv}^j + x_{vu}^j \leq 1, \forall e = uv \in E, \forall j \in \{1..J\} \tag{3.2}$$

$$\sum_{e=uv\in E} (x_{uv}^j + x_{vu}^j) = 2\zeta_u^j, \forall u \in V, \forall j \in \{1..J\} \tag{3.3}$$

$$\sum_{j\in\{1...J\}} (2\chi_{uv}^j - x_{uv}^j - x_{vu}^j) \geq d_{e=uv}, \forall e = uv \in E \tag{3.4}$$

$$\sum_{u\in V} z_u^j \leq 1, \forall j \in \{1..J\} \tag{3.5}$$

$$\sum_{e=uv\in E} x_{uv}^j \leq 1 + z_u^j, \forall u \in V, \forall j \in \{1..J\} \tag{3.6}$$

$$volt_v^j - volt_u^j \geq \alpha x_{uv}^j - (1 - x_{uv}^j), \forall (uv), (vu) \in E, \forall j \in \{1..J\} \tag{3.7}$$

$$\chi_{uv}^j \leq \frac{1}{2}(\zeta_u^j + \zeta_v^j), \forall e = uv \in E, \forall j \in \{1..J\} \tag{3.8}$$

La función objetivo indica que buscamos minimizar el costo de los p-ciclos a utilizar. La restricción 3.2 vincula a cada arista con a lo sumo un ciclo en cada CS_j, en un sentido, en el otro o en ninguno. La restricción 3.3 obliga a que las variables que representan los nodos del ciclo solo puedan valer 1 si el nodo tiene dos aristas adyacentes en el ciclo. La inecuación 3.4 asegura la cobertura de las demandas. La restricción 3.5 permite a lo sumo un nodo raíz en cada CS_j. La restricción 3.6 establece que solo el nodo raíz puede servir como cabeza de un ciclo. La restricción 3.7 especifica que la cola de un ciclo debe tener mayor voltaje que su cabeza. Finalmente, la restricción 3.8 indica que una arista puede ser protegida por el único ciclo en CS_j si sus dos nodos están en CS_j.

En 2011, [55] propusimos tres modelos que veremos en detalle en la sección siguiente.

En 2013, Delgadillo [15] también propuso un modelo que genera a lo sumo J ciclos basado en flujos, utilizando las restricciones de eliminación de *subtours* de Gavish y Graves [21] con mejores resultados experimentales que los modelos anteriores.

3.4. Modelos de programación entera mixta para SCA

Dado un ciclo, podemos calcular fácilmente su costo y sus cuerdas y, por lo tanto, podemos determinar las demandas que protege. Si tuviésemos todos los ciclos posibles del grafo, podríamos obtener todas las combinaciones posibles de ciclos que cubran todas las demandas de la red. Podríamos, así mismo, obtener el óptimo inspeccionando cuál o cuáles de estas combinaciones satisfacen las demandas y es la más barata, pero sería una forma extremadamente poco eficiente: recordemos que un grafo completo tiene una cantidad exponencial de ciclos.

Podemos encontrar todos los ciclos posibles de un grafo utilizando alguno de los algoritmos conocidos de teoría de grafos, como por ejemplo el algoritmo de Johnson [39], el de Tarjan [81] o el de Welch-Gibbs [44]. En el trabajo de Mateti *et al.* [49] y en el de Boon Chai Lee [44] podemos encontrar una compilación de los principales algoritmos. En 2013, Birmelé *et al.* [12] desarrollaron un algoritmo para grafos no dirigidos que es asintóticamente óptimo en el sentido de las operaciones necesarias.

Si disponemos de todos los ciclos podemos formular un simple modelo de programación lineal entera para determinar la solución óptima del SCA. Aquí presentamos una versión simplificada (sin variables de ruteo) del modelo propuesto por Grover *et al.* [64]:

Coeficientes:

p_k^e; este coeficiente codifica la relación de protección entre la arista e y el ciclo elegible k. $p_k^e \in \{0, 1, 2\}$. Toma el valor 1 si la arista e pertenece al ciclo k y por lo tanto es protegida una sola vez, toma el valor 2 si la arista e es una cuerda del ciclo k y es nulo en cualquier otro caso.

d_e; demanda de la arista e.

c_k; costo unitario del ciclo k.

Variables:

$x_k \geq 0$: variable entera. Representa la cantidad de copias del ciclo k.

Formulación:

$$\min \sum_{k \in K} c_k x_k \tag{3.9}$$

sujeto a

$$\sum_{k \in K} p_k^e x_k \geq d_e \ \forall e \in E \tag{3.10}$$

La restricción 3.10 exige que los p-ciclos cubran las demandas de cada arista del grafo.

El problema con este modelo es que puede tener una cantidad exponencial de variables. Por este motivo, este modelo solo puede ser implementado si el grafo es muy pequeño o muy ralo. Este modelo es muy apto para ser resuelto mediante algoritmos de generación de columnas como veremos más adelante en este capítulo.

En las próximas secciones expondremos nuestras propuestas de modelos de programación lineal entera mixta sin enumeración de ciclos. Estos modelos fueron inicialmente pensados como modelos para el problema de *pricing* (con algunas modificaciones) para un algoritmo *Branch-and-Price*. Pero debido a que en la práctica se requieren pocos ciclos en la solución óptima, decidimos evaluar su utilidad como modelos independientes. Su importancia es doble, por un lado como modelos para algoritmos de *pricing*, ya que es el cuello de botella de nuestro algoritmo *Branch-and-Price* (que considera al modelo de Grover *et al.* como problema maestro); y, por otro, como modelos que tampoco requieren la enumeración previa de los ciclos, implementados sobre algoritmos *Branch-and-Cut*.

La cantidad mínima de ciclos para proteger una red con p-ciclos es de $J \geq \lceil D/2 \rceil$ donde D es el valor máximo de demanda en una arista del grafo G. También podemos decir que la cantidad máxima de ciclos es $J \leq \sum_{i \in E} d_i$ donde d_i es la demanda en la arista i. Por lo tanto tenemos:

$$\sum_{i \in E} d_i \geq J \geq \lceil D/2 \rceil \tag{3.11}$$

La cota superior es ajustada en el peor caso, como podemos ver en la figura 3.4. En este grafo hay ciclos de tres aristas con costo nulo y demanda igual a uno en una de las aristas y nula en las demás. Cada ciclo está conectado a su ciclo vecino por aristas con costo infinito y demanda nula.

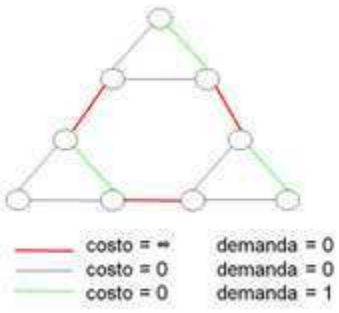

Fig. 3.4: Cota superior para J.

Esto podría ser visto como una limitación, pero en las redes reales que evaluamos en la sección 3.8, el óptimo no se encuentra lejos de la cota inferior. A modo de ejemplo, en la red COST239 (detallada en B.1), utilizada por la comunidad como *benchmark*, $D = 11$, por lo tanto $J \geq 6$ y la cantidad de ciclos en sus 19 soluciones óptimas es de 7, incluso para las relajaciones lineales. En las redes reales, el costo está muy relacionado a la distancia y las demandas no son tan ralas como en el ejemplo de la figura 3.4. Por otro lado D no suele ser demasiado mayor que el resto de las demandas.

El modelo para el problema de *pricing* presentado por Stidsen y Thomadsen [80] dentro de un algoritmo *Branch-and-Price* para el problema JCA podría, en principio, ser

convertido en un modelo independiente.

3.4.1. Modelo PLE basado en eliminación de *subtours* Miller-Tucker-Zemlin.

En el primer modelo que proponemos para representar el problema SCA sin enumeración previa de ciclos, como en los modelos de Schupke [72] y de Wu *et al.* [84], también limitamos la cantidad de ciclos a una cantidad J arbitraria. Esto es necesario para determinar las aristas que son cuerdas del p-ciclo (*straddles*). Llamamos «conjunto de ciclos» CS_j donde $j \in \{1 \dots J\}$ a los ciclos generados por el modelo.

La idea de este modelo es usar las restricciones Miller-Tucker-Zemlin (MTZ) de eliminación de *subtours*. La formulación original MTZ fue propuesta para el problema del viajante de comercio (TSP) donde los ciclos a considerar pasan por todos los nodos y requiere de una variable entera u que establece el orden de cada nodo en el ciclo. Se puede fijar como primer nodo a un nodo arbitrario. Las variables binarias x_{ij} indican si un arco (del nodo i al nodo j) pertenece al ciclo.

$$u_1 = 1 \tag{3.12}$$

$$2 \leq u_i \leq n, \qquad \forall i \in V \tag{3.13}$$

$$u_i - u_j + 1 \leq n(1 - x_{ij}), \qquad \forall i \neq 1, \qquad \forall j \neq 1 \tag{3.14}$$

En nuestro problema, los ciclos no tienen por qué cubrir todos los nodos, por lo tanto necesitamos una variable z_k para determinar el nodo u_1, el cual es único para cada CS_j. Esto, a su vez, crea simetría en el espacio de soluciones ya que, para cada solución que nos interesa, se repiten todas las combinaciones posibles de nodos u_1 de cada ciclo. Para evitar esto, veremos en la proposición 3.4.2 que podemos exigir que el índice de z_k sea el menor posible.

La formulación MTZ requiere que para cada arista $e = (u, v)$, definamos dos variables

x, una en cada sentido, es decir x_{e0} para representar el sentido uv (u: cola, v: cabeza) y x_{e1} para representar el sentido vu (v: cola, u: cabeza). Solo una de las dos puede valer 1. La variable y_e^j representa las cuerdas de los ciclos.

3.4.1.1. Coeficientes:

L: Número muy grande.

3.4.1.2. Variables:

x_{eo}^j: variable binaria. Toma valor 1 si la arista e está en el ciclo generado CS_j. Los arcos son dirigidos y el índice o indica el sentido ($o \in \{0, 1\}$).

x_o^j: vector de variables x_{eo}^j de tamaño m.

y_e^j: variable binaria. Toma valor 1 si la arista e es una cuerda (*straddle*) del ciclo generado j.

z_k^j: variable binaria. Solo una de ellas toma valor 1 en cada ciclo generado para reemplazar al $u_1 = 1$ (primer nodo del ciclo) de la formulación Miller-Tucker-Zemlin.

u_k^j: variable entera. Esta variable sirve para representar la posición que ocupa el nodo k en el ciclo dentro de la formulación Miller-Tucker-Zemlin.

3.4.1.3. Formulación:

$$\min \sum_{j\in\{1..J\}} \sum_{e\in E} costo_e(x^j_{e0} + x^j_{e1}) \tag{3.15}$$

sujeto a

$$\begin{aligned}
x^j_{e0} + x^j_{e1} &\leq 1,\ \forall e \in E,\ \forall j \in \{1..J\} && (3.16)\\
Mx^j &\leq 2,\ \forall j \in \{1..J\} && (3.17)\\
\frac{1}{2}(x^j_0 + x^j_1)(\frac{1}{2}M^tM - 2I) &\geq y^j,\ \forall j \in \{1..J\} && (3.18)\\
\sum_{e=uv\in E} (x^j_{uv0} - x^j_{uv1}) &= 0,\ \forall u \in V,\ \forall j \in \{1..J\} && (3.19)\\
d_e &\leq \sum_{j\in\{1..J\}} x^j_e + 2.y^j_e,\ \forall e \in E && (3.20)\\
\sum_{k\in V} z^j_k &= 1,\ \forall j \in \{1..J\} && (3.21)\\
1 &\leq z^j_k + u^j_k \leq |V|,\ \forall k \in V,\ j \in \{1..J\} && (3.22)\\
(u^j_d + z^j_d) - (u^j_h + z^j_h) + 1 &\leq |V|(1 - x^j_{e,o}) + z^j_d L + z^j_h L && (3.23)\\
& \forall e = (dh) \in E,\ o \in \{0,1\},\ j \in \{1..J\} && (3.24)\\
(u^j_h + z^j_h) - (u^j_d + z^j_d) + 1 &\geq -|V|(1 - x^j_{e,o}) - z^j_h L - z^j_d L && (3.25)\\
& \forall e = (dh) \in E,\ o \in \{0,1\},\ j \in \{1..J\} && (3.26)\\
& x^j_e \in \{0,1\},\ \forall j \in \{1..J\},\ e \in E,\ o \in \{0,1\} && (3.27)\\
& y^j_e \in \{0,1\},\ \forall j \in \{1..J\},\ e \in E && (3.28)\\
& z^j_k \in \{0,1\},\ \forall j \in \{1..J\},\ k \in V && (3.29)\\
& u^j_k \in \{0,n\},\ \forall j \in \{1..J\},\ k \in V && (3.30)
\end{aligned}$$

La restricción 3.16 establece un único sentido para cada arista en cada CS_j. La restricción 3.17 limita el grado de cada nodo a un máximo de dos y la restricción 3.19 obliga a

que los sentidos de las aristas sean coherentes entre sí.

En este modelo proponemos una representación diferente de las cuerdas. Sea M la matriz de incidencia del grafo $G = (V, E)$. Si x representa un ciclo en el sentido de las aristas, entonces $r = \frac{1}{2}Mx$ representa los nodos del ciclo y sus elementos pueden tomar únicamente valores en $\{0, 1\}$. El resultado de $x(M^t r - 2I)$ es un vector cuyos elementos pueden tomar únicamente valores en $\{0, 1, 2\}$. El significado de esos valores corresponde a la cantidad de nodos del p-ciclo incidentes a cada arista restando las adyacencias debidas a las aristas del ciclo en sí mismo. Por lo tanto, cuando ese valor es 2 se trata de una cuerda. Este concepto se utiliza en la restricción 3.18.

La restricción 3.20 obliga a cumplir con las demandas de cada arista. Finalmente, la formulación MTZ es expresada mediante las restricciones 3.21, 3.22, 3.23 y 3.25. La restricción 3.21 determina un único nodo inicial del ciclo. La restricción 3.22 acota los valores de orden de los nodos.

El modelo tiene $3J|E| + 2J|V|$ variables y $6J|E| + 4J|V| + |E| + J$ restricciones.

3.4.1.4. Cortes de optimalidad:

Los cortes de optimalidad son restricciones que solo deben ser satisfechas por las soluciones óptimas del problema y no necesariamente por el resto de las soluciones factibles. Restringen el espacio de búsqueda eliminando soluciones válidas que se sabe no podrán ser soluciones óptimas del problema.

Para eliminar parte de la simetría, podemos exigir que el tamaño del ciclo generado en un CS_j sea mayor o igual al tamaño del ciclo del conjunto siguiente CS_{j+1}, como lo establece la restricción 3.31.

$$\sum_{e \in E} (x_{e0}^{j+1} + x_{e1}^{j+1}) \leq \sum_{e \in E} (x_{e0}^{j} + x_{e1}^{j}), \forall j \in \{1, \ldots, J-1\} \tag{3.31}$$

Proposición 3.4.1. *Dada una solución (factible u óptima) del problema SCA, se la representa en esta formulación en cada permutación de CS_j. Por lo tanto, podemos restringir el espacio de soluciones del modelo mediante la restricción 3.31 permitiendo al menos una permutación de cada solución.*

También, para eliminar parte de la simetría, podemos exigir que z_k^j de cada CS_j sea lo menor posible. Como lo establece la restricción 3.32.

$$z_v^j - z_k^j \leq 1 - (x_{e0}^j + x_{e1}^j), \forall e = (k, v) \in E | v > k, \forall j \in \{1, \ldots, J\} \tag{3.32}$$

Proposición 3.4.2. *Dado un ciclo en CS_j, éste es representado, en esta formulación, en cada valor de z_k^j de cada nodo. Por lo tanto, podemos restringir el espacio de soluciones del modelo mediante la restricción 3.32, permitiendo al menos una permutación de cada solución.*

3.4.1.5. Desigualdades válidas:

Las desigualdades válidas son restricciones que son satisfechas por todas las soluciones factibles del problema. Se utilizan para eliminar soluciones no enteras en el árbol de búsqueda.

Proposición 3.4.3. *Para cada CS_j, la cantidad de aristas en el ciclo no puede ser mayor a la cantidad de nodos. Por lo tanto, la restricción 3.33 es una desigualdad válida.*

$$\sum_{e \in E} (x_{e0}^j + x_{e1}^j) \leq |V|, \forall j \in \{1..J\} \tag{3.33}$$

Proposición 3.4.4. *Una arista puede estar en el ciclo o ser una cuerda, pero no las dos a la vez. Por lo tanto, la restricción 3.34 es una desigualdad válida.*

$$(x_{e0}^j + x_{e1}^j + y_e^j) \leq 1, \forall e \in E, \forall j \in \{1..J\} \tag{3.34}$$

Estas desigualdades 3.34, reemplazan a las restricciones 3.16 del modelo ya que son más fuertes.

Proposición 3.4.5. *La sumatoria de las variables que representan a las aristas que están en el ciclo más sus cuerdas no puede superar la cantidad total de aristas para cada CS_j. Por lo tanto, la restricción 3.35 es una desigualdad válida.*

$$\sum_{e \in E} (x_{e0}^j + x_{e1}^j + y_e^j) \leq |E|, \forall j \in \{1..J\} \tag{3.35}$$

La desigualdad 3.35 no es utilizada en los experimentos de la sección 3.8 ya que podemos derivarla de sumar las desigualdades 3.34 para todas las aristas.

3.4.2. Modelo PLEM basado en restricciones de voltaje eléctrico.

Acá proponemos una variante al modelo anterior, en la que reemplazamos las restricciones de eliminación de *subtour* de MTZ por una formulación inspirada en las restricciones de voltaje eléctrico similares a las de Wu *et al.* [84] mencionadas en la sección 3.3. Nosotros utilizamos una notación y formulación ligeramente diferente. El modelo resultante es relativamente compacto ya que posee $3J|E| + J|V|$ variables enteras, $J|V|$ variables continuas y $4J|E| + 2J|V| + |E| + J$ restricciones. Si bien existe una cierta semejanza entre las restricciones MTZ y las de voltaje eléctrico, la diferencia radica en que se utilizan variables continuas.

Como en el modelo anterior, para cada arista $e = (u, v)$, definimos dos variables x, una en cada sentido, es decir x_{e0} para representar el sentido uv (u: cola, v: cabeza) y x_{e1} para representar el sentido vu (v: cola, u: cabeza). Solo una de las dos puede valer 1. En esta formulación la variable x también representa una circulación y no un flujo como en el modelo de Wu *et al.* Un valor continuo, llamado voltaje, es asignado a cada nodo. Exigimos para cada arista que la cola tenga un valor más alto que la cabeza para todos los nodos excepto el nodo raíz, el cual es único para cada CS_j. De esta forma, asegurarse un único ciclo en cada CS_j equivale a asegurarse la compatibilidad de voltajes.

Coeficientes:

$\alpha = 1/|V|$: coeficiente para garantizar voltajes compatibles.

L: número muy grande.

Variables:

x^j_{eo}: variable binaria. Toma valor 1 si la arista e está en el ciclo generado CS_j. Los arcos son dirigidos y el índice o indica el sentido.

y^j_e: variable binaria. Toma valor 1 si la arista e es una cuerda (*straddle*) del ciclo generado

j.

z_k^j: variable binaria. Solo una de ellas toma valor 1 en cada ciclo generado para indicar el nodo raíz de los voltajes.

$volt_k^j$: variable continua. Indica el voltaje de cada nodo.

Formulación:

$$\min \sum_{j\in\{1..J\}} \sum_{e\in E} costo_e(x_{e0}^j + x_{e1}^j) \tag{3.36}$$

sujeto a

$$\begin{aligned}
\sum_{e=uv\in E} (x_{uv0}^j - x_{uv1}^j) &= 0, \forall u \in V,\, j \in \{1..J\} && (3.37)\\
Mx^j &\leq 2, \forall j \in \{1..J\} && (3.38)\\
\frac{1}{2}(x_0^j + x_1^j)(\frac{1}{2}M^tM - 2I) &\geq y^j, \forall j \in \{1..J\} && (3.39)\\
x_0^j + x_1^j &\leq 1, \forall e \in E,\, j \in \{1..J\} && (3.40)\\
d_e &\leq \sum_{j\in\{1..J\}} (x_e^j + 2.y_e^j), \forall e \in E && (3.41)\\
\sum_{k\in V} z_k^j &= 1, \forall j \in \{1..J\} && (3.42)\\
volt_h^j - volt_d^j &\geq \alpha x_{eo}^j - (1 - x_{eo}^j) - Lz_h^j && (3.43)\\
& \forall e = (d,h) \in E,\, o \in \{0,1\},\, \forall j \in \{1..J\} && (3.44)\\
& x_e^j \in \{0,1\},\, \forall j \in \{1..J\},\, e \in E, o \in \{0,1\} && (3.45)\\
& y_e^j \in \{0,1\},\, \forall j \in \{1..J\},\, e \in E && (3.46)\\
& z_k^j \in \{0,1\},\, \forall j \in \{1..J\},\, k \in V && (3.47)\\
& volt_k^j \in \{0,\infty),\, \forall j \in \{1..J\},\, k \in V && (3.48)
\end{aligned}$$

Las restricciones 3.37, 3.38, 3.39, 3.40 y 3.41 son iguales al modelo anterior. Es decir, que la restricción 3.40 establece un único sentido para cada arista en cada CS_j, la restricción 3.38 limita el grado de cada nodo a un máximo de dos, la restricción 3.37 obliga a que los sentidos de las aristas sean coherentes entre sí, la restricción 3.39 determinar las cuerdas y la restricción 3.41 obliga a cumplir con las demandas.

La eliminación de *subtours* es diferente y está representada en las restricciones 3.42 y 3.43. Las restricciones 3.42 determinan la unicidad del nodo raíz dentro de cada CS_j, de forma similar a la formulación MTZ. Las restricciones 3.43 exigen la compatibilidad de los voltajes; es decir, que la cola de un ciclo debe tener mayor voltaje que su cabeza.

Las desigualdades válidas 3.33, 3.34 y 3.35 y cortes de optimalidad 3.31 y 3.32 del modelo anterior también son válidas en este modelo.

3.4.3. Modelo PLEM basado en el espacio de ciclos

El otro modelo para el problema SCA que proponemos en este trabajo utiliza el concepto de espacio de ciclos con restricciones de eliminación de *subtours* de voltaje eléctrico idénticas a las del modelo anterior.

Los ciclos de un grafo se pueden representar como vectores binarios, donde la i-ésima posición indica si la i-ésima arista es parte del ciclo. Los ciclos y la unión disjunta de ciclos (adición módulo 2 o XOR), en el sentido de las aristas, forman el espacio que llamaremos $C = C(G)$. El espacio de ciclos $C = C(G)$ es el subespacio de $E(G)$ abarcado por todos los ciclos en G. La dimensión de $C = C(G)$ es el número ciclomático $\mu = |E| - |V| + c$, donde c es la cantidad de componentes conexos del grafo. Así, en los grafos 2-conexos, como en nuestro caso, $c = 1$. Para más detalles sobre el concepto de espacios de ciclos ver el libro de Diestel [16] (pág. 21) o el libro de Jungnickel [40] (pág. 265).

Si consideramos un árbol de cobertura T de G, entonces cada arista no en T determina un único ciclo si es agregada al árbol. Estos son llamados *ciclos fundamentales de G respecto de T*. El conjunto de todos los ciclos fundamentales respecto de un árbol de cobertura arbitrario es una base del espacio de ciclos $C(G)$. Por lo tanto, este modelo requiere que se calcule previamente un árbol de cobertura del grafo y que se determine el correspondiente conjunto de ciclos fundamentales.

Coeficientes:

$\alpha = 1/|V|$: coeficiente para garantizar voltajes compatibles.

B: matriz de ciclos fundamentales del grafo G (μ x m).

C: conjunto de ciclos fundamentales con respecto a un árbol arbitrario.

L: número muy grande.

M: matriz de incidencia del grafo G (n x m).

M^t: matriz de incidencia transpuesta del grafo G (m x n).

Variables:

c_i^j: variable binaria. Toma valor 1 si el ciclo CS_j es generado por el ciclo fundamental i.

p_e^j: variable entera. Es una variable auxiliar que indica cuántos pares de ciclos fundamentales son seleccionados para generar el ciclo j que incluyen a la arista e.

x_{eo}^j: variable binaria. Toma valor 1 si la arista e está en el ciclo generado CS_j. Los arcos son dirigidos y el índice o indica el sentido, $o \in \{0, 1\}$.

x_o^j: vector de variables x_{eo}^j de tamaño m.

x^j: suma de vectores de variables x_{eo}^j. Es decir, $x^j = x_0^j + x_1^j$.

y_e^j: variable binaria. Toma valor 1 si la arista e es una cuerda (*straddle*) del ciclo generado CS_j.

z_k^j: variable binaria. Solo una de ellas toma valor 1 en cada ciclo generado para indicar el nodo raíz de los voltajes.

$volt_k^j$: variable continua. Indica el voltaje de cada nodo.

Formulación:

$$\min \sum_{j\in\{1..J\}} \sum_{e\in E} costo_e(x^j_{e0} + x^j_{e1}) \tag{3.49}$$

sujeto a

$$\begin{aligned}
B.c^j &= 2.p^j + x^j, \forall j \in \{1..J\} && (3.50)\\
x^j_0 + x^j_1 &\leq 1, \forall e \in E,\, j \in \{1..J\} && (3.51)\\
\sum_{e=uv\in E} (x^j_{uv0} - x^j_{uv1}) &= 0, \forall u \in V,\, j \in \{1..J\} && (3.52)\\
Mx^j &\leq 2, \forall j \in \{1..J\} && (3.53)\\
\frac{1}{2}(x^j_0 + x^j_1)(\frac{1}{2}M^tM - 2I) &\geq y^j, \forall j \in \{1..J\} && (3.54)\\
d_e &\leq \sum_{j\in\{1..J\}} (x^j_e + 2.y^j_e), \forall e \in E && (3.55)\\
\sum_{k\in V} z^j_k &= 1, \forall j \in \{1..J\} && (3.56)\\
volt^j_h - volt^j_d &\geq \alpha x^j_{eo} - (1 - x^j_{eo}) - Lz^j_h && (3.57)\\
& \forall e = (d,h) \in E,\, o \in \{0,1\},\, j \in \{1..J\} && (3.58)\\
& c^j_i \in \{0,1\},\, \forall j \in \{1..J\},\, i \in C && (3.59)\\
& p^j_e \in \{0,\infty),\, \forall j \in \{1..J\},\, e \in E && (3.60)\\
& x^j_e \in \{0,1\},\, \forall j \in \{1..J\},\, e \in E, o \in \{0,1\} && (3.61)\\
& y^j_e \in \{0,1\},\, \forall j \in \{1..J\},\, e \in E && (3.62)\\
& z^j_k \in \{0,1\},\, \forall j \in \{1..J\},\, k \in V && (3.63)\\
& volt^j_k \in \{0,\infty),\, \forall j \in \{1..J\},\, k \in V && (3.64)
\end{aligned}$$

La principal diferencia en la formulación es que utilizamos una variable c^j_i que indica los ciclos fundamentales involucrados. Las restricciones 3.50 establecen que el ciclo generado es

una combinación de uniones disjuntas de ciclos fundamentales, donde la variable x^j indica el ciclo y la variable p^j descuenta los pares de aristas de ciclos fundamentales en esa arista

La formulación tiene $5J|E|+J$ variables enteras, $J|V|$ continuas y $5J|E|+2J|V|+J+|E|$ restricciones. Pero en relación con las variables enteras p, podemos decir que para todas las aristas que pertenecen a un solo ciclo fundamental p es nula. De esta manera solo tenemos $J|V|$ variables p y no $J|E|$. La cantidad de variables enteras queda entonces en $4J|E| + J|V| + J$.

3.4.3.1. Desigualdades válidas:

Las restricciones 3.33, 3.34 y 3.35 también son desigualdades válidas en esta formulación por los mismos motivos que en las formulaciones anteriores.

Llamamos I_e a la cantidad de ciclos fundamentales en los cuales la arista $e \in E$ forma parte.

$$p_e^j \leq \left\lfloor \frac{I_e}{2} \right\rfloor, \forall e \in E, \forall j \in \{1, \ldots, J\} \tag{3.65}$$

Proposición 3.4.6. *Dado que la variable p_e^j representa los pares de ciclos fundamentales de una solución factible, es trivial acotar a la mitad la cantidad máxima de intersecciones entre ciclos fundamentales para cada CS_j.*

Podemos pensar un grafo F que llamaremos grafo de ciclos de G. Sus nodos corresponden a los ciclos del grafo G y sus aristas corresponden a las intersecciones entre ciclos en el grafo G. Es decir que una arista en F, $f = (c_1, c_2)$ indica que los ciclos c_1 y c_2 de G tienen aristas en común. Recordemos que G es biconexo, y que los ciclos fundamentales forman

una base del espacio de ciclos, por estas razones, podemos decir que siempre existe un árbol de cubrimiento del grafo F utilizando solo los ciclos fundamentales. Esta propiedad da lugar a la siguiente proposición y su correspondiente desigualdad válida. Esta última genera importantes mejoras en la eficiencia del modelo.

Proposición 3.4.7. *Dado el grafo G, y su grafo de ciclos F. Si G es biconexo, siempre existe un árbol de cubrimiento del grafo F utilizando solo los ciclos fundamentales. Por lo tanto, dado un ciclo en CS_j, la desigualdad 3.66 es una desigualdad válida.*

$$\sum_{i \in C} c_i^j \leq \sum_{e \in E} p_e^j + 1, \, \forall j \in \{1, \ldots, J\} \tag{3.66}$$

3.4.3.2. Cortes de optimalidad:

Las restricciones 3.31 y 3.32 también son cortes de optimalidad en esta formulación por los mismos motivos que en las formulaciones anteriores.

3.5. Modelos de programación por restricciones para SCA

3.5.1. Motivación

El desarrollo de un algoritmo *Branch-and-Price* requiere de un algoritmo de generación de ciclos que mejore una determinada solución. Los algoritmos de Programación por Restricciones suelen ser poco eficientes en la búsqueda de una solución óptima, pero son muy veloces para encontrar una solución factible. Entonces, si en el modelo de Programación por Restricciones exigimos, mediante una restricción, que el ciclo mejore la solución actual, tendremos un algoritmo de búsqueda exacto para utilizar como algoritmo de *pricing*.

Por otro lado, dentro del esquema de ramificación del algoritmo *Branch-and-Price*, necesitamos asegurarnos de que ciertos ciclos no sean propuestos como ciclos de mejora debido a que las variables que los representan están acotadas superiormente en un determinado nodo. Este requisito resta eficiencia a los algoritmos de *pricing* basados en PLE, pero los de Programación por Restricciones se ven afectados en menor grado.

A continuación, explicaremos brevemente Programación por Restricciones y propondremos dos modelos para resolver el problema SCA; el mejor de estos modelos se adaptará al problema de *pricing* (y competirá con los modelos de PLE y PLEM) que analizaremos en la sección de *Branch-and-Price* 3.7.

3.5.2. Programación por Restricciones

Programación por restricciones (o programación con restricciones) es un paradigma de programación donde las relaciones entre variables son establecidas en forma de restricciones. Ellas no especifican pasos o secuencias de pasos a ser ejecutados, sino que implican propiedades de una solución a ser encontrada. Esto hace de la Programación por Restricciones una forma de programación declarativa. Las restricciones utilizadas pueden ser restricciones lógicas o matemáticas (lineales o no lineales) y las disponibles están usual-

mente embebidas en un lenguaje de programación o provistas por librerías de software.

Cuando un programa o modelo de programación por restricciones es expresado en forma de programación por restricciones lógicas (*constraint logic programming*), éste se expresa como un programa lógico. Ejemplos de lenguajes de programación por restricciones lógicas son Prolog III, CLP(R) y CHIP. En programación por restricciones, las restricciones son mezcladas con programación funcional, reescritura de términos y lenguajes imperativos. Ejemplos de software de programación por restricciones son CHOCO, Google CP Solver, Gurobi [25] e IBM ILOG CP. Para más detalles sobre programación por restricciones referimos al lector a Rossi *et al.* [65].

Desarrollamos dos modelos en programación por restricciones para resolver SCA, ambos del tipo constructivo, es decir, no requieren de la enumeración previa de los ciclos. Al igual que en los modelos vistos anteriormente, construimos J ciclos.

3.5.3. Modelo Constructivo

El primer modelo (llamado **Constructivo CP**), está basado en la construcción de ciclos en un vector que indica las aristas que lo componen. La ventaja de este modelo es que no hace falta eliminar *subtours*, pero se necesitan muchas restricciones para que el vector represente correctamente un ciclo.

Coeficientes:

Q: cantidad máxima de repeticiones de ciclos.

Variables:

$c_k^j \in \{0..m\}$: variable entera. Representa a los ciclos en el sentido de las aristas. Es decir que indican el índice de la arista elegida en cada posición del ciclo.

$l^j \in \{0..n\}$: variable entera. Representa el largo de cada ciclo.

s_e^j: variable binaria. Indica si una arista es cuerda del ciclo.

$r^j \in \{0..Q\}$: variable entera. Repetición del ciclo CS_j.

Formulación:

$$\min \sum_{j\in\{1..J\}} \sum_{e\in E} \sum_{k\in V} costo_e * (c_k^j = e) \tag{3.67}$$

sujeto a

$$\sum_{e\in E} \sum_{i\in V} (c_k^j = e) * ((e.origen = k) + (e.destino = k)) <= 2 \quad \forall k \in V, \forall j \in \{1..J\} \tag{3.68}$$

$$(c_k^j = 0) => (l^j < k),\ \forall k \in V, \forall j \in \{1..J\} \tag{3.69}$$

$$(c_k^j \neq 0) => (l^j >= k),\ \forall k \in V, \forall j \in \{1..J\} \tag{3.70}$$

$$(l^j >= 3) \text{ OR } (l^j = 0),\ \forall j \in \{1..J\} \tag{3.71}$$

$$(c_k^j \neq 0) => (c_k^j = e1) \text{ AND } (c_{k-1}^j = e2) \text{ AND } ((e1.origen = e2.origen) \text{ OR } (e1.origen = e2.destino) \text{ OR } (e1.destino = e2.origen) \text{ OR } (e1.destino = e2.destino)) \quad \forall e1 \in E, \forall e2 \in E, \forall k \in V, \forall j \in \{1..J\} \tag{3.72}$$

$$\begin{aligned}(l^j \neq 0) => \sum_{i \in V} \sum_{e_1 \in E} \sum_{e_2 \in E / e_1 \neq e_2} (c_1^j = e_1) \\ \text{AND } (c_i^j = e_2) \text{ AND } (i = l^j) \text{ AND } ((e_1.origen = e_2.origen) \\ \text{OR } (e_1.origen = e_2.destino) \text{ OR } (e_1.destino = e_2.origen) \\ \text{OR } (e_1.destino = e_2.destino)) = 1 \\ \forall j \in \{1..J\}\end{aligned} \tag{3.73}$$

$$\begin{aligned}(1 - s_e^j) * L + s_e^j * e \neq c_k^j \\ \forall e \in E, \forall k \in V, \forall j \in \{1..J\}\end{aligned} \tag{3.74}$$

$$\begin{aligned}\sum_{i \in V} \sum_{e_2 \in E} (c_i^j = e_2) \text{ AND } ((e_1.origen = e_2.origen) \\ \text{OR } (e_1.origen = e_2.destino)) >= (s_{e_1}^j = 1) \\ \forall e_1 \in E, \forall j \in \{1..J\}\end{aligned} \tag{3.75}$$

$$\begin{aligned}\sum_{i \in V} \sum_{e_2 \in E} (c_i^j = e_2) \text{ AND } ((e_1.destino = e_2.origen) \\ \text{OR } (e_1.destino = e_2.destino)) >= (s_{e_1}^j = 1) \\ \forall e_1 \in E, \forall j \in \{1..J\}\end{aligned} \tag{3.76}$$

$$\begin{aligned}(c_1^j < c_i^j) \text{ OR } (c_i^j = 0) = 1 \\ \forall i \in V, \forall j \in \{1..J\}\end{aligned} \tag{3.77}$$

$$\begin{aligned}c_2^j <= \sum_{i \in V} (i = l^j) * c_i^j \\ \forall j \in \{1..J\}\end{aligned} \tag{3.78}$$

$$e.demanda <= (\sum_{j\in\{1..J\}}\sum_{i\in V}((e = c_i^j) * r^j)) + \sum_{j\in\{1..J\}} s_e^j * 2 * r^j \quad \forall e \in E \tag{3.79}$$

$$l^j <= l^{j+1} \quad \forall j \in \{1..J\} \tag{3.80}$$

$$\sum_{i\in V}(c_i^{j_1} \neq c_i^{j_2}) >= (l^{j_1} \neq 0) * (l^{j_2} \neq 0) \quad \forall j_1 \in \{1..J\}, \forall j_2 \in \{1..J\} \,|j_1 < j_2 \tag{3.81}$$

$$\sum_{j\in\{1..J\}} r^j <= Q \tag{3.82}$$

El primer grupo de restricciones determina la construcción de ciclos. La restricción 3.68 indica que los nodos no se pueden repetir, las restricciones 3.69, 3.70 y 3.71 son necesarias para poder definir el largo del ciclo, la 3.72 establece que cada arista debe ser adyacente con su antecesor; y la restricción 3.73 indica adyacencia cíclica, es decir, que el último es adyacente al primero.

Otro grupo de restricciones determina cuales son las cuerdas (*straddles*) del ciclo. La restricción 3.74 establece que una arista *straddle* no puede estar en los vectores anteriores si estos no son nulos. Las restricciones 3.75 y 3.76 son necesarias para obligar que la arista sea adyacente a dos aristas en cada nodo. En esta restricción e_1 es la arista que puede ser *straddle* y e_2 es la que pertenecería al ciclo.

Se prestó especial atención a la simetría que aparece naturalmente en la formulación. En un modelo de programación por restricciones, la simetría hace que la búsqueda pase

muchas veces por soluciones equivalentes. En el capítulo 10 de [65] podemos encontrar una excelente introducción a la simetría en programación por restricciones desde el punto de vista de teoría de grupos.

En esta formulación solo nos encargamos de evitar la simetría mediante restricciones. En el segundo modelo, basado en el espacio de ciclos, eliminamos la simetría desde la formulación misma, pero nos encontramos con el problema de los *subtours*.

Al expresar los ciclos en términos de aristas como un vector de variables, podemos visitar el mismo ciclo una vez por cada arista componente. En nuestro modelo evitamos esto mediante la restricción 3.77 que obliga a que la primera arista sea la de menor índice.

Por otro lado, la formulación permite que cada ciclo se represente dos veces, una por cada sentido del ciclo. Esto se evita con la restricción 3.78 que obliga a que el segundo elemento sea menor al último.

La satisfacción de la demandas para cada arista la expresamos mediante la restricción 3.79.

Dado que todas las representaciones de ciclos del 1 al J son equivalentes y el modelo tiene una variable de repetición de ciclos, para eliminar simetría podemos exigir que los ciclos estén ordenados por largo. Esto lo hacemos con la restricción 3.80. Además, por el mismo motivo, podemos obligar a que todos los ciclos sean distintos mediante la restricción 3.81.

La incorporación de las restricciones para romper la simetría debe ir acompañada de modificaciones en las fases de búsqueda correlativa a esas restricciones. De esta forma, orientamos la búsqueda para que comience por las soluciones que evitan la simetría. Por ejemplo la restricción 3.77 tiene que combinarse con una búsqueda que comience por las variables de menor valor en la formación del ciclo desde el primer elemento.

Es importante mencionar que los modelos de programación por restricciones requieren

que los dominios de las variables hayan sido cuidadosamente estudiados, de forma tal que sean lo más acotados posibles, sin excluir las posibles soluciones óptimas. Es por eso que mediante la restricción 3.82 limitamos la cantidad total de ciclos en la solución (incluidas las repeticiones) para reducir el espacio de búsqueda.

La fase de búsqueda del motor de programación por restricciones se redefinió para que seleccione en primer lugar las variables de menor tamaño de dominio y de menor valor. De las opciones permitidas por IBM ILOG CP, ésta resultó ser la más eficiente luego de incorporar las restricciones para eliminar simetrías.

Este modelo fue implementado con IBM ILOG CP, en una máquina intel core i3, con procesador de 2,20GHz y 8Gb de memoria RAM. En la sección de resultados numéricos 3.8 se pueden ver los detalles de las ejecuciones para las distintas instancias disponibles del problema. Para la instancia COST239 (instancia para la cual otros modelos encuentran el óptimo en un par de segundos), este modelo necesitó de 177 segundos para llegar a una solución de $34.720, es decir a 7,3 % del óptimo de $ 32.340.- En la sección 3.8.1 se describen las métricas de las instancias y en el apéndice B.1 se describe el problema COST239 en detalle.

3.5.4. Modelo Espacio de Ciclos

El segundo modelo utiliza el concepto de espacio de ciclos (que describimos en la subsección 3.4.3), es decir que construye los ciclos por unión disjunta (adición módulo 2, utilizamos el símbolo $\oplus$) de los ciclos fundamentales del grafo. Esta forma de describir los ciclos es excelente para eliminar la simetría, pero requiere de restricciones para eliminar los *subtours*.

Coeficientes:

Q: cantidad máxima de repeticiones de ciclos.

Variables:

$fund_l^j$: variable binaria. Indica si tomamos al ciclo fundamental l en la composición del ciclo j.

c_e^j: variable binaria. Indica si una arista pertenece al ciclo.

s_e^j: variable binaria. Indica si una arista es cuerda del ciclo.

$r^j \in \{0..Q\}$: variable entera. Repetición del ciclo CS_j.

z_k^j: variable binaria. Utilizada para eliminar *subtours*. Indica el nodo que tomamos como referencia para el conteo de los nodos del ciclo.

$u_k^j \in \{0..n\}$: variable entera. Utilizada para eliminar *subtours*. Representa el conteo de nodos.

Formulación:

$$\min \sum_{j\in\{1..J\}} \sum_{e\in E} costo_e * r^j * c_e^j \tag{3.83}$$

sujeto a

$$\begin{aligned} c_e^j &= \sum_{l\in C} fund_l^j \oplus 2 \\ \forall e \in E, \forall j \in \{1..J\} \end{aligned} \tag{3.84}$$

$$\begin{aligned} c_e^j + s_e^j &\leq 1 \\ \forall e \in E, \forall j \in \{1..J\} \end{aligned} \tag{3.85}$$

$$s_e^j \leq \sum_{a \in E | (a.origen = e.origen \text{ OR } a.destino = e.origen) \text{ AND } a \neq e} c_a^j \quad \forall e \in E, \forall j \in \{1..J\} \tag{3.86}$$

$$s_e^j \leq \sum_{a \in E | (a.origen = e.destino \text{ OR } a.destino = e.destino) \text{ AND } a \neq e} c_a^j \quad \forall e \in E, \forall j \in \{1..J\} \tag{3.87}$$

$$\sum_{k \in V} (u_k^j > 0) = \sum_{e \in E} c_e^j \quad \forall j \in \{1..J\} \tag{3.88}$$

$$\sum_{k \in V} z_k^j = 1 \quad \forall j \in \{1..J\} \tag{3.89}$$

$$z_k^j \leq u_k^j \quad \forall k \in V, \forall j \in \{1..J\} \tag{3.90}$$

$$u_k^j \leq \sum_{e \in E} c_e^j \quad \forall k \in V, \forall j \in \{1..J\} \tag{3.91}$$

$$u_{k_1}^j \neq 0 \Longrightarrow u_{k_1}^j \neq u_{k_2}^j \quad \forall k_1, k_2 \in V | k_1 \neq k_2, \forall j \in \{1..J\} \tag{3.92}$$

$$\begin{aligned}(c_e^j \neq 0) \Longrightarrow \\ ((u_{e.origen}^j + 1 = u_{e.destino}^j) \text{ OR} \\ (u_{e.origen}^j - 1 = u_{e.destino}^j) \text{ OR} \\ (z_{e.origen}^j = 1) \text{ OR } (z_{e.destino}^j = 1)) \geq 1 \\ \forall e \in E, \forall j \in \{1..J\}\end{aligned} \tag{3.93}$$

$$\begin{aligned}\sum_{e \in E | e.origen = k \text{ OR } e.destino = k} c_e^j \leq 2 \\ \forall k \in V, \forall j \in \{1..J\}\end{aligned} \tag{3.94}$$

$$\begin{aligned}e.demanda <= \sum_{j \in \{1..J\}} r^j * (c_e^j + s_e^j * 2) \\ \forall e \in E\end{aligned} \tag{3.95}$$

$$\sum_{j \in \{1..J\}} r^j <= Q \tag{3.96}$$

Llamamos *circs* a la unión disjunta de ciclos, sean o no ciclos. La restricción 3.84 establece la relación entre los ciclos fundamentales y los *circs* creados.

Una arista no puede ser cuerda y parte del ciclo al mismo tiempo, esto es expresado en la restricción 3.85. Además, para ser una cuerda necesitamos que sea adyacente a nodos del ciclo. Esto es lo que implican las restricciones 3.86 y 3.87.

Para que no queden *subtours* utilizamos restricciones inspiradas en la formulación MTZ, que etiqueta los nodos del ciclo de forma secuencial, habiendo elegido un nodo inicial. Si bien esta formulación no es buena en PLE porque su relajación no es ajustada, en

programación por restricciones esto no nos preocupa, porque solo tenemos que ver el espacio de búsqueda y su simetría. La restricción 3.88 implica que solo utilizamos la variable u_k^j para contar nodos en el ciclo, al igual que la restricción 3.91. La restricción 3.89 determina que podemos elegir un solo nodo inicial, el cual, a su vez, tiene que estar en el ciclo 3.90. Las restricciones 3.92 y 3.93 son equivalentes a la formulación MTZ.

Por otro lado, obligamos a que los grados de los nodos no sean mayores a 2 mediante la restricción 3.94 para achicar el espacio de búsqueda.

Al igual que en el modelo anterior, necesitamos cubrir las demandas 3.95 y acotar la cantidad total de ciclos 3.96.

Este modelo tiene una eficiencia similar al anterior. Para el problema COST239 (instancia para la cual otros modelos encuentran el óptimo en un par de segundos), este modelo necesitó de 122 segundos para llegar a una solución de \$35.020, es decir a 8,3 % del óptimo de \$ 32.340.-

Adicionalmente, este modelo admite restricciones válidas que permiten acotar el espacio de búsqueda y el dominio de las variables pero no generan una mejora de la eficiencia en los experimentos que detallamos en la sección 3.8.

$$u_k^j \leq \sum_{e \in E | e.desde=k \text{ OR } e.hasta=k} c_e^j * N \qquad \forall k \in V, \forall j \in \{1..J\} \tag{3.97}$$

$$r^j = 0 \Longrightarrow \sum_{e \in E} c_e^j + s_e^j + \sum_{k \in V} u_k^j + z_k^j = 0 \qquad \forall j \in \{1..J\} \tag{3.98}$$

$$r^j \neq 0 \Longrightarrow \sum_{k \in V} z_k^j = 1 \quad \forall j \in \{1..J\} \tag{3.99}$$

$$z_k^j = 1 \Longrightarrow u_k^j = 1 \quad \forall k \in V, \forall j \in \{1..J\} \tag{3.100}$$

La restricción 3.97 exige que únicamente las variables u de nodos que pertenecen al ciclo tomen valores no nulos.

La restricción 3.98 implica que la variable r^j determina cuando no crear un ciclo. Se combina con que exige cambiar la restricción 3.89 por la restricción 3.99

Por la restricción 3.100 la variable u tiene que comenzar siempre en el nodo que indica la variable z.

3.6. Algoritmo GRASP para el problema SCA

3.6.1. Motivación

Un algoritmo de generación de columnas, como el que veremos en la sección siguiente, requiere partir de un conjunto de columnas que garanticen una solución inicial factible. Esto se puede lograr con los modelos descriptos anteriormente. Sin embargo, también es importante contar con una cantidad reducida de buenos ciclos, que a su vez permitan formar buenas soluciones.

GRASP es una metaheurística para problemas de optimización combinatoria del tipo iterativo (o multi-inicio), en la cual cada iteración consta básicamente de dos fases: construcción y búsqueda local. La fase de construcción se encarga de crear una solución factible, cuya vecindad es investigada hasta encontrar un óptimo local mediante la fase de búsqueda. En principio, no hay relación entre las iteraciones, solo se retiene la mejor solución encontrada.

En la próxima subsección describiremos brevemente las principales características de GRASP; para una explicación más detallada recomendamos ver Resende *et al.* [63] y el libro de González [24].

GRASP provee fácilmente un conjunto de buenos ciclos para darlos como *pool* inicial al algoritmo de generación de columnas.

Inicialmente desarrollamos un algoritmo GRASP para el problema SCA. Éste partía del conjunto de todos los ciclos del grafo que habían sido enumerados con el algoritmo de Johnson [39]. La fase constructiva consistía en incorporar ciclos a la solución parcial en base al indicador de *eficiencia real*. La fase de búsqueda local consistía en una metaheurística *Tabu Search*.

En 2013, Delgadillo [15] desarrolla otro algoritmo GRASP partiendo de un conjunto

acotado de ciclos candidatos. La fase de construcción implementada es una adaptación de la heurística CIDA introducida por Grover *et al.* en [64]. La búsqueda local reemplaza ciclos de la solución por otros que se encuentren en el conjunto de candidatos, siguiendo el criterio de la eficiencia real.

En la sección 3.6.4 describiremos una tercera implementación GRASP que desarrollamos para dar buenos ciclos iniciales a nuestro algoritmo *Branch-and-Price*. En la próxima sección daremos una breve descripción de la metaheurística GRASP.

3.6.2. GRASP

El pseudocódigo que se muestra a continuación describe los bloques principales de GRASP para el caso de minimización, en el cual se ejecutan un máximo de iteraciones (*Max_Iteraciones*). *Semilla* es utilizada para generar números pseudoaleatorios.

```
1 procedure GRASP(Max_Iteraciones, Semilla)
2   Leer_datos();
3   for k ← 1, ... , Max_Iteraciones do
4     Solución ⟵ Fase_Constructiva_Aleatorizada(Semilla);
5     Solución ⟵ Búsqueda_Local(Solución);
6     Actualizar_Solución(Solución, Mejor_Solución);
7   end;
8   return Mejor_Solución;
9 end GRASP.
```

En cada iteración se construye una lista de candidatos, ordenados de manera decreciente con respecto a su beneficio. Una heurística avara convencional simplemente seleccionaría el elemento que se encuentra primero en la lista, es decir, el mejor de todos. Aún cuando se espera obtener una buena solución con este procedimiento, generalmente no se obtendrá la óptima.

Una estrategia ligeramente distinta puede producir resultados diferentes y probablemente mejores. En el caso de GRASP, tal estrategia constituye la componente aleatoria del método. Así, en vez de seleccionar el mejor elemento en cada paso, se toma uno al azar de entre una *lista restringida de candidatos* (LRC), cuyas limitaciones impuestas aseguran que se elija un elemento bueno, aunque no necesariamente óptimo. La cantidad de elementos de LRC puede ser limitida por cardinalidad o por calidad.

La fase constructiva consiste en avanzar iterativamente hacia una solución factible, añadiendo un elemento cada vez. Esto implica visualizar cada solución como una estructura que puede obtenerse mediante incrementos de una base (generalmente vacía al inicio), donde cada nuevo elemento se elige de un conjunto de posibles candidatos. Para este fin, se debe contar con una forma de evaluar la bondad de añadir cada componente específico, cuidando además que la subestructura resultante forme parte de una solución factible.

Una vez elegido el elemento, se aumenta la solución parcial y se procede a recalcular los beneficios asociados a cada elemento. En este proceso resulta particularmente importante reflejar los cambios producidos al incorporar la selección previa como una componente de la solución parcial. En esto consiste el aspecto adaptativo de GRASP.

```
1  procedure Fase_Constructiva_Aleatorizada(Semilla)
2    Solución ⟵ ∅;
3    Evaluar los costos incrementales de los elementos candidatos;
4    while Solución incompleta do
5      Construir la LRC;
6      Seleccionar un elemento s de la LRC de forma aleatoria;
7      Solución ⟵ Solución ∪{s};
8      Reevaluar los costos incrementales;
9    end;
10   return Solución;
11 end Fase_Constructiva_Aleatorizada.
```

La fase constructiva no garantiza optimalidad local, incluso con respecto a entornos definidos de forma tan simple como un intercambio de pares de elementos. Por lo tanto, es importante utilizar un procedimiento de postproceso para mejorar la solución obtenida en la fase de construcción. La idea que apoya esto es que GRASP produce una diversidad de buenas soluciones, esperando que al menos una de ellas esté en un entorno de la solución óptima, o al menos en el de una solución muy cercana al valor óptimo.

Existen varias opciones para llevar a cabo este postprocesamiento. La forma más sencilla es limitar la búsqueda a una *k-vecindad*. Dado la variable x que representa una solución del problema, se llama *k-vecindad* a todas las soluciones y que son distintas a x en, a lo sumo, k elementos. Un procedimiento de *k-intercambio* consiste en cambiar a lo sumo k elementos de una solución en particular. Cuando k es igual a la cantidad total de elementos de la solución, estamos haciendo una búsqueda exhaustiva.

```
1 procedure Búsqueda_Local(Solución)
2   while Solución no es óptimo local do
3     Encontrar s' ∈ N(Solución) tal que f(s') < f(Solución);
4     Solución ⟵ s';
5   end;
6   return Solución;
7 end Búsqueda_Local.
```

GRASP puede ser visto como una técnica de muestreo repetitiva. Cada iteración produce una solución muestra de una distribución desconocida, cuya media y varianza son funciones de la naturaleza restrictiva de la LRC. Por ejemplo, si la LRC es restringida a un único elemento, entonces la misma solución será producida en todas las iteraciones. La varianza de la distribución será nula y la media será igual al valor de la solución avara. Si a la LRC se le permite tener más elementos, entonces se producirán muchas soluciones diferentes, implicando una varianza mayor. Dado que la avaricia juega un rol menor en este caso, el valor de la media de las soluciones podría ser peor. Sin embargo, el valor de

la mejor solución encontrada supera el valor de la media y frecuentemente es óptima.

Un defecto del GRASP es la independencia de sus iteraciones, es decir, el hecho de no aprender de la historia de soluciones encontradas en las iteraciones anteriores. Esto se debe a que el algoritmo básico descarta información de cualquier solución encontrada que no mejore la solución actual. La información recolectada de buenas soluciones puede ser usada para implementar procedimientos *basados en memoria* para influenciar la fase constructiva, modificando las probabilidades de selección asociadas a cada elemento de la LRC.

3.6.3. Heurísticas para obtener buenos p-ciclos

En principio, dado que una solución de SCA está conformada por p-ciclos, podría pensarse en un esquema GRASP cuya fase constructiva los incorpore a la solución parcial. Pero para eso es necesario disponer, ya sea de algunos de ellos o de todos. Como ya vimos, disponer de todos los p-ciclos puede ser una tarea titánica, ya que la cantidad crece exponencialmente con el tamaño del grafo. Es por este motivo que necesitamos heurísticas para obtener buenos p-ciclos.

En 2002 Zhang *et al.* [87] proponen una heurística llamada *Straddling Link Algorithm* (SLA) con la intención de enumerar un conjunto muy pequeño de los p-ciclos posibles de la red. Para cada arista del grafo, SLA llama dos veces al algoritmo de Dijkstra para encontrar el camino más corto y el siguiente camino disjunto más corto que conecte los dos nodos de la arista original y sin utilizarla. Un ciclo es formado uniendo los dos caminos. Así, la arista elegida se convierte en una cuerda del ciclo resultante. Este algoritmo es muy rápido ya que solo hay, a lo sumo, $|E|$ p-ciclos de este tipo. Los ciclos resultantes son de baja calidad ya que son muy pequeños y poseen pocas cuerdas.

En 2003, Doucette *et al.* [17], basándose en *SLA*, propusieron tres nuevas heurísticas, llamadas *SP-add*, *Expand* y *Grow*.

La heurística *SP-add* parte de un ciclo existente, llamado ciclo primario. Para cada una de sus aristas, un nuevo ciclo es creado reemplazándolas por el camino más corto disjunto en el sentido de los nodos que conecten los dos nodos de la arista original (sin utilizarla). Al igual que en el caso anterior, la arista original se convierte en una cuerda del ciclo.

El algoritmo *Expand* es una modificación de *SP-add*, en el cual, una vez que una arista del ciclo fue convertida en cuerda, pasamos a la siguiente arista para intentar convertirla también en cuerda. Con esta modificación los ciclos resultantes son más grandes y tienen más cuerdas que los ciclos obtenidos por *SP-add*.

A su vez, el algoritmo *Grow* es una modificación de *Expand*, y consiste en llamar recursivamente a *Expand* cada vez que un ciclo es encontrado.

El algoritmo Grow es el que obtiene una mayor «eficiencia a priori» (AE) promedio, aunque también es el que devuelve un conjunto de ciclos más numeroso. En 2013 Delgadillo [15] muestra que, asumiendo que empieza con, a lo sumo, $|E|$ ciclos, devuelve, a lo sumo, $|E|^2 |V|$ ciclos.

En este tipo de heurísticas es importante no generar p-ciclos repetidos para no afectar la eficiencia global.

El indicador de «eficiencia para una instancia» presentado en 2.4 nos da una idea de la eficiencia de un p-ciclo para las demandas que restan por cubrir (dentro de una solución parcial) en una instancia en particular y es utilizado como base para un algoritmo avaro iterativo llamado CIDA desarrollado por Grover *et al.* [64]. Este algoritmo parte de un conjunto de ciclos que son evaluados utilizando el indicador antes mencionado. Dicho indicador es recalculado en cada iteración.

En 2004, Liu *et al.* [45] propusieron una heurística para seleccionar ciclos a partir de una búsqueda DFS. Se tiene en cuenta la eficiencia a priori y se incluyen los ciclos más cortos que protegen a cada eje. Los resultados reportados muestran un mejor desempeño de este algoritmo sobre Grow.

```
1 procedure CIDA
2   inicializar demandaRestante, Solución, ConjuntoDeCiclos
3   ConjuntoDeCiclos ⟵ EnumerarCiclos()
4   while demandaRestante[i] > 0 para alguna arista i do
5     MejorCiclo ⟵ 0
6     for each ciclo p en ConjuntoDeCiclos do
7       Calcular Ep
8       if Ep > EMejorCiclo then
9         MejorCiclo ⟵ p
10        EMejorCiclo ⟵ Ep
11      end if
12    end for
13    Solución[MejorCiclo] ⟵ Solución[MejorCiclo] + 1
14    for each arista on−cycle i de MejorCiclo do
15      demandaRestante[i] ⟵ demandaRestante[i] − 1
16    end for
17    for each arista straddle i de MejorCiclo do
18      demandaRestante[i] ⟵ demandaRestante[i] − 2
19    end for
20  end while
21 end procedure
```

3.6.4. Algoritmo GRASP propuesto

Nosotros proponemos una heurística que llamamos *Fundamentales - Expansión - Combinación* (FEC). Esta heurística es bastante más compleja que las anteriores y posee más parámetros. El algoritmo FEC parte de un conjunto de ciclos arbitrario; en las sucesivas iteraciones incorporamos los ciclos obtenidos en la búsqueda local a este conjunto que es

ampliado mediante combinaciones de sus elementos, generando ciclos que dominan algún ciclo de él mismo. La LRC es construida a partir de dicho conjunto. Es decir que construimos una solución tomando elementos de forma avara y aleatoria de la LRC. Finalmente, en la fase de búsqueda local (definida como un 1-intercambio) se obtienen los mejores reemplazos de cada ciclo de la solución mediante un modelo PLEM. A continuación desarrollamos los detalles de cada etapa.

3.6.4.1. Heurística *Fundamentales - Expansión - Combinación*, FEC

La fase constructiva incorpora de a un ciclo por vez a la solución parcial. Este ciclo es obtenido llamando al algoritmo FEC que es adaptativo y aleatorio.

El algoritmo FEC parte de un conjunto de ciclos; en nuestra implementación elegimos arbitrariamente ciclos fundamentales (base del espacio de ciclos que describimos en la subsección 3.4.3). Esta elección se debe a dos razones: la primera es que un conjunto de ciclos fundamentales cubre, por definición, todo el grafo; la segunda razón es que, cambiando el nodo inicial y el orden de preferencia de la próxima arista del algoritmo de generación del árbol de cobertura, se generan distintos conjuntos de ciclos fundamentales. Esto es un factor más que contribuye a la aleatorización del algoritmo.

El primer paso consiste en construir un ranking de aristas en función del máximo ahorro que aportaría si ésta fuese una cuerda del ciclo (*straddle*) teniendo en cuenta la demanda restante a cubrir. Es decir, consideramos dos veces el costo de esa arista si hay dos o más demandas restantes en ella, una vez el costo si solo queda una unidad de demanda y cero en otro caso.

Para obtener ciclos fundamentales, necesitamos determinar un árbol de cobertura mínimo. El mismo es calculado mediante el algoritmo de Prim, como criterio de minimización utilizamos los ahorros calculados en el paso anterior. Adicionalmente, el nodo inicial del árbol es un nodo distinto en cada iteración.

En algunos casos, los ciclos iniciales pueden tener muchas cuerdas y por lo tanto nos conviene minimizar el ahorro del árbol de cobertura. Luego de la generación de los ciclos fundamentales, buscamos ciclos que los dominen, convirtiendo en cuerdas las aristas del ciclo y evaluando su conveniencia. En estos casos conviene que el árbol de cobertura sea de ahorro máximo, porque esas aristas se convierten en cuerdas en el paso posterior. Es por este motivo que generamos dos veces ciclos fundamentales en cada llamada a FEC, una basada en un árbol de cobertura de ahorro mínimo y otra basada en un árbol de cobertura de ahorro máximo.

Al conjunto de ciclos obtenidos hasta ese momento se le incorporan los ciclos obtenidos en las búsquedas locales de la iteraciones anteriores y, finalmente, se le aplica un procedimiento que hace uso de la proposición 3.2.1. Es decir que, dados dos ciclos, verifica si estos comparten una única arista y ningún otro nodo además de los dos nodos de dicha arista. En ese caso se genera un tercer ciclo por combinación de los dos anteriores. Este procedimiento genera ciclos que, a su vez, pueden ser comparados con todos los del conjunto. La cantidad de ciclos totales en el conjunto se limita en la implementación mediante un parámetro.

El próximo paso consiste en elegir el mejor ciclo del conjunto e incorporarlo a la solución parcial. Para esto determinamos las demandas restantes que cubre cada ciclo, su costo y su eficiencia. La elección es aleatoria; no se elige el mejor ciclo (en términos de eficiencia), sino que se toma uno al azar de entre una lista restringida de candidatos (LRC), cuyas limitaciones impuestas aseguran que se elija un elemento bueno, aunque no necesariamente el mejor.

Finalmente, el ciclo elegido es analizado en busca de otro posible ciclo que lo domine; en ese caso se incorpora el ciclo dominante en lugar del original. Este análisis se limita a buscar dominaciones en el sentido de la proposición 3.2.1.

```
1 procedure FEC(Semilla)
2   inicializar demandaRestante, Solución
3   while demandaRestante[i] > 0 para alguna arista i do
4     inicializar Pool
5     ranking de aristas
6     ranking de nodos
7     árbol de cobertura
8     ciclos fundamentales
9     Pool ⟵ Fundamentales ∪ Combinación
10              ∪ ResultadosBúsquedaLocal
11    MejorCiclo ⟵ 0
12    Elección aleatorizada de MejorCiclo de LRC
13    Búsqueda de dominancia sobre MejorCiclo
14    Solucion[MejorCiclo] ⟵ Solución[MejorCiclo] + 1
15    for each arista on−cycle i de MejorCiclo do
16      demandaRestante[i] ⟵ demandaRestante[i] − 1
17    end for
18    for each arista straddle i de MejorCiclo do
19      demandaRestante[i] ⟵ demandaRestante[i] − 2
20    end for
21  end while
22 end procedure
```

3.6.5. Búsqueda local

Si a la fase de búsqueda local de nuestro algoritmo GRASP la definimos como k-intercambio, es decir respecto de una vecindad que difiere en k ciclos (sin considerar sus repeticiones), entonces los modelos constructivos de la sección 3.4 y la sección 3.5 pueden

obtener los ciclos óptimos faltantes. Cuanto menor sea el parámetro k, menor la simetría del modelo contructivo y mayor su eficiencia.

En nuestra implementación, una vez obtenida una solución completa en la fase constructiva, el algoritmo GRASP implementado pasa a la fase de búsqueda local, definida como 1-intercambio. Es decir que se obtienen los mejores reemplazos de cada ciclo de la solución; esto se hace mediante un modelo PLEM. Dicho modelo es una adaptación del modelo propuesto por Wu *et al.* [84], que resultó ser el de mejor eficiencia de todos los modelos de la sección 3.4 que fueron implementados en CPLEX, utilizando su algoritmo *Branch-and-Cut*. Los resultados experimentales se muestran en la sección 3.8.

Como ya mencionamos, los modelos constructivos propuestos tienen una gran simetría en el espacio de búsqueda por tener que generar una cantidad J de ciclos. Así, durante la búsqueda, los ciclos generados para un CS_j dado pueden resultar exactamente iguales a los de otro $CS_{j'}$. Por el contrario, en la búsqueda local, solo se necesita generar un único ciclo. Esto mejora drásticamente la eficiencia de estos modelos ya que podemos evitar la mayor parte de la simetría.

```
1 procedure Búsqueda_Local(Solución)
2   for each ciclo en Solución
3     determinar demandas insatisfechas si se saca ciclo
4     nuevoCiclo ⟵ mejor ciclo posible para cubrir esas demandas.
5     if nuevoCiclo no nulo
6       Solución[ciclo] ⟵ 0
7       Solución[nuevoCiclo] ⟵ Solución[nuevoCiclo] + 1
8       Pool ⟵ Pool ∪ nuevoCiclo
9     end if
10  end for
11 end Búsqueda_Local.
```

La efectividad de la búsqueda local depende de varios aspectos, tales como la estructura de la vecindad, la técnica de búsqueda sobre ella, la evaluación rápida de la función de costos de los vecinos y de la propia solución inicial.

La adaptación de los modelos de PLEM antes vistos consiste en fijar $J = 1$ y tomar como demandas del grafo únicamente aquellas que no son cubiertas por la solución parcial (solución inicial sin el ciclo en análisis). Es decir que la demanda de la arista e, llamada d_e, es solo aquella demanda no cubierta por el resto de los ciclos de la solución. A continuación detallamos la formulación del modelo de la subsección 3.4.3 adaptado a la búsqueda local de GRASP.

Coeficientes:

$\alpha = 1/|V|$: coeficiente para garantizar voltajes compatibles.

B: matriz de ciclos fundamentales del grafo G (μ x m).

C: conjunto de ciclos fundamentales con respecto a un árbol arbitrario.

L: número muy grande.

M: matriz de incidencia del grafo G (n x m).

M^t: matriz de incidencia transpuesta del grafo G (m x n).

Variables:

c_i: variable binaria. Toma valor 1 si el ciclo es generado por el ciclo fundamental i.

p_e: variable entera. Es una variable auxiliar que indica cuántos pares de ciclos fundamentales son seleccionados para generar el ciclo que incluyen a la arista e.

x_{eo}: variable binaria. Toma valor 1 si la arista e está en el ciclo generado. Los arcos son dirigidos y el índice o indica el sentido.

x_o: vector de variables x_{eo} de tamaño m.

x: suma de vectores de variables x_{eo}. Es decir, $x = x_0 + x_1$.

y_e: variable binaria. Toma valor 1 si la arista e es una cuerda (*straddle*) del ciclo generado.

z_k: variable binaria. Solo una de ellas toma valor 1 en cada ciclo generado para indicar el nodo raíz de los voltajes.

$volt_k$: variable continua. Indica el voltaje de cada nodo.

Formulación:

$$\min \sum_{e \in E} costo_e(x_{e0} + x_{e1}) \tag{3.101}$$

sujeto a

$$\begin{aligned}
B.c &= 2.p + x && (3.102)\\
x_{e0} + x_{e1} &\leq 1, \forall e \in E && (3.103)\\
\sum_{e=uv \in E} (x_{uv0} - x_{uv1}) &= 0, \forall u \in V && (3.104)\\
Mx &\leq 2 && (3.105)\\
\frac{1}{2}(x_0 + x_1)(\frac{1}{2}M^tM - 2I) &\geq y && (3.106)\\
d_e &\leq x_e + 2.y_e, \forall e \in E && (3.107)\\
\sum_{k \in V} z_k &= 1 && (3.108)\\
volt_h - volt_d &\geq \alpha x_{eo} - (1 - x_{eo}) - Lz_h && (3.109)\\
& \forall e = (d,h) \in E, \forall o \in 0,1 && (3.110)\\
& c_i \in \{0,1\}, \forall i \in C && (3.111)\\
& p_e \in \{0,\infty), \forall e \in E && (3.112)\\
& x_e \in \{0,1\}, \forall e \in E, o \in \{0,1\} && (3.113)\\
& y_e \in \{0,1\}, \forall e \in E && (3.114)\\
& z_k \in \{0,1\}, \forall k \in V && (3.115)\\
& volt_k \in \{0,\infty), \forall k \in V && (3.116)
\end{aligned}$$

3.7. Algoritmo *Branch-and-Price* propuesto para SCA

3.7.1. Motivación

El modelo más simple e intuitivo para representar el problema SCA es el descripto por Grover *et al.* en el capítulo 16 del libro *Handbook of Optimization in Telecommunications* [64] y que mostramos en la sección 3.3 y en la subsección 4.6.3 de este capítulo. En este modelo cada variable representa la cantidad de veces que se utiliza un ciclo del grafo. Por lo tanto, se trata de una formulación con una cantidad exponencial de variables. Para instancias pequeñas del problema (en cantidad de ciclos), el modelo deriva en algoritmos muy eficientes, ya que las relajaciones son muy buenas.

Un aspecto importante del problema es que las soluciones óptimas involucran a muy pocos ciclos en relación con la cantidad de ciclos totales. En las instancias del presente trabajo, las soluciones óptimas nunca necesitaron más de 24 ciclos. Si limitamos la cantidad de ciclos a considerar de manera adecuada, el problema se puede resolver más rápido sin perder calidad. En Doucette *et al.* [18] los autores informan haber probado el modelo de la sección 3.3 para la red COST_239, la cual tiene 3531 ciclos con los 250 ciclos mejor rankeados según el indicador *apriori efficiency* (AE) lo que produjo resultados en el orden del 1 % del óptimo en una milésima del tiempo respecto del problema completo.

En la literatura encontramos ejemplos de la utilización de algoritmos de *Branch-and-Price* para resolver problemas de diseño de redes de anillos, como describen Henningsson *et al.* en el libro [64] capítulo 12.

Estos hechos nos llevaron a preguntarnos qué eficiencia podríamos lograr resolviendo el modelo de Grover con un algoritmo de *Branch-and-Price*.

3.7.2. Bibliografía

A continuación mencionamos trabajos donde se proponen algoritmos *Branch-and-Price* para problemas similares al SCA. A diferencia del SCA, en el problema *Joint Capacity Allocation* (JCA) el ruteo es parte del problema. Este problema puede resultar poco realista desde el punto de vista de telecomunicaciones, ya que el diseño de la red es de carácter estratégico mientras que el ruteo es de carácter operativo. Aun así, varios autores encararon métodos para intentar resolverlo. En 2002 Rajan y Atamtürk [60] desarrollaron un método de generación de columnas heurístico y fueron capaces de producir resultados eficientes para redes con grafos densos de hasta 70 nodos. Ellos utilizan un concepto que llaman *cycle-slack* para formular el modelo, las ramificaciones y el problema de *pricing*. Cada *cycle-slack* representa la cantidad de demanda ruteada en un ciclo determinado.

En 2004 Stidsen y Thomadsen [80] también desarrollaron un método de generación de columnas heurístico para el problema JCA. El cuello de botella de este método resultó ser el algoritmo de *pricing* que genera los ciclos a incorporar al problema. Vale aclarar que el método que propusieron tiene dos subproblemas de *pricing*, uno para los ciclos y otro para los caminos.

En 2012 Jaumard *et al.* [36], trabajaron centrándose en las fallas de nodos. Para eso analizaron el problema FIPP y un nuevo tipo de p-ciclos llamados p-ciclos de nodos. Volveremos sobre este trabajo en la sección 4.6 de *Branch-and-Price* para FIPP.

3.7.3. *Branch-and-Price*

A continuación describiremos brevemente las principales características de los algoritmos *Branch-and-Price*. Para una explicación mas detallada recomendamos ver Lübbecke y Desrosiers [47], Barnhart *et al.* [9], Wolsey [83] o Salazar González [71].

Un método de *Branch-and-Price* se basa en valorizar implícitamente las variables no

básicas para generar nuevas columnas o para probar la optimalidad en un nodo del árbol de *Branch-and-Bound*, si no hay más columnas a agregar y la solución no es entera entonces se ramifica.

A continuación transcribiremos de Barnhart *et al.* [9] una explicación breve del funcionamiento de los métodos *Branch-and-Cut* y *Branch-and-Price*.

Dada una formulación y clases de desigualdades válidas, preferentemente facetas del convexo de soluciones factibles, la idea básica de un algoritmo de *Branch-and-Cut* es dejar de lado gran parte de las restricciones en la relajación lineal ya que hay demasiadas como para que sean manipuladas de forma eficiente. Además, la mayoría de ellas no son necesarias en la solución óptima. Entonces, si una solución óptima del problema relajado resulta incompatible, se llama a un sub-problema, llamado problema de separación, el cual es resuelto para intentar identificar restricciones violadas. Si encontramos una o más restricciones violadas, agregamos algunas de ellas al programa lineal para cortar la solución no factible. Luego reoptimizamos el programa lineal. Cuando no encontramos restricciones violadas es cuando debemos ramificar. El método de *Branch-and-Cut* es una generalización de *Branch-and-Bound*, pero con relajaciones que permiten que la separación y corte sean aplicadas através del árbol de *Branch-and-Bound*.

Un solución exitosa de un programa entero mixto requiere de una formulación cuyas relajaciones lineales den una buena aproximación del convexo de soluciones factibles.

La filosofía de generación de columnas es similar a la de *Branch-and-Cut*, excepto que el procedimiento se enfoca en la generación de columnas en lugar de enfocarse en generación de filas. De hecho, valorizar (*pricing*) y cortar son procedimientos complementarios para ajustar una relajación lineal.

En el esquema de generación de columnas, muchas son dejadas de lado en la relajación lineal porque hay demasiadas como para que sean manipuladas eficientemente y, de todas maneras, la mayoría de ellas va a tener su variable asociada nula en la solución óptima.

Entonces, para verificar la optimalidad de un programa lineal, se resuelve un subproblema, llamado problema de *pricing*, que permite identificar columnas que puedan entrar a la base y mejorar la función objetivo. Si esa clase de columnas se encuentra con el problema de *pricing*, el programa lineal es reoptimizado. La ramificación ocurre cuando no se encuentran más de esas columnas y la solución no satisface las condiciones de integralidad u otras.

Branch-and-Price es una generalización de *Branch-and-Bound* ya que se aplica la generación de columnas en el árbol de *Branch-and-Bound*.

En los esquemas de *Branch-and-Price* típicamente se parte de un modelo que pueda tener problemas de simetría o en los que sus soluciones relajadas están muy lejos del óptimo. Entonces, el modelo es reformulado utilizando la descomposición Dantzig-Wolfe. Debido a esto, el modelo resultante suele tener muchas columnas, motivo por el cual el esquema de *Branch-and-Price* es elegido. En nuestro caso, el modelo original tiene una cantidad exponencial de columnas.

En el desarrollo de un esquema de generación de columnas pueden surgir varias dificultades, entre ellas las dos siguientes:

- La ramificación tradicional de programación entera puede no ser efectiva porque la fijación de variables puede destruir la estructura del problema de *pricing*.
- Resolver los problemas lineales y los subproblemas hasta la optimalidad puede no ser eficiente, en cuyo caso se pueden aplicar reglas para administrar el árbol.

Dado que los modelos que se resuelven por generación de columnas suelen tener una cantidad enorme de columnas, es necesario trabajar con versiones restringidas del modelo que solo contengan un subconjunto de estas y generar adicionales solo cuando se necesite. Al problema se lo llama problema maestro (PM) y a su versión restringida se lo llama problema maestro restringido (PMR).

El problema de *pricing* identifica una columna con costo reducido mínimo. Si el costo

reducido mínimo es menor a cero, hemos identificado una columna para que entre a la base; si el costo reducido mínimo es mayor o igual a cero, hemos probado que la solución actual del PMR es también óptima del problema maestro.

La solución de una relajación lineal resuelta por generación de columnas no es necesariamente entera y aplicar el procedimiento estándar de *Branch-and-Bound* al problema restringido con las columnas existentes no garantiza una solución óptima (o factible). Después de ramificar, puede existir el caso en que una columna que sea elegida por el modelo de *pricing* no esté presente en el problema maestro. Por lo tanto, para encontrar una solución óptima debemos generar columnas después de ramificar.

Para iniciar el esquema de generación de columnas se debe proveer un problema maestro restringido. Este PMR inicial debe tener una relajación factible para asegurar que dispongamos de información dual correcta para pasarle al problema de *pricing*.

El componente de cálculo más intensivo del esquema de generación de columnas es la resolución de los programas lineales, lo que incluye la solución de muchos problemas de *pricing*.

En un programa lineal de maximización, cualquier columna con costo reducido positivo es un candidato a entrar a la base; el problema de *pricing* consiste en encontrar una columna con el costo reducido más alto. De esta manera, si una columna con costo reducido positivo existe, siempre la va a identificar. Esto garantiza que se encontrará una solución óptima al programa lineal. Sin embargo, no es necesario seleccionar la columna con el costo reducido más alto, cualquier columna con costo reducido positivo sirve. Utilizando esta observación podemos mejorar la eficiencia global cuando el problema de *pricing* es computacionalmente intensivo.

Varios esquemas de generación de columnas pueden ser desarrollados usando algoritmos heurísticos para resolver el problema de *pricing*. Para garantizar la optimalidad, se aplica un enfoque de dos fases. Mientras que el algoritmo heurístico produzca columnas con

costo reducido positivo, esa columna será agregada al problema maestro restringido. Si el algoritmo heurístico falla en producirla, entonces se llama a un algoritmo de *pricing* exacto. Este esquema reduce el tiempo de cada iteración. Sin embargo, el número de iteraciones puede incrementarse y no es seguro que el efecto global sea positivo.

Durante el proceso de resolución del esquema de generación de columnas, el problema restringido maestro crece. Puede ser recomendable eliminar columnas no básicas con costo reducido muy negativo para reducir el tiempo por iteración.

Todas estas ideas pueden ser combinadas en el siguiente esquema general:

1 **procedure** Generación_de_Columnas
2 Determinar una solución inicial factible del PMR.
3 Inicializar el pool de columnas como vacío.
4 Resolver el PMR actual.
5 Eliminar columnas no básicas con costo reducido muy negativo del PMR.
6 Si el pool de columnas todavía contiene columnas con costo reducido positivo, seleccionar un subconjunto de ellas y agregarlos al PMR e ir al paso 4.
7 Vaciar el pool de columnas.
8 Llamar a la heurística de pricing para generar una o más columnas con costo reducido positivo. Si se generan columnas, agregarlas al pool e ir al paso 6.
9 Llamar al algoritmo exacto de pricing para probar la optimalidad o generar una o más columnas con costo reducido positivo. Si se generan columnas, agregarlas al pool e ir al paso 6.
10 **end** Generación_de_Columnas

Dependiendo del problema de *pricing*, puede incluso ser posible generar más de una

columna con costo reducido positivo por iteración sin incrementar mucho el tiempo de cálculo. Un esquema de este tipo incrementará el tiempo por iteración dado que el problema restringido va a ser más grande, pero podría disminuir la cantidad de iteraciones.

Sol *et al.* [75] describen una heurística rápida para generar columnas con costo reducido positivo. Ellos toman columnas existentes con costo reducido nulo y les aplican mejoras locales para construir columnas con costo reducido positivo.

Es fácil calcular cotas en el programa lineal final basado en el PMR y los costos reducidos. Esto es especialmente importante para evitar una gran cantidad de iteraciones para probar la optimalidad.

3.7.4. Variantes de generación de columnas

Un esquema de generación de columnas puede ser fácilmente convertido en una heurística efectiva si encontrar buenas soluciones factibles del programa entero es muy importante y probar su optimalidad no tanto. Esto se puede lograr ramificando y buscando en el árbol en forma avara.

El esquema de generación de columnas también permite desarrollar una heurística, muy utilizada en la práctica, frente a problemas con muchas variables. Esta heurística consiste en resolver el problema restringido y relajado hasta la optimalidad (generando columnas) y, luego, con el conjunto final de columnas (es decir, el modelo sigue siendo restringido) resolverlo como un programa entero. Al final del proceso, la solución no es necesariamente óptima ya que no tenemos garantías de que esas columnas generadas sean las óptimas.

Otro posibilidad consiste en combinar generación de columnas con generación de filas. Esta opción puede generar relajaciones del programa lineal muy fuertes. Sin embargo, combinar los dos esquemas no es trivial. La principal dificultad es que el problema de *pricing* puede ser mucho más difícil luego de que se le hayan agregado filas, ya que ellas pueden destruir su estructura.

Existe otra alternativa, llamada SPRINT, que consiste en resolver subproblemas que solo tienen subconjuntos de columnas. Esto es repetido hasta que todas las columnas hayan sido consideradas. Luego, un nuevo subproblema es formado con las soluciones óptimas y las columnas con buenos costos reducidos de todos los subproblemas anteriores. Ver Anbil, Tanga y Johnson [2] o Bixby *et al.* [13].

Por lo tanto, existen varias alternativas basadas en el esquema de generación de columnas.

- **Exacta**. Corresponde a la variante clásica de la implementación de generación de columnas que garantiza la obtención del óptimo. Veremos sus detalles en las secciones siguientes.
- **SPRINT**. Descripta brevemente.
- **Heurística de generación de columnas**. Como ya mencionamos, esta heurística consiste en resolver el problema restringido y relajado hasta la optimalidad y luego, con ese conjunto de columnas resolverlo como un programa entero. Hemos implementado esta variante que ejecutamos previamente al *Branch-and-Price* exacto.

3.7.5. *Framework*

Los programas de programación entera (como por ejemplo CPLEX y Gurobi), no permiten modificar el modelo de forma directa en los nodos del árbol de búsqueda de *Branch-and-Bound*. Por lo tanto, solo se pueden utilizar como motores de PL dentro de un *framework* de generación de columnas.

Nosotros optamos por desarrollar nuestro propio *framework* de generación de columnas, pero podríamos haber utilizado alguno de los disponibles como jORLib o SCIP. Utilizamos CPLEX como librería para resolver los PL, llamándolo desde código en C++ sobre *Visual Studio*.

```
1  procedure Branch−and−Bound
2    inicializa L con el problema entero original;
3    z̃ ← ∞;
4    while L ≠ ∅ then
5      extrae un subproblema de la lista L;
6      calcula un óptimo x* de la relajación lineal, y sea z* ← c^T x*;
7      if z* < z̃ then
8        if x* entero then
9          z̃ ← z*;
10         x̃ ← x*;
11       else
12         elige h tal que x*_h ∉ ℤ;
13         añade a L el subproblema con x_h ≤ ⌊x*_h⌋;
14         añade a L el subproblema con x_h ≥ ⌈x*_h⌉;
15   end while
16 end procedure Branch−and−Bound
```

Las principales características de nuestro *framework* son las siguientes:

- Toma soluciones iniciales desde GRASP o generando un conjunto de ciclos fundamentales.
- Permite llamar a una heurística de *Generación de Columnas* antes de resolver el nodo raíz. De esta forma disponemos de una solución entera de buena calidad y, por lo tanto, de buenas cotas desde el comienzo del algoritmo.
- Un procedimiento recorre los nodos activos (que todavía no fueron resueltos) y determina una cota inferior para informar la distancia máxima al óptimo y reducir el tiempo de demostración de la optimalidad (*tailing off*). También elimina de la lista a los nodos hijos cuyo padre haya quedado con una solución óptima relajada peor que la mejor solución entera disponible.
- El *framework* permite utilizar los distintos algoritmos de *pricing* basados en los modelos de PLEM y CP descriptos en las secciones anteriores.

Todo el *framework* se desarrolla teniendo como base un esquema de *Branch-and-Bound*. Para ello utilizamos un esquema clásico que mostramos en el siguiente pseudocódigo.

3.7.6. Problema maestro

Como mencionamos en la subsección 3.7.1, el modelo más simple e intuitivo para representar el problema SCA es el descripto por Grover *et al.* en el capítulo 16 del libro *Handbook of Optimization in Telecommunications* [64]. En este modelo cada variable representa la cantidad de veces que se utiliza un ciclo del grafo, por ello, se trata de una formulación con una cantidad exponencial de variables. Se trata de un modelo que no se puede descomponer, pero que tiene muy buenas relajaciones lineales. Recordemos que nos referimos a este modelo como **Problema Maestro** o **PM** y a su versión restringida (es decir con algunas de las variables) **Problema Maestro Restringido** o **PMR**.

A continuación repetimos la formulación del modelo de Grover *et al.* visto en la sección 3.3.

Variables:

$x_k \geq 0$: variable entera. Representa la cantidad de copias del ciclo k.

Formulación:

$$\min \sum_{k \in K} c_k x_k \tag{3.117}$$

sujeto a

$$\sum_{k \in K} p_k^e x_k \geq d_e \ \forall e \in E \tag{3.118}$$

3.7.7. Ramificación

Un esquema de ramificación válido divide el espacio de soluciones de forma tal que la solución fraccionaria actual quede excluida; las soluciones enteras, intactas; y la finitud

del algoritmo, asegurada. Es más, algunas reglas prácticas resultan útiles, tal como crear ramas de tamaños iguales de ser posible; esta regla se llama *balancear el árbol*. Se debe elegir un esquema de ramificación compatible con el subproblema de *pricing* para prevenir que columnas, cuya cantidad fue limitada a un máximo, sean regeneradas por el subproblema. Esto, en general, llevaría a encontrar la $k^{\text{ésima}}$ mejor solución al subproblema de *pricing* en lugar de la óptima. Independientemente de la complicación conceptual, esto modifica o destruye una posible buena estructura. Esto es de extrema importancia, sobre todo, cuando el subproblema se puede resolver utilizando algoritmos de optimización combinatoria. Dado ese caso, se podría pensar en otro esquema de ramificación.

El **Problema Maestro** admite soluciones que pueden ser fraccionarias en una sola variable. Supongamos el grafo de la figura 3.5: si las aristas ab, bc, cd y da tienen demandas nulas y costo nulo, y la arista ac tiene una unidad de demanda unitaria y costo muy alto, entonces la solución óptima del **PM** es de una unidad de ciclo formado por las aristas ab, bc, cd y da. Pero si relajamos la restricción de integralidad, en la solución óptima, la variable que indica las repeticiones de este ciclo toma el valor 1/2.

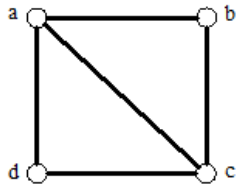

Fig. 3.5: Instancia con solución fraccionaria para el problema SCA

Esta característica, sumada a que el problema carece de otros conceptos involucrados como variables, nos impidieron desarrollar un esquema del tipo Ryan-Foster, aunque esto no demuestra que no pueda existir tal esquema. Ante este impedimento, decidimos modificar el problema de *pricing* (detallado en la subsección 3.7.9) incorporando una restricción por cada ciclo que la ramificación no permite. Claramente esto degrada la eficiencia del algoritmo de *pricing* a medida que descendemos en el árbol de *Branch-and-Bound*.

En el esquema de *Branch-and-Bound* surgen varios aspectos que determinan la eficiencia del algoritmo implementado. La mejor opción para cada aspecto depende del problema a resolver y de las otras características del algoritmo. Algunos de esos aspectos son los siguientes:

- Cada vez que ramificamos surgen problemas nuevos que se agregan a la lista de problemas pendientes. El orden en que resolvamos los problemas de la lista puede influir notablemente en la posibilidad de disponer rápidamente, o no, de una buena cota inferior que permita descartar ramas. Las dos opciones más utilizadas son:

  - *Primero el más nuevo.* Esta opción permite modificar el problema padre y reprocesar ahorrando tiempo de cálculo. La desventaja es que las soluciones son muy similares y la cota cambiará muy lentamente.
  - *Primero el de menor cota.* Esta opción intenta reducir la distancia lo más pronto posible entre la mejor solución entera disponible y la mejor relajación lineal hasta el momento.

  En nuestra implementación elegimos el nodo de menor cota debido a su mayor eficiencia en los experimentos realizados.

- Variable a elegir. En cada nodo obtenemos una solución relajada del **PMR** que puede contener diversas variables fraccionarias. La elección de la variable a partir de la cual ramificar afectará la calidad de los nodos hijos. Dos opciones comúnmente usadas son:

  - *Ramificación simple (Most Infeasible Branching).* Consiste en tomar la variable cuyo componente fraccionario sea lo más próximo posible a 0,5.
  - *Ramificación fuerte.* Para cada una de las posibles variables fraccionarias, calcular las cotas que tendrán sus dos problemas hijos y considerar aquella variable con mayor media entre las cotas de sus subproblemas. Este método es más complejo pero producirá nodos hijos con ambas cotas mayores.

En nuestra implementación, la ramificación fuerte generó los mejores resultados. Es importante aclarar que las soluciones relajadas de nuestro problema no involucran muchas variables, esto hace que la ramificación fuerte no sea costosa en términos de procesamiento.

- No crear dos nodos hijos, sino varios. Esto se puede hacer eligiendo una variable no entera y fijarla en distintos valores enteros que ésta pueda asumir, creando, en general, más de dos nodos. Otra alternativa es tomar un grupo de variables cuya suma sea fraccionaria y ramificar en función de dicha suma. Estas dos opciones no generaron mejores resultados que la creación de dos nodos hijos en nuestra implementación.

Es fundamental que cualquiera que sean los criterios seleccionados para ramificar, los nodos hijos constituyan problemas lo más distantes posibles del nodo padre. Si algún nodo (o varios) es muy diferente del problema padre, pero otro nodo hijo es muy parecido, entonces el proceso de *Branch-and-Bound* tiende a dar malos resultados. En efecto, se habrá avanzado poco cuando se pase del problema padre a ese problema hijo parecido. Por ello, se suele decir que la regla de ramificación debe tender a crear nodos *equilibrados*, con cotas lo más distantes posibles de la cota del padre. Esto justifica el proceder de la ramificación fuerte antes descrito. Para mas detalles respecto de las reglas de ramificación referimos a Achterberg et al. [1].

3.7.8. Proceso de acotación

En un esquema de *Branch-and-Bound*, el proceso de acotación también admite distintas posibles variantes. La primera opción es utilizar el valor óptimo de la relajación lineal. Sin embargo, en algunos problemas enteros es posible la obtención de otras cotas inferiores a partir de consideraciones lógicas o combinatorias del problema.

En nuestra implementación, aprovechamos la ramificación fuerte descripta en la subsección anterior para evaluar la cota que tendrán los nodos hijos y compararlas con la mejor

solución entera disponible. En caso de que la variable elegida para ramificar tenga uno o ambos nodos hijos con cotas mayores a la mejor solución entera se los descarta. El gap informado por el algoritmo es obtenido recorriendo toda la lista de nodos pendientes de procesar para determinar la función objetivo mas pequeña (correspondiente al nodo padre) y hacer el cociente entre ella y el valor de la función objetivo de la mejor solución entera encontrada hasta ese momento.

3.7.9. *Pricing*

La función del problema de *pricing* es determinar si existe una columna (ciclo) que pueda ser incorporada al PMR de forma tal que mejore su función objetivo. En caso de que no exista tal columna, y se trate de un algoritmo exacto, el *pricing* demuestra la optimalidad de la solución.

Los algoritmos de *pricing* que utilizamos están basados en los modelos descritos en la sección 3.4 y en la sección 3.5 con las siguientes modificaciones.

- El parámetro J se fija en el valor uno.
- La función objetivo de los problemas de *pricing* consiste en minimizar el costo neto (costo de construirlo menos beneficio por proteger sus aristas y el doble por cubrir sus cuerdas) del ciclo propuesto. Suponiendo que la variable binaria x_e indica si la arista e es parte del ciclo (*on-cycle*), la variable y_e indica si la arista e es protegida como *straddle*, c_e es el costo y π_e el valor del dual, entonces la función objetivo será: $z = \sum_{\forall e \in E} (c_e - \pi_e) x_e - 2\pi_e y_e$.
- La restricción de cubrimiento de demandas se elimina.
- Restricciones de ciclos prohibidos.

El proceso de *Branch-and-Bound* aplicado al PMR se encarga de obtener la integralidad

de la solución. En cada nodo cuya solución relajada sea fraccionaria, ramificamos el problema en dos subproblemas. Supongamos que la variable c_k, que representa las repeticiones del ciclo k en la solución del nodo padre, es la variable fraccionaria elegida para ramificar. El valor de dicha variable en la solución óptima relajada es representada por c_k^*. Entonces un nodo hijo incorpora la restricción $c_k \leq \lfloor c_k^* \rfloor$ y el otro nodo hijo incorpora la restricción $c_k \geq \lceil c_k^* \rceil$. En el primer caso, debemos asegurarnos de que el problema de *pricing* no genere el ciclo k como ciclo recomendado. Por este motivo, debemos agregar las *restricciones de ciclos prohibidos*. Esto hace que, para cada nivel del árbol de *Branch-and-Bound*, se tenga que agregar una restricción extra al problema de *pricing* en uno de los nodos hijos.

Sea el ciclo c_k un ciclo prohibido de longitud λ y sea x_e la variable binaria que indica si la arista e es parte del ciclo (*on-cycle*), entonces la proposición 3.7.1 es trivialmente válida.

Proposición 3.7.1. *La restricción 3.119 prohibe que el ciclo c_k forme parte de la solución factible, sin excluir del espacio de soluciones factibles a ningún otro ciclo.*

$$\sum_{e \in c_k} x_e \leq \lambda_{c_k} - 1 \tag{3.119}$$

El algoritmo de *pricing* es el cuello de botella de la implementación. Esto se debe a que lo resolvemos mediante un algoritmo *Branch-and-Cut* basado en un modelo de PLE y que además suma las restricciones de ciclos prohibidos a medida que se avanza en la profundidad del árbol. Esto nos lleva a pensar en distintas formas de limitarlo. Si la suma del doble de los beneficios es menor que el costo del ciclo de menor costo, entonces no tiene sentido intentar generar más columnas. El ciclo más barato se puede obtener mediante una única corrida del modelo de *pricing* (previa al proceso de *Branch-and-Price*).

El problema de *pricing* permite varias alternativas. Una de ellas es que no es necesario resolverlo hasta la optimalidad, ya que un ciclo que mejora la función objetivo del PMR es

suficiente, aunque esto puede requerir más iteraciones. Otra alternativa es que el modelo de *pricing* aporte varias columnas y no una sola. En nuestra implementación exigimos al algoritmo de *pricing* que solo devuelva un ciclo cuya función objetivo sea menor a un parámetro f. Esto nos permitió evaluar la opción de tomar varios ciclos y que todos ellos cumplan con la condición, pero la eficiencia no mejoró en ese caso. El parámetro f resulta de mayor importancia respecto del tiempo total del algoritmo de *Branch-and-Price*. También influye mucho con respecto al tiempo que requiere todo el proceso para demostrar la optimalidad de una solución.

El problema de *pricing* depende de los valores de la solución dual del PMR. En el caso de que tenga soluciones alternativas, es probable que funcione mejor el algoritmo de punto interior. En nuestra implementación, esta alternativa aumentó notablemente la cantidad de iteraciones de *pricing* necesarias, empeorando el tiempo global del algoritmo.

A continuación detallamos la formulación del modelo de la subsección 3.4.3 adaptado al problema de *pricing* de *Branch-and-Price*. Las adaptaciones son las que se describieron al inicio de esta subsección .

Coeficientes:

$\alpha = 1/|V|$: coeficiente para garantizar voltajes compatibles.

B: matriz de ciclos fundamentales del grafo G (μ x m).

C: conjunto de ciclos fundamentales con respecto a un árbol arbitrario.

L: número muy grande.

M: matriz de incidencia del grafo G (n x m).

M^t: matriz de incidencia transpuesta del grafo G (m x n).

Ω: conjunto de ciclos prohibidos.

Variables:

c_i: variable binaria. Toma valor 1 si el ciclo es generado por el ciclo fundamental i.

p_e: variable entera. Es una variable auxiliar que indica cuántos pares de ciclos fundamentales son seleccionados para generar el ciclo que incluyen a la arista e.

x_{eo}: variable binaria. Toma valor 1 si la arista e está en el ciclo generado. Los arcos son dirigidos y el índice o indica el sentido.

x_o: vector de variables x_{eo} de tamaño m.

x: suma de vectores de variables x_{eo}. Es decir, $x = x_0 + x_1$.

y_e: variable binaria. Toma valor 1 si la arista e es una cuerda (*straddle*) del ciclo generado.

z_k: variable binaria. Solo una de ellas toma valor 1 en cada ciclo generado para indicar el nodo raíz de los voltajes.

$volt_k$: variable continua. Indica el voltaje de cada nodo.

Formulación:

$$\sum_{\forall e \in E} (c_e - \pi_e)x_e - 2\pi_e y_e \tag{3.120}$$

sujeto a

$$\begin{aligned}
B.c &= 2.p + x && (3.121)\\
x_0 + x_1 &\leq 1, \forall e \in E && (3.122)\\
\sum_{e=uv \in E} (x_{uv0} - x_{uv1}) &= 0, \forall u \in V && (3.123)\\
Mx &\leq 2 && (3.124)\\
\frac{1}{2}(x_0 + x_1)(\frac{1}{2}M^t M - 2I) &\geq y && (3.125)\\
\sum_{k \in V} z_k &= 1 && (3.126)\\
volt_h - volt_d &\geq \alpha x_{e,o} - (1 - x_{e,o}) - Lz_h && (3.127)\\
& \forall e = (d,h) \in E, \forall o \in 0,1 && (3.128)\\
\sum_{e \in c_k} x_e &\leq \lambda_{c_p} - 1, \forall c_k \in \Omega && (3.129)\\
& c_i \in \{0,1\}, \forall i \in C && (3.130)\\
& p_e \in \{0,\infty), \forall e \in E && (3.131)\\
& x_e \in \{0,1\}, \forall e \in E && (3.132)\\
& y_e \in \{0,1\}, \forall e \in E && (3.133)\\
& z_k \in \{0,1\}, \forall k \in V && (3.134)\\
& volt_k \in \{0,\infty), \forall k \in V && (3.135)
\end{aligned}$$

3.7.10. Heurística para obtener integralidad

En la mayoría de los nodos del árbol de *Branch-and-Bound* obtendremos soluciones fraccionarias. Mediante una heurística, podemos obtener rápidamente una o varias soluciones enteras. Trivialmente, este problema admite que una solución relajada de PMR se redondee hacia arriba para obtener una solución entera factible. Naturalmente, en esta instancia pueden sobrar ciclos. Por este motivo, la heurística recorre los ciclos calculando sus costos y demandas cubiertas y luego ordena los ciclos por costo descendente y los recorre en ese sentido para verificar si se los puede retirar de la solución, manteniendo las demandas cubiertas.

3.7.11. Heurísticas primales

Las heurísticas primales consisten en tomar columnas del PMR con costo reducido nulo e intentar convertirlas en columnas con costo reducido negativo. Estas modificaciones se llevan a cabo mediante una heurística local similar a la utilizada en la implementación GRASP de la sección 3.6 y que se basa en recorrer las aristas del ciclo y analizar la vecindad de ambos nodos para determinar si se puede ampliar el ciclo, convirtiendo a la arista en cuestión en *straddle* y el costo reducido del ciclo pase a ser negativo. Luego debemos revisar que el ciclo generado no sea igual a uno ya existente en el PMR, porque esto generaría problemas de simetría.

3.7.12. Solución inicial

Como ya adelantamos en la sección 3.6, a partir de nuestra implementación de un algoritmo GRASP, obtenemos un conjunto de ciclos buenos que garantizan una solución inicial factible de muy buena calidad. Si bien GRASP maneja una cantidad importante de ciclos, como solución inicial solo tomamos los ciclos de la mejor solución obtenida y los ciclos obtenidos por la búsqueda local exacta.

Igualmente a lo que ocurre en la subsección 3.7.11, es muy importante que GRASP no devuelva ciclos repetidos, porque eso generaría simetría en el árbol de *Branch-and-Bound*.

El conjunto de ciclos aportados por GRASP es tomado por una heurística de generación de columnas que incorpora nuevas columnas y devuelve una solución entera de excelente calidad (utilizando el algoritmo *Branch-and-Cut* de CPLEX). El algoritmo de *Branch-and-Price* solo toma como columnas iniciales las de la solución entera y las que tienen buenos costos reducidos. Esto es de fundamental importancia para la eficiencia del árbol de *Branch-and-Bound*.

3.7.13. Pseudocódigo

1 **procedure** Generación_de_Columnas
2 Determinar una solución inicial factible del PMR.
3 Inicializar el pool de columnas como vacío.
4 Resolver el PMR actual.
5 Ejecutar heurística de integralidad.
6 Ejecutar heurística primal.
7 Eliminar columnas no básicas con costo reducido muy negativo del PMR.
8 Si el pool de columnas todavía contiene columnas con costo reducido positivo, seleccionar un subconjunto de ellas y agregarlos al PMR e ir al paso 4.
9 Vaciar el pool de columnas.
10 Llamar al algoritmo exacto de pricing para probar la optimalidad o generar una o más columnas con costo reducido positivo. Si se generan columnas, agregarlas al pool e ir al paso 8.
11 **end** Generación_de_Columnas

Pseudocódigo del proceso de generación de columnas descripto en la subsección 3.7.3 adaptado a nuestra implementación.

3.7.14. Conclusiones del algoritmo de *Branch-and-Price*

Logramos un algoritmo *Branch-and-Price* muy eficiente a partir de la combinación de distintos algoritmos. Es de fundamental importancia, para la eficiencia del árbol de *Branch-and-Price*, que se utilice una solución inicial de la mejor calidad posible. Esto lo logramos ejecutando previamente la heurística de generación de columnas que devuelve una solución entera de excelente calidad.

En la implementación del algoritmo de *Branch-and-Price* no pudimos desarrollar un esquema de ramificación del tipo Ryan-Foster como mencionamos en la subsección 3.7.7. Esto degrada la eficiencia del algoritmo de *pricing* a medida que descendemos en el árbol de *Branch-and-Bound*, convirtiéndolo en el cuello de botella de todo el proceso. Por lo tanto, las características más importantes de la implementación están relacionadas a minimizar las llamadas al algoritmo de *pricing*. En ese sentido aportaron una importante mejora a la eficiencia las siguientes características:

1. Calidad de la solución inicial.
2. Cálculo de cota inferior.
3. Heurística de integralidad.
4. Heurística primal.

3.8. Resultados numéricos. SCA

El mejor modelo reportado en la bibliografía corresponde a Wu *et al.* [84]. Con el objetivo de evaluar la eficiencia de nuestros algoritmos decidimos implementarlo en CPLEX utilizando su algoritmo *Branch-and-Cut*. A modo de referencia y verificación también implementamos, de la misma forma, el modelo propuesto por Grover *et al.* [64]. Este último modelo requiere de todos los ciclos del grafo que fueron obtenidos implementando el algoritmo de Johnson [39].

Nuestros algoritmos obtuvieron excelentes resultados (menos de 1,5 % del óptimo) para las instancias de redes artificiales (grafos completos desde K_9 hasta K_{12}) con más de 59 millones de ciclos posibles. Para topologías de redes reales (COST239 y las instancias VZ_US_PIP) de hasta 27 nodos, todos los resultados que obtuvimos se ubicaron a menos de 4,5 % del óptimo en los primeros 5 minutos de procesamiento.

Para las instancias más difíciles, nuestro algoritmo de *Branch-and-Price* fue el de mejor rendimiento. En el resto de las instancias, en los casos en que no fue el mejor algoritmo, estuvo muy cerca de serlo.

En la próxima subsección detallaremos las principales métricas de las instancias utilizadas, en la subsección 3.8.2 la parametrización de los algoritmos y finalmente en la subsección 3.8.3 los resultados detallados.

3.8.1. Instancias para SCA

Las topologías de redes de EEUU fueron obtenidas en base a información pública, interpretada gracias a nuestra experiencia laboral de 2007 a 2012 en el equipo *Global Planning Tools* de *Verizon Business*. En esta experiencia desarrollamos modelos y algoritmos para la optimización de diversos aspectos de la red de dicha compañía en EEUU y a nivel global.

También se usaron redes creadas en forma arbitraria (instancias de K_9 a K_{12}) y algunas

redes conocidas que tienen resultados publicados. En la siguiente tabla se muestran las principales métricas de las instancias. En el anexo B se muestran los detalles y grafos de *Network 1* [46] y *COST239* [52].

En general, el valor óptimo del parámetro J es desconocido. Determinamos heurísticamente un valor para cada instancia en base a prueba y error, cuando J es demasiado pequeño no hay solución factible y, cuando es demasiado grande, los valores de la función objetivo no mejoran (para un tiempo de cálculo razonablemente alto).

Las principales características de las instancias son:

Instancia	Nombre de la instancia.
Nodos	Cantidad de nodos.
Aristas	Cantidad de aristas.
Ciclos	Cantidad de ciclos.
D	Demanda máxima para una arista.
μ	Dimensión del espacio de ciclos (*i.e.* número ciclomático de G).
Óptimo	En los casos que se indica $\leq$, se trata de la mejor solución encontrada entre todos los algoritmos evaluados.
J	Parámetro J utilizado en los experimentos.

A continuación describimos sus valores.

Instancia	**Nodos**	**Aristas**	**Ciclos**	**D**	μ	**Óptimo**	**J**
COST_239	11	26	3531	11	16	$ 32.340	7
RED_001	4	5	3	5	2	$ 13.720	5
RED_003	5	7	6	4	3	$ 14.240	4
RED_004	4	6	7	6	3	$ 56	4
Network_001	10	22	833	10	13	$ 113,9	5
K_8	8	28	8018	19	21	$ 55.555	14
K_9	9	36	62814	20	28	$ 36.653	15
K_{10}	10	45	556014	18	36	$ 23.720	15
K_{11}	11	55	5488059	20	45	$\leq$ $ 59.526	15
K_{12}	12	66	59740609	20	55	$\leq$ $ 30.342	15
VZ_US_PIP_001	13	39	106967	11	27	$ 32.240	7
VZ_US_PIP_002	12	35	37286	11	24	$ 37.370	6
VZ_US_PIP_003	12	35	37286	12	24	$ 35.170	7
VZ_US_PIP_004	12	35	37286	12	24	$ 39.980	8
VZ_US_PIP_005	11	32	15341	11	22	$ 24.937	7
VZ_US_PIP_006	11	32	15341	10	22	$ 26.190	6
VZ_US_PIP_007	11	32	15341	14	22	$ 30.534	7
VZ_US_PIP_008	11	32	15341	12	22	$ 32.721	7
VZ_US_PIP_009	11	32	15341	12	22	$ 37.071	7
VZ_US_PIP_010	11	32	15341	12	22	$ 44.084	8
VZ_US_PIP_011	10	27	3354	12	18	$ 42.054	8
VZ_US_PIP_012	9	24	1636	12	16	$ 35.754	8
VZ_US_PIP_013	8	17	167	9	10	$ 29.984	7
VZ_US_PIP_014	7	14	70	7	8	$ 14.740	6
VZ_US_PIP_015	6	10	18	6	5	$ 14.775	6
VZ_US_PIP_016	14	50	?	11	37	$\leq$ $ 36.310	8
VZ_US_PIP_017	14	56	?	11	43	$\leq$ $ 38.780	9
VZ_US_PIP_018	21	67	?	11	47	$\leq$ $ 58.350	11
VZ_US_PIP_019	23	73	?	11	51	$\leq$ $ 72.370	16
VZ_US_PIP_020	25	78	?	11	54	$\leq$ $ 56.470	19
VZ_US_PIP_021	27	85	?	11	59	$\leq$ $ 47.430	24

3.8.2. Algoritmos evaluados y parametrizaciones

Los algoritmos evaluados son los siguientes:

Nombre	Método	Autores
Grover	Gen. ciclos + *Branch-and-Cut* (PLE)	Johnson [39] + Grover *et al.* [64]
Wu	*Branch-and-Cut* (PLEM)	Wu *et al.* [84]
MTZ	*Branch-and-Cut* (PLE)	Propuesto en este trabajo.
Volt	*Branch-and-Cut* (PLEM)	Propuesto en este trabajo.
Espacio	*Branch-and-Cut* (PLEM)	Propuesto en este trabajo.
CPConstr	CP	Propuesto en este trabajo.
CPEspacio	CP	Propuesto en este trabajo.
GRASP	GRASP	Propuesto en este trabajo.
B&P	*Branch-and-Price*	Propuesto en este trabajo.

Los algoritmos se implementaron en C++ de Visual Studio 2012, Concert Technologies e IBM ILOG CPLEX 12.6. El hardware es un procesador intel core i3, 2,20GHz (2 núcleos, 4 procesadores lógicos), 8Gb RAM corriendo bajo Windows 10 Pro.

Los modelos de PLE y PLEM (Grover, Wu, MTZ, Volt y Espacio) son implementados utilizando el algoritmo *Branch-and-Cut* de CPLEX. Las desigualdades válidas y los cortes de optimalidad propuestos están disponibles en todos los nodos como *User Cuts*. CPLEX posee una rutina para administrar automáticamente los cortes provistos por el usuario. A partir de la versión 12.3 también permite modificar esa rutina mediante *callbacks*, pero no las utilizamos.

El algoritmo GRASP incluye en su búsqueda local el algoritmo *Branch-and-Cut* de CPLEX para el modelo de PLEM detallado en la subsección 3.6.5. Los experimentos para

distintos límites de tiempo son independientes, por lo tanto, en el caso de GRASP, pueden obtener peores resultados con límites mayores.

El algoritmo *Branch-and-Price* también utiliza el algoritmo *Branch-and-Cut* de CPLEX para el modelo PLEM como *pricing* y para el modelo PLE en la heurística de *generación de columnas* que provee la primera solución entera. El modelo del PMR también se resuelve con CPLEX pero en este caso con el algoritmo de programación lineal continua.

Los parámetros de CPLEX para los distintos algoritmos son los siguientes:

Algoritmo	**RootAlg**	**NodeSel**	**MIPEmph**	**Prior.**	**SolLim**
Grover	3	2	3	-	-
Wu	-	-	4	-	-
MTZ	-	-	4	x	-
Volt	-	-	4	x, z	-
Espacio	-	-	4	c, y	-
GRASP. B. Local	-	-	4	-	-
B&P. Pricing	-	-	4	-	1
B&P. PMR	-	-	-	-	-

Entendiendo que:

- **RootAlg**: Algoritmo de programación lineal continua. (-) *Default*; CPLEX elige. 3; *Network simplex.*
- **NodeSel**: Selección de próximo nodo cuando realiza un *backtrack.* (-) *Default*; *Depth-first search*; 2 *Best-estimate search.*
- **MIPEmph**: Controla el balance entre velocidad, factibilidad, optimalidad y acotación en PLE. 3; *Emphasize moving best bound.* 4; *Emphasize finding hidden feasible solutions.*
- **Prior.**: Prioridad de variables.

- **SolLim**: Cantidad de soluciones a encontrar antes de detenerse.

Los parámetros se eligieron a partir de una extensa experimentación con las instancias más grandes (reales y artificiales).

Es interesante mencionar que el algoritmo *Branch-and-Price* no necesita generar más de 300 variables para el límite máximo de tiempo y la instancia mas grande (VZ_US_PIP_021).

Los modelos de CP se probaron con las opciones por default, con la excepción de sendas fases de búsqueda que debieron ser adaptadas a las restricciones para evitar simetría, como se describió oportunamente.

Todos los experimentos se realizaron sobre las instancias descriptas anteriormente en 3.8.1 con límites de tiempos de cálculo de 5, 30 y 120 minutos y *Optimality Gap* en 0 %.

3.8.3. Resultados detallados

Tiempo límite: 5 minutos. Valores de la función objetivo resultante para cada instancia y algoritmo evaluado.

Instancia	Grover	Wu	MTZ	Volt	Espacio	CPConstr	CPEspacio	GRASP	B&P
COST_239	**32.340**	**32.340**	**32.340**	**32.340**	**32.340**	**32.340**	**32.340**	33.780	**32.340**
RED_001	**13.720**	**13.720**	**13.720**	**13.720**	**13.720**	**13.720**	**13.720**	**13.720**	**13.720**
RED_003	**14.240**	**14.240**	**14.240**	**14.240**	**14.240**	**14.240**	**14.240**	**14.240**	**14.240**
RED_004	**56**	**56**	**56**	**56**	**56**	**56**	**56**	**56**	**56**
Network_001	**113,9**	**113,9**	**113,9**	**113,9**	**113,9**	**113,9**	**113,9**	114,6	**113,9**
K_8	**55.555**	**55.555**	**55.555**	**55.555**	**55.555**	**55.555**	**55.555**	58.914	**55.555**
K_9	**36.653**	**36.653**	**36.653**	**36.653**	**36.653**	**36.653**	**36.653**	39.057	**36.653**
K_{10}	-	**23.720**	24.263	24.815	23.961	-	28.310	25.207	**23.720**
K_{11}	-	59.625	64.305	63.713	59.534	-	91.400	64.521	**59.526**
K_{12}	-	**30.410**	37.523	30.894	37.533	-	-	31.147	**30.410**
VZ_US_PIP_001	**32.240**	32.300	34.390	33.440	32.970	-	45.230	35.750	32.255
VZ_US_PIP_002	**37.370**	**37.370**	**37.370**	**37.370**	**37.370**	-	45.230	40.900	**37.370**
VZ_US_PIP_003	**35.170**	35.410	35.820	36.060	**35.170**	-	38.600	36.730	**35.170**
VZ_US_PIP_004	**39.980**	**39.980**	43.850	42.990	42.080	-	46.830	42.390	**39.980**
VZ_US_PIP_005	**24.937**	**24.937**	27.754	26.297	25.083	-	32.908	27.044	25.066
VZ_US_PIP_006	**26.190**	**26.190**	**26.190**	**26.190**	**26.190**	-	28.024	27.430	**26.190**
VZ_US_PIP_007	**30.534**	**30.534**	**30.534**	**30.534**	**30.534**	-	30.905	32.787	**30.534**
VZ_US_PIP_008	**32.721**	32.736	33.986	35.246	33.291	-	37.117	34.089	**32.721**
VZ_US_PIP_009	**37.071**	**37.071**	37.616	37.616	37.351	37.351	37.351	40.094	**37.071**
VZ_US_PIP_010	**44.084**	**44.084**	45.219	45.219	44.274	44.274	44.274	45.984	44.094
VZ_US_PIP_011	**42.054**	42.354	44.769	44.769	42.204	42.204	42.204	43.572	42.154
VZ_US_PIP_012	**35.754**	**35.754**	36.086	36.086	35.816	35.816	35.816	**35.754**	**35.754**
VZ_US_PIP_013	**29.984**	**29.984**	**29.984**	**29.984**	**29.984**	**29.984**	**29.984**	30.792	**29.984**
VZ_US_PIP_014	**14.740**	**14.740**	**14.740**	**14.740**	**14.740**	**14.740**	**14.740**	**14.740**	**14.740**
VZ_US_PIP_015	**14.775**	**14.775**	**14.775**	**14.775**	**14.775**	**14.775**	**14.775**	**14.775**	**14.775**
VZ_US_PIP_016	-	**36.310**	47.754	46.780	44.580	-	59.550	39.400	36.500
VZ_US_PIP_017	-	42.180	58.220	69.980	52.940	-	-	41.880	**38.780**
VZ_US_PIP_018	-	58.420	80.005	139.500	161.865	-	-	68.350	**58.350**
VZ_US_PIP_019	-	-	169.520	-	-	-	-	82.570	**72.370**
VZ_US_PIP_020	-	-	-	-	-	-	-	60.860	**56.470**
VZ_US_PIP_021	-	-	-	-	-	-	-	53.990	**47.595**

Tiempo límite: 30 minutos. Valores de la función objetivo resultante para cada instancia y algoritmo evaluado.

Instancia	Grover	Wu	MTZ	Volt	Espacio	CPConstr	CPEspacio	GRASP	B&P
COST_239	**32.340**	**32.340**	**32.340**	**32.340**	**32.340**	**32.340**	**32.340**	33.780	**32.340**
RED_001	**13.720**	**13.720**	**13.720**	**13.720**	**13.720**	**13.720**	**13.720**	**13.720**	**13.720**
RED_003	**14.240**	**14.240**	**14.240**	**14.240**	**14.240**	**14.240**	**14.240**	**14.240**	**14.240**
RED_004	**56**	**56**	**56**	**56**	**56**	**56**	**56**	**56**	**56**
Network_001	**113,9**	**113,9**	**113,9**	**113,9**	**113,9**	**113,9**	**113,9**	114,6	**113,9**
K_8	**55.555**	**55.555**	**55.555**	**55.555**	**55.555**	**55.555**	**55.555**	58.444	**55.555**
K_9	**36.653**	**36.653**	**36.653**	**36.653**	**36.653**	**36.653**	**36.653**	39.535	**36.653**
K_{10}	-	**23.720**	**23.720**	**23.720**	**23.720**	44.034	27.106	24.591	**23.720**
K_{11}	-	59.625	59.753	59.568	59.534	-	85.148	63.505	**59.526**
K_{12}	-	30.368	30.368	30.368	**30.342**	-	37.477	31.021	30.410
VZ_US_PIP_001	**32.240**	**32.240**	34.390	32.400	**32.240**	-	36.570	34.260	32.255
VZ_US_PIP_002	**37.370**	**37.370**	**37.370**	**37.370**	**37.370**	-	43.410	39.790	**37.370**
VZ_US_PIP_003	**35.170**	35.410	**35.170**	35.270	**35.170**	-	38.400	36.860	**35.170**
VZ_US_PIP_004	**39.980**	**39.980**	41.800	41.320	40.980	57.020	45.780	43.070	**39.980**
VZ_US_PIP_005	**24.937**	**24.937**	25.036	**24.937**	**24.937**	28.429	28.919	25.596	25.038
VZ_US_PIP_006	**26.190**	**26.190**	**26.190**	**26.190**	**26.190**	-	28.024	27.268	**26.190**
VZ_US_PIP_007	**30.534**	**30.534**	**30.534**	**30.534**	**30.534**	-	30.905	32.297	**30.534**
VZ_US_PIP_008	**32.721**	32.736	32.976	32.736	32.736	-	36.252	34.241	**32.721**
VZ_US_PIP_009	**37.071**	**37.071**	**37.071**	**37.071**	**37.071**	**37.071**	**37.071**	40.544	**37.071**
VZ_US_PIP_010	**44.084**	**44.084**	**44.084**	**44.084**	**44.084**	**44.084**	**44.084**	45.912	**44.084**
VZ_US_PIP_011	**42.054**	42.354	42.354	42.354	**42.054**	**42.054**	**42.054**	43.609	**42.054**
VZ_US_PIP_012	**35.754**	**35.754**	**35.754**	**35.754**	**35.754**	**35.754**	**35.754**	37.067	**35.754**
VZ_US_PIP_013	**29.984**	**29.984**	**29.984**	**29.984**	**29.984**	**29.984**	**29.984**	30.122	**29.984**
VZ_US_PIP_014	**14.740**	**14.740**	**14.740**	**14.740**	**14.740**	**14.740**	**14.740**	**14.740**	**14.740**
VZ_US_PIP_015	**14.775**	**14.775**	**14.775**	**14.775**	**14.775**	**14.775**	**14.775**	**14.775**	**14.775**
VZ_US_PIP_016	-	**36.310**	44.260	36.840	36.340	-	42.170	38.320	36.500
VZ_US_PIP_017	-	38.860	52.920	41.460	39.210	-	43.120	41.700	**38.780**
VZ_US_PIP_018	-	58.420	74.630	100.700	65.920	-	-	67.970	**58.350**
VZ_US_PIP_019	-	73.540	102.070	90.750	105.780	-	-	82.340	**72.370**
VZ_US_PIP_020	-	-	72.110	116.310	-	-	-	59.600	**56.470**
VZ_US_PIP_021	-	-	70.710	-	-	-	-	52.320	**47.430**

Tiempo límite: 120 minutos. Valores de la función objetivo resultante para cada instancia y algoritmo evaluado.

Instancia	Grover	Wu	MTZ	Volt	Espacio	CPConstr	CPEspacio	GRASP	B&P
COST_239	**32.340**	**32.340**	**32.340**	**32.340**	**32.340**	**32.340**	**32.340**	33.500	**32.340**
RED_001	**13.720**	**13.720**	**13.720**	**13.720**	**13.720**	**13.720**	**13.720**	**13.720**	**13.720**
RED_003	**14.240**	**14.240**	**14.240**	**14.240**	**14.240**	**14.240**	**14.240**	**14.240**	**14.240**
RED_004	**56**	**56**	**56**	**56**	**56**	**56**	**56**	**56**	**56**
Network_001	**113,9**	**113,9**	**113,9**	**113,9**	**113,9**	**113,9**	**113,9**	114,6	**113,9**
K_8	**55.555**	**55.555**	**55.555**	**55.555**	**55.555**	**55.555**	**55.555**	58.282	**55.555**
K_9	**36.653**	**36.653**	**36.653**	**36.653**	**36.653**	**36.653**	**36.653**	38.187	**36.653**
K_{10}	-	**23.720**	**23.720**	**23.720**	**23.720**	27.505	27.106	24.468	**23.720**
K_{11}	-	59.625	59.753	59.568	59.534	-	76.502	63.071	**59.526**
K_{12}	-	30.368	30.368	30.368	**30.342**	-	34.890	31.431	30.410
VZ_US_PIP_001	**32.240**	**32.240**	**32.240**	32.400	**32.240**	38.915	35.880	34.360	32.255
VZ_US_PIP_002	**37.370**	**37.370**	**37.370**	**37.370**	**37.370**	-	43.410	39.570	**37.370**
VZ_US_PIP_003	**35.170**	35.410	**35.170**	35.270	**35.170**	39.190	37.060	37.200	**35.170**
VZ_US_PIP_004	**39.980**	**39.980**	41.800	41.320	40.980	57.020	42.630	41.210	**39.980**
VZ_US_PIP_005	**24.937**	**24.937**	25.036	**24.937**	**24.937**	28.429	27.220	26.115	**24.937**
VZ_US_PIP_006	**26.190**	**26.190**	**26.190**	**26.190**	**26.190**	-	27.774	26.591	**26.190**
VZ_US_PIP_007	**30.534**	**30.534**	**30.534**	**30.534**	**30.534**	-	30.905	32.643	**30.534**
VZ_US_PIP_008	**32.721**	32.736	32.976	32.736	32.736	-	34.326	34.169	**32.721**
VZ_US_PIP_009	**37.071**	**37.071**	**37.071**	**37.071**	**37.071**	**37.071**	**37.071**	40.334	**37.071**
VZ_US_PIP_010	**44.084**	**44.084**	**44.084**	**44.084**	**44.084**	**44.084**	**44.084**	46.049	**44.084**
VZ_US_PIP_011	**42.054**	42.354	42.354	42.354	**42.054**	**42.054**	**42.054**	43.602	**42.054**
VZ_US_PIP_012	**35.754**	**35.754**	**35.754**	**35.754**	**35.754**	**35.754**	**35.754**	37.017	**35.754**
VZ_US_PIP_013	**29.984**	**29.984**	**29.984**	**29.984**	**29.984**	**29.984**	**29.984**	30.122	**29.984**
VZ_US_PIP_014	**14.740**	**14.740**	**14.740**	**14.740**	**14.740**	**14.740**	**14.740**	**14.740**	**14.740**
VZ_US_PIP_015	**14.775**	**14.775**	**14.775**	**14.775**	**14.775**	**14.775**	**14.775**	**14.775**	**14.775**
VZ_US_PIP_016	-	**36.310**	44.260	36.840	36.340	-	41.370	39.550	36.500
VZ_US_PIP_017	-	38.860	52.920	41.460	39.210	-	43.120	40.950	**38.780**
VZ_US_PIP_018	-	58.420	72.390	69.730	64.580	-	-	68.020	**58.350**
VZ_US_PIP_019	-	73.540	102.070	90.750	95.515	-	-	81.740	**72.370**
VZ_US_PIP_020	-	-	72.110	116.310	-	-	-	59.990	**56.470**
VZ_US_PIP_021	-	-	70.710	-	-	-	-	52.440	**47.430**

3.9. Conclusiones sobre SCA

Hemos propuesto tres modelos de *Programación Lineal Entera* que implementamos sobre CPLEX (**Branch-and-Cut**), dos modelos de *Programación por Restricciones* implementados sobre CPLEX CP, un algoritmo GRASP con búsqueda local exacta, un algoritmo heurístico de *Generación de Columnas* y un *Branch-and-Price* exacto, cada uno de ellos con sus ventajas y desventajas, siendo el algoritmo **Branch-and-Cut** basado en el modelo de Grover muy eficiente para instancias pequeñas y algunos de los algoritmos basados en los modelos de PLEM y el GRASP muy buenos para las instancias más difíciles. Pero, claramente, la forma más eficiente de resolver este problema es mediante el algoritmo *Branch-and-Price* con el que rápidamente obtuvimos excelentes resultados (menos de 1,5 % del óptimo) para las instancias de redes artificiales (grafos completos desde K_9 hasta K_{12}) con mas de 59 millones de ciclos posibles. Para topologías de redes reales (COST239 y las instancias VZ_US_PIP) de hasta 27 nodos obtuvimos resultados de menos de 4,5 % del óptimo.

4. PROBLEMA DE ALOCACIÓN DE CAPACIDADES DE REPUESTO EN FIPP P-CICLOS

Los esquemas llamados *path-segment* o *flow-protecting* generalizan el concepto de protección de vínculos cuerda (*straddles*) y de ciclo (*on-cycle*). La generalización consiste en proteger flujos contiguos de demanda sobre caminos (*path segment*) que pueden contener multiples vínculos y nodos. No es necesario proteger los caminos completos según las demandas, sino que alcanza con proteger partes. Este tipo de esquema ofrece, en general, mucho mayor eficiencia, pero aumenta la complejidad de operación. Sin embargo, el uso selectivo de tan solo uno o unos pocos caminos a proteger, en conjunto con otros esquemas de protección, puede ser atractivo. Uno de esos posibles usos sería el de dar soporte a un transporte óptico transparente de un flujo expreso que debe atravesar una red regional. Para más detalles ver Grover *et al.* [64].

El concepto de *Failure Independent Path Protecting* (FIPP) p-ciclos surge en 2005 en un trabajo de Grover *et al.* [27]. Se trata de una extensión del concepto básico de p-ciclo orientado a un esquema de protección de caminos (rutas). Este esquema protege las rutas de principio a fin (y no solo segmentos) de fallas de nodo o de vínculo.

En este capítulo abordaremos el problema SCA para FIPP p-ciclos (de ahora en más diremos directamente problema FIPP). Para ello proponemos un nuevo modelo PLEM implementado sobre el algoritmo **Branch-and-Cut** de CPLEX, un modelo de CP, una metaheurística GRASP con búsqueda local exacta y un algoritmo **Branch-and-Price** exacto.

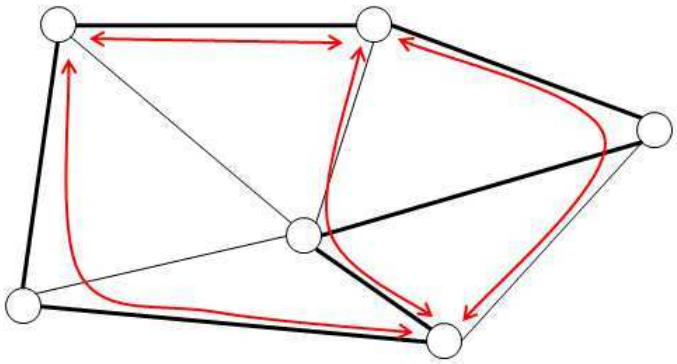

Fig. 4.1: FIPP p-ciclo

4.1. Complejidad y dominación en el problema FIPP

El problema FIPP es una generalización del problema SCA para p-ciclos. Por este motivo para realizar una reducción polinomial de SCA a FIPP, basta con tomar las demandas de SCA directamente. Como vimos en la sección 3.1, el problema SCA es NP-difícil, por lo tanto FIPP también lo es.

La proposición 3.2.1 es igualmente válida para este problema.

4.2. Bibliografía sobre FIPP

En 2005, Grover *et al.* [27] crean un nuevo tipo de esquema de protección basado en p-ciclos llamados FIPP p-ciclos. Ellos aclaran que un problema importante al momento de aplicar un método de protección compartida de camino (*shared-backup path protection*, **SBPP**) a una red óptica es que los canales de repuesto para el camino de protección deben ser conectados inmediatamente después de ocurrida la falla. Eso lleva tiempo y señalización, pero, lo más importante, hasta que todas las conexiones se realicen, no se sabe si el camino de protección tendrá la integridad de transmisión adecuada. Por lo tanto, la velocidad y la integridad de transmisión son razones importantes para intentar que los

caminos de protección esten completamente preconectados antes de la falla. Los p-ciclos son, por definición, completamente preconectados pero no son estructuras de protección punto a punto.

El método SBPP es eficiente con respecto a la cantidad de canales necesarios, independientemente de las fallas, es decir que solo deben ser detectadas en alguno de los puntos finales y son ellos los que activan los cambios necesarios para transmitir por otro camino, pero los caminos de protección no están preconectados. Los **FIPP p-ciclos** soportan esta misma independencia, pero son completamente preconectados. Como esquema completamente preconectado y orientado a proteger caminos los FIPP son potencialmente más atractivos para redes ópticas que SBPP. Los resultados en la bibliografía indican que, además, pueden ser igual de eficientes.

En una red óptica, la preconexión puede ser incluso más importante que la velocidad de restauración, debido a que esto permite que los caminos de protección puedan ser diseñados y probados, y su condición de trabajo se sabe antes de ser utilizada.

En 2008, Baloukov *et al.* [8] desarrollaron heurísticas para resolver este problema; en 2009 Rocha *et al.* [68] desarrollaron un algoritmo de generación de columnas jerárquica.

En 2012, Jaumard *et al.* [36] notaron que para que los FIPP p-ciclos protejan el 100 % de las fallas de nodos, es necesario exigir, adicionalmente, que las rutas sean disjuntas en el sentido de los nodos para cada FIPP p-ciclo.

4.3. Modelos de programación entera mixta para FIPP

Al igual que en el problema SCA de p-ciclos, en el problema de SCA para FIPP p-ciclos, dado un ciclo, podemos calcular fácilmente su costo y sus cuerdas y determinar las demandas que protege. Si tuviésemos todos los ciclos posibles del grafo, podríamos obtener todas las combinaciones posibles de caminos de demandas y ciclos y obtener también el óptimo revisando cuál o cuáles de estas combinaciones es la más barata, satisfaciendo las demandas. Obviamente, ésta sería una forma extremadamente poco eficiente. (Recordemos que un grafo completo tiene una cantidad exponencial de ciclos, según la ecuación 2.1).

En este caso, el ruteo también está predefinido, es decir que es dato para el problema. Otra aclaración importante es que cada ruta debe estar completamente incluida dentro de un único p-ciclo, es decir que no se puede cubrir un tramo con un p-ciclo y el resto de la ruta con otro p-ciclo.

Aquí también, disponiendo de todos los ciclos, podemos formular un modelo de programación lineal entera para determinar los ciclos óptimos. A continuación transcribimos el modelo para resolver el problema de alocación de capacidades de repuesto para FIPP p-ciclos propuesto por Grover *et al.* [27] en 2005.

Coeficientes:

Δ: constante positiva grande (100 000).

∇: constante positiva pequeña (0,0001).

$\partial_{m,n}$: coeficiente igual a 1 si las relaciones de demandas m y n son «rivales». Esto significa que los caminos definidos para proteger las relaciones de demandas m y n no son mutuamente disjuntos (nodos o aristas).

π_e^p: coeficiente igual a 1 si el ciclo p cruza la arista e, 0 si no.

x_r^p: coeficiente igual a 1 si la relación de demanda r está en el p-ciclo p como *on-cycle*,

2 si es una cuerda y 0 en otro caso.

W: conjunto de relaciones de demandas, indexado por r.

P: conjunto de ciclos elegibles, indexado por p.

Variables:

s_e: capacidad de repuesto ubicada en la arista e.

n^p: cantidad de copias del ciclo p utilizadas como FIPP p-ciclo en la solución.

n_r^p: cantidad de copias unitarias del ciclo p para proteger la relación de demanda r. La integralidad de esta variable se puede relajar sin afectar la integralidad de la solución.

γ_r^p: igual a 1 si el ciclo p protege la relación de demanda r, 0 si no.

Formulación:

$$\min \sum_{e \in E} costo_e . s_e \tag{4.1}$$

sujeto a

$$\begin{aligned}
\sum_{\forall p \in P} x_r^p . n_r^p &\geq d_r && (4.2)\\
n^p &\geq n_r^p, \forall r \in W && (4.3)\\
s_e &\geq \sum_{e \in E} n^p . \pi_e^p, \forall e \in E && (4.4)\\
\gamma_r^p &\geq \nabla . n_r^p, \forall r \in W, \forall p \in P && (4.5)\\
\gamma_r^p &\leq \Delta . n_r^p, \forall r \in W, \forall p \in P && (4.6)\\
\partial_{m,n} + \gamma_m^p + \gamma_n^p &\leq 2, \forall m, n \in W^2 | m \neq n, \forall p \in P && (4.7)
\end{aligned}$$

La función objetivo del modelo busca minimizar el costo total de las capacidades de

repuesto a instalar. La restricción 4.2 obliga a proteger toda la demanda de la relación r; la restricción 4.3 indica que la cantidad de ciclos a ubicar debe ser mayor o igual a la mayor cantidad individual mientras que la 4.4 establece que se debe instalar una cantidad suficiente para poder formar todos los p-ciclos. La restricción 4.7 impide ubicar el mismo p-ciclo para proteger dos demandas «rivales» (los caminos definidos para protegerlas no son mutuamente disjuntos en el sentido de los nodos o de las aristas).

Una versión parecida a este modelo será utilizada como problema maestro para nuestro algoritmo de *Branch & Price*, para esto hacemos las siguientes observaciones:

1. La variable s_j no es estrictamente necesaria si disponemos de los costos de cada ciclo.
2. El parámetro Δ no es necesario (podría valer 1), ya que la restricción 4.6 solo busca anular γ_r^p cuando n_r^p es nulo.
3. Podemos crear la restricción 4.7 solo cuando $\partial_{m,n}$ tome el valor uno.
4. Este modelo considera que si dos relaciones de demanda no son disjuntas no pueden ir en el mismo ciclo, sin importar cuantas repeticiones tengamos. Desde el punto de vista matemático, esto es incorrecto, ya que podríamos tener copias del ciclo para distintas demandas. Asumimos que esto se debe a una restricción de negocio, ya que una de las características más importantes del concepto de FIPP p-ciclo es la integridad de transmisión. Una forma de evitar este inconveniente es la de tener varias copias de cada ciclo en el conjunto P (con costos perturbados para evitar la simetría). En nuestra versión como problema maestro para nuestro algoritmo de *Branch-and-Price* utilizaremos una variable para cada combinación posible de demandas compatibles. Así, nuestro modelo tendrá todavía mas columnas.

Al igual que en el modelo propuesto por Grover *et al.* para el problema SCA de p-ciclos, en el de FIPP p-ciclos la cantidad de ciclos implica también una cantidad exponencial de

variables. Por este motivo, este modelo solo puede ser implementado si el grafo es muy pequeño o muy ralo.

En la próxima sección expondremos nuestra propuesta de modelo de programación lineal entera mixta sin enumeración de ciclos. Este modelo fue inicialmente pensado como modelo para un algoritmo de *pricing* (con $J = 1$ y restricciones adicionales) para un esquema de generación de columnas, pero como en la práctica se requieren pocos ciclos en la solución óptima, decidimos evaluar la eficiencia de este modelo implementado como algoritmo independiente. Entonces, su importancia es doble: por un lado, como modelo para un algoritmo de *pricing*, ya que suelen ser el cuello de botella del esquema de *Branch-and-Price*; y, por otro, como modelo para un algoritmo independiente, que tampoco requiere la enumeración explícita de todos los ciclos.

Es importante recalcar que este tipo de modelos independientes tienen una gran simetría inherente a la formulación por bloques que se puede atenuar muy levemente mediante restricciones de optimalidad.

4.3.1. Modelo PLEM basado en el espacio de ciclos

Los modelos desarrollados en la sección 3.4 para el problema SCA para p-ciclos pueden ser adaptados al problema SCA para FIPP p-ciclos. Dado que el modelo basado en espacio de ciclos es el que deriva en un algoritmo de mayor eficiencia, es el que adaptamos para FIPP. Éste utiliza el concepto de espacio de ciclos (que describimos en la subsección 3.4.3), con restricciones de eliminación de *subtours* de voltaje eléctrico para generar ciclos y utiliza restricciones similares al modelo antes descripto para asignar demandas a ciclos.

Coeficientes:

$\alpha = 1/|V|$: coeficiente para garantizar voltajes compatibles.

B: matriz de ciclos fundamentales del grafo G (μ x m).

C: conjunto de ciclos fundamentales con respecto a un árbol arbitrario.

E_r: conjunto de aristas del camino de la demanda r.

L: número muy grande.

M: matriz de incidencia del grafo G (n x m).

M^t: matriz de incidencia transpuesta del grafo G (m x n).

P: conjunto de ciclos elegibles, indexado por p.

W: conjunto de relaciones de demandas, indexado por r.

Variables:

c_i^j: variable binaria. Toma valor 1 si el ciclo CS_j es generado por el ciclo fundamental i.

p_e^j: variable entera. Es una variable auxiliar que indica cuántos pares de ciclos fundamentales son seleccionados para generar el ciclo j que incluyen a la arista e.

x_{eo}^j: variable binaria. Toma valor 1 si la arista e está en el ciclo generado CS_j. Los arcos son dirigidos y el índice o indica el sentido.

x_o^j: vector de variables x_{eo}^j de tamaño m.

x^j: suma de vectores de variables x_{eo}^j. Es decir, $x^j = x_0^j + x_1^j$.

y_e^j: variable binaria. Toma valor 1 si la arista e es una cuerda (*straddle*) del ciclo generado CS_j.

z_k^j: variable binaria. Solo una de ellas toma valor 1 en cada ciclo generado para indicar el nodo raíz de los voltajes.

$volt_k^j$: variable continua. Indica el voltaje de cada nodo.

γ_r^j: igual a 1 si el ciclo CS_j protege la relación de demanda r, 0 si no.

φ_r^j: variable binaria que indica si la protección del p-ciclo CS_j sobre la demanda r es completamente *straddle* o no.

Formulación:

$$\min \sum_{j\in\{1..J\}} \sum_{e\in E} costo_e(x_{e0}^j + x_{e1}^j) \tag{4.8}$$

sujeto a

$$\begin{aligned}
B.c^j &= 2.p^j + x^j, \forall j \in \{1..J\} && (4.9)\\
x_{e0}^j + x_{e1}^j &\leq 1, \forall e \in E, \forall j \in \{1..J\} && (4.10)\\
\sum_{e=uv\in E} (x_{uv0}^j - x_{uv1}^j) &= 0, \forall u \in V, \forall j \in \{1..J\} && (4.11)\\
Mx^j &\leq 2, \forall j \in \{1..J\} && (4.12)\\
\frac{1}{2}(x_0^j + x_1^j)(\frac{1}{2}M^tM - 2I) &\geq y^j, \forall j \in \{1..J\} && (4.13)\\
\sum_{k\in V} z_k^j &= 1, \forall j \in \{1..J\} && (4.14)\\
volt_h^j - volt_d^j &\geq \alpha x_{eo}^j - (1 - x_{eo}^j) - Lz_h^j && (4.15)\\
&\quad \forall e = (d,h) \in E, \forall o \in 0,1, \forall j \in \{1..J\} && (4.16)\\
\partial_{m,n} + \gamma_m^p + \gamma_n^p &\leq 2, \forall m,n \in W^2 | m \neq n, \forall p \in P && (4.17)\\
\varphi_r^j &\leq \gamma_r^j, \forall r \in W, \forall j \in \{1..J\} && (4.18)\\
\gamma_r^j + \varphi_r^j &\leq x_e^j + 2.y_e^j, \forall e \in E_r, \forall r \in W, \forall j \in \{1..J\} && (4.19)\\
\sum_{j\in\{1..J\}} (\gamma_r^j + \varphi_r^j) &\geq d_r, \forall r \in W && (4.20)
\end{aligned}$$

$$c_i^j \in \{0,1\}\,,\ \forall j \in \{1..J\}\,, i \in C \tag{4.21}$$

$$p_e^j \in \{0,\infty)\,,\ \forall j \in \{1..J\}\,, e \in E \tag{4.22}$$

$$x_e^j \in \{0,1\}\,,\ \forall j \in \{1..J\}\,, e \in E, o \in \{0,1\} \tag{4.23}$$

$$y_e^j \in \{0,1\}\,,\ \forall j \in \{1..J\}\,, e \in E \tag{4.24}$$

$$z_k^j \in \{0,1\}\,,\ \forall j \in \{1..J\}\,, k \in V \tag{4.25}$$

$$volt_k^j \in \{0,\infty)\,,\ \forall j \in \{1..J\}\,, k \in V \tag{4.26}$$

$$\gamma_r^j \in \{0,1\}\,,\ \forall j \in \{1..J\}\,, r \in W \tag{4.27}$$

$$\varphi_r^j \in \{0,1\}\,,\ \forall j \in \{1..J\}\,, r \in W \tag{4.28}$$

Expresamos la restricción 4.17 de forma similar al modelo de Grover *et al.* [27] detallado en la sección 4.3. Pero en la implementación utilizamos una lista de incompatibilidades entre demandas para expresar solo las restricciones necesarias y, además, evitar restricciones duplicadas.

4.3.1.1. Desigualdades válidas:

Las restricciones 3.33, 3.34, 3.35, 3.65 y 3.66 también son desigualdades válidas en esta formulación por los mismos motivos que en las formulaciones de la sección 3.4.

4.3.1.2. Cortes de optimalidad:

Las restricciones 3.31 y 3.32 también son cortes de optimalidad en esta formulación por los mismos motivos que en las formulaciones de la sección 3.4.

4.4. Modelos de programación por restricciones para FIPP

4.4.1. Motivación

Al igual que en la sección 3.5 del capítulo anterior, el desarrollo de un algoritmo de *Branch-and-Price* requiere de un algoritmo de búsqueda de ciclos que mejore una determinada solución. Por otro lado, en la búsqueda de la integralidad, necesitamos asegurarnos de que ciertas columnas no sean propuestas como columnas de mejora debido a que las variables que las representan están acotadas superiormente en un determinado nodo. Los algoritmos de CP manejan eficientemente este requisito. Nosotros proponemos un modelo del tipo constructivo en programación por restricciones para resolver el problema FIPP, es decir, no requiere de la enumeración previa de los ciclos. Al igual que en los modelos vistos anteriormente, construimos J ciclos.

4.4.2. Modelo Espacio de Ciclos

Utilizamos nuevamente el concepto de espacio de ciclos (que describimos en la subsección 3.4.3), es decir que construye los ciclos por unión disjunta de los ciclos fundamentales del grafo. Esta forma de describir los ciclos es excelente para eliminar la simetría, pero requiere de restricciones para eliminar los *subtours*.

Coeficientes:

Q: cantidad máxima de repeticiones de ciclos.

Variables:

$fund_l^j$: variable binaria. Indica si tomamos al ciclo fundamental l en la composición del ciclo j.

c_e^j: variable binaria. Indica si una arista pertenece al ciclo.

s_e^j: variable binaria. Indica si una arista es cuerda del ciclo.

$r^j \in \{0..Q\}$: variable entera. Repetición del ciclo j.

z_k^j: variable binaria. Utilizada para eliminar *subtours*. Indica el nodo que tomamos como referencia para el conteo de los nodos del ciclo.

$u_k^j \in \{0..n\}$: variable entera. Utilizada para eliminar *subtours*. Representa el conteo de nodos.

γ_r^j: variable igual a 1 si el ciclo j protege la relación de demanda r, 0 si no.

φ_r^j: variable binaria que indica si la protección del p-ciclo j sobre la demanda r es completamente *straddle* o no.

Formulación:

$$\min \sum_{j\in\{1..J\}} \sum_{e\in E} costo_e * r^j * c_e^j \tag{4.29}$$

sujeto a

$$\begin{aligned} c_e^j &= \sum_{l\in C} fund_l^j \oplus 2 \\ \forall e \in E, \forall j \in \{1..J\} \end{aligned} \tag{4.30}$$

$$\begin{aligned} c_e^j + s_e^j &\leq 1 \\ \forall e \in E, \forall j \in \{1..J\} \end{aligned} \tag{4.31}$$

$$\begin{aligned} s_e^j \leq \sum_{a\in E|(a.origen=e.origen \text{ OR } a.destino=e.origen) \text{ AND } a\neq e} c_a^j \\ \forall e \in E, \forall j \in \{1..J\} \end{aligned} \tag{4.32}$$

$$s_e^j \leq \sum_{a \in E | (a.origen = e.destino \text{ OR } a.destino = e.destino) \text{ AND } a \neq e} c_a^j \quad \forall e \in E, \forall j \in \{1..J\} \tag{4.33}$$

$$\sum_{k \in V} (u_k^j > 0) = \sum_{e \in E} c_e^j \quad \forall j \in \{1..J\} \tag{4.34}$$

$$\sum_{k \in V} z_k^j = 1 \quad \forall j \in \{1..J\} \tag{4.35}$$

$$z_k^j \leq u_k^j \quad \forall k \in V, \forall j \in \{1..J\} \tag{4.36}$$

$$u_k^j \leq \sum_{e \in E} c_e^j \quad \forall k \in V, \forall j \in \{1..J\} \tag{4.37}$$

$$u_{k_1}^j \neq 0 \Longrightarrow u_{k_1}^j \neq u_{k_2}^j \quad \forall k_1, k_2 \in V | k_1 \neq k_2, \forall j \in \{1..J\} \tag{4.38}$$

$$\begin{aligned} (c_e^j \neq 0) \Longrightarrow \\ ((u_{e.origen}^j + 1 = u_{e.destino}^j) \text{ OR} \\ (u_{e.origen}^j - 1 = u_{e.destino}^j) \text{ OR} \\ (z_{e.origen}^j = 1) \text{ OR } (z_{e.destino}^j = 1)) \geq 1 \\ \forall e \in E, \forall j \in \{1..J\} \end{aligned} \tag{4.39}$$

$$\sum_{e \in E | e.origen = k \text{ OR } e.destino = k} c_e^j \leq 2 \qquad \forall k \in V, \forall j \in \{1..J\} \tag{4.40}$$

$$\sum_{j \in \{1..J\}} r^j <= Q \tag{4.41}$$

$$\partial_{m,n} + \gamma_m^p + \gamma_n^p \leq 2 \qquad \forall m, n \in W^2 | m \neq n, \forall p \in P \tag{4.42}$$

$$\varphi_r^j \leq \gamma_r^j \qquad \forall r \in W, \forall j \in \{1..J\} \tag{4.43}$$

$$\gamma_r^j + \varphi_r^j \leq c_e^j + s_e^j * 2 \qquad \forall e \in E_r, \forall r \in W, \forall j \in \{1..J\} \tag{4.44}$$

$$\sum_{j \in \{1..J\}} r^j * (\gamma_r^j + \varphi_r^j) \geq d_r \qquad \forall r \in W \tag{4.45}$$

La restricción 4.30 establece la relación entre los ciclos fundamentales y los *circs* creados por su unión disjunta.

Una arista no puede ser cuerda y parte del ciclo al mismo tiempo, esto es expresado en la restricción 4.31. Además, para ser una cuerda, necesitamos que sea adyacente a nodos del ciclo. Esto es lo que implican las restricciones 4.32 y 4.33.

Para que no queden *subtours* utilizamos restricciones inspiradas en la formulación MTZ, que etiqueta los nodos del ciclo de forma secuencial, habiendo elegido un nodo inicial. Si bien esta formulación no es buena en PLE porque su relajación no es ajustada, en programación por restricciones esto no nos preocupa, porque solo tenemos que ver el espacio de búsqueda y su simetría. La restricción 4.34 implica que solo utilizamos la variable u_k^j para contar nodos en el ciclo, al igual que la restricción 4.37. La restricción 4.35 determina que podemos elegir un solo nodo inicial, el cual, a su vez, tiene que estar en el ciclo 4.36. Las restricciones 4.38 y 4.39 son equivalentes a la formulación MTZ.

Por otro lado, obligamos a que los grados de los nodos no sean mayores a 2 mediante la restricción 4.40 para achicar el espacio de búsqueda.

Al igual que en todos los modelos anteriores, necesitamos cubrir las demandas 4.45 y acotar la cantidad total de ciclos 4.41.

Expresamos la restricción 4.42 de forma similar a las de programación entera. Pero en la implementación también utilizamos una lista de incompatibilidades entre demandas para expresar solo las restricciones necesarias y, además, evitar restricciones duplicadas.

Adicionalmente, este modelo admite restricciones válidas que permiten acotar el espacio de búsqueda y el dominio de las variables.

$$u_k^j \leq \sum_{e \in E | e.desde = k \text{ OR } e.hasta = k} c_e^j * N \qquad \forall k \in V, \forall j \in \{1..J\} \tag{4.46}$$

$$r^j = 0 \Longrightarrow \sum_{e \in E} c_e^j + s_e^j + \sum_{k \in V} u_k^j + z_k^j = 0 \qquad \forall j \in \{1..J\} \tag{4.47}$$

$$r^j \neq 0 \Longrightarrow \sum_{k \in V} z_k^j = 1 \quad \forall j \in \{1..J\} \tag{4.48}$$

$$z_k^j = 1 \Longrightarrow u_k^j = 1 \quad \forall k \in V, \forall j \in \{1..J\} \tag{4.49}$$

La restricción 4.46 exige que únicamente las variables u de nodos que pertenecen al ciclo tomen valores no nulos.

La restricción 4.47 implica que la variable r^j determina cuando no crear un ciclo. Se combina con que exige cambiar la restricción 4.35 por la restricción 4.48

La restricción 4.49 implica que u tiene que arrancar siempre en z.

4.5. Algoritmo GRASP para el problema FIPP

4.5.1. Motivación

Para poder desarrollar un algoritmo de *Branch-and-Price*, como el que veremos en la sección siguiente, es conveniente partir de una solución inicial factible. Esto se puede lograr fácilmente con los modelos descriptos anteriormente. Sin embargo, también es importante contar con una cantidad reducida de buenos ciclos, que a su vez permitan formar buenas soluciones. Dada la naturaleza iterativa de GRASP, es muy fácil en esta metaheurística retener un conjunto de buenos ciclos para darlos como *pool* inicial al algoritmo de generación de columnas.

Si a la fase de búsqueda local de nuestro algoritmo GRASP la definimos como k-intercambio, es decir respecto de una vecindad que difiere en k ciclos (sin considerar sus repeticiones), entonces los modelos anteriores pueden construir los ciclos óptimos faltantes. Cuanto menor sea el parámetro k, menor la simetría del modelo contructivo y mayor su eficiencia.

En la subsección 3.6.2 del capítulo anterior explicamos brevemente las principales características de esta metaheurística. En el capítulo anterior, en la subsección 3.6.3, hacemos un repaso de las heurísticas disponibles en la bibliografía para obtener p-ciclos.

4.5.2. Esquema de GRASP implementado

Al igual que en la implementación GRASP de la subsección 3.6.4 del capítulo anterior nosotros proponemos una heurística que esta vez llamamos *Cubrimiento - Expansión Rutas - Combinación* (CERC). El algoritmo CERC parte de un conjunto de ciclos que garantiza el cubrimiento de todas las relaciones de demandas y en las sucesivas iteraciones incorporamos a este conjunto los ciclos obtenidos en la búsqueda local. Este conjunto es ampliado mediante combinaciones de sus elementos, generando ciclos que dominan algún

ciclo de él. La LRC es construida a partir de dicho conjunto, es decir que construimos una solución tomando elementos de forma avara y aleatorizada de la LRC. Finalmente, en la fase de búsqueda local, definida como un 1-intercambio, se obtienen los mejores reemplazos de cada ciclo de la solución mediante un modelo PLEM. A continuación, desarrollamos los detalles de cada etapa.

4.5.2.1. Heurística «*Cubrimiento - Expansión Rutas - Combinación*», CERC

La fase constructiva incorpora de a un ciclo por vez a la solución parcial. Este ciclo es obtenido llamando al algoritmo *Cubrimiento - Expansión - Combinación* (CERC). El algoritmo CERC es adaptativo y aleatorizado.

El algoritmo CERC parte de un conjunto de ciclos que garantiza el cubrimiento de todas las relaciones de demandas. Para ello consideramos cada demanda (camino) y el camino (disjunto en el sentido de las aristas y los nodos) más corto entre sus dos nodos extremos, formando así un ciclo que cubre la demanda. Esto no siempre es posible a pesar de que los grafos son 2-conexos. Por ejemplo, en el grafo de la figura 4.2, si la demanda es el camino b-a-c-d, no existe otro camino disjunto. En ese caso obtenemos un ciclo llamando al modelo PLEM utilizado en la fase de búsqueda local.

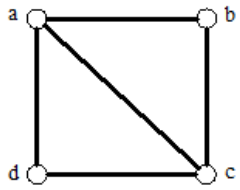

Fig. 4.2: Ejemplo de caminos más cortos disjuntos que no son solución.

En este GRASP, a diferencia del GRASP de la sección 3.6, cada ciclo tiene asociado un *vector de cubrimiento*. Es decir que un determinado ciclo puede pertenecer a la solución más de una vez, pero cubriendo distintas demandas.

Una vez construido un ciclo, que cubre una demanda en particular, CERC verifica la existencia de otras demandas compatibles, que son incorporadas secuencialmente al «vector de cubrimiento del ciclo».

Al conjunto de ciclos obtenidos hasta ese momento, se le incorporan los ciclos obtenidos en las búsquedas locales de la iteraciones anteriores y, finalmente, se le aplica un procedimiento que hace uso de la proposición 3.2.1. Así, dados dos ciclos, verifica si estos comparten una única arista y ningún otro nodo, además de los dos nodos de dicha arista. En ese caso se genera un tercer ciclo por combinación de los dos anteriores. Este procedimiento genera ciclos que a su vez pueden ser comparados contra todos los del conjunto. La cantidad de ciclos totales en el conjunto se limita en la implementación mediante un parámetro.

```
1  procedure CERC(Semilla)
2    inicializar demandaRestante, Solución
3    while demandaRestante ≠ ∅ do
4      inicializar Pool
5      ciclos de cobertura
6      Pool ⟵ ciclos de cobertura ∪ Combinación
7                ∪ ResultadosBúsquedaLocal
8      MejorCiclo ⟵ 0
9      Elección aleatorizada de MejorCiclo de LRC
10     if Búsqueda de dominancia sobre MejorCiclo ← true then
11       MejorCiclo ⟵ NuevoMejorCiclo
12     end if
13     Actualización demandaRestante(MejorCiclo)
14   end while
15 end procedure
```

El próximo paso consiste en elegir el mejor ciclo del conjunto e incorporarlo a la solución

parcial. Para esto, determinamos las demandas restantes que cubre cada ciclo, su costo y su eficiencia. La elección es aleatoria: no se elige el mejor ciclo (en términos de eficiencia), sino que se toma uno al azar de entre una lista restringida de candidatos (LRC), cuyas limitaciones impuestas aseguran que se elija un elemento bueno, aunque no necesariamente el mejor.

Finalmente, el ciclo elegido es analizado en busca de un posible ciclo que lo domine, en ese caso se incorpora el ciclo dominante en lugar del original y se revisa si alguna de las demandas restantes puede ser incluida en el ciclo resultante. Este análisis se limita a buscar dominaciones en el sentido de la proposición 3.2.1.

4.5.3. Búsqueda local

Una vez obtenida una solución completa en la fase constructiva, el algoritmo GRASP implementado pasa a la fase de búsqueda local, definida como 1-intercambio. De esta manera, se obtienen los mejores reemplazos de cada ciclo de la solución, mediante un modelo PLEM. Dicho modelo es una adaptación del modelo de la sección 4.3. Fue resuelto con el algoritmo *Branch-and-Cut* de CPLEX y sus resultados experimentales se muestran en la sección 4.7.

La efectividad de la búsqueda local depende de varios aspectos, tales como la estructura de la vecindad, la técnica de búsqueda sobre dicha vecindad, la evaluación rápida de la función de costos de los vecinos y de la propia solución inicial.

Como ya mencionamos, los modelos constructivos de la sección 4.3 y la sección 4.4 tienen una gran simetría en el espacio de búsqueda por tener que generar una cantidad J de ciclos. Es decir que, durante la búsqueda, los ciclos generados para un j' dado pueden resultar exactamente iguales a los de otro j''. Por el contrario, en la búsqueda local, solo se necesita generar un único ciclo. Esto reduce drásticamente la simetría de estos modelos. La adaptación de los modelos de PLEM antes vistos consiste en fijar $J = 1$ y tomar como

demandas del grafo únicamente a aquellas que no son cubiertas considerando la solución parcial sin el ciclo en análisis. Es decir que los caminos demandados son solo aquellos no cubiertos por el resto de los ciclos de la solución.

```
1 procedure Búsqueda_Local(Solución)
2   for each ciclo en Solución
3     determinar demandas insatisfechas si se saca ciclo
4     nuevoCiclo ⟵ mejor ciclo posible para cubrir esas demandas.
5     if nuevoCiclo no nulo
6       Solución[ciclo] ⟵ 0
7       Solución[nuevoCiclo] ⟵ Solución[nuevoCiclo] + 1
8       Pool ⟵ Pool ∪ nuevoCiclo
9     end if
10  end for
11 end Búsqueda_Local.
```

A continuación detallamos la formulación del modelo de la subsección 4.3.1 adaptado a la búsqueda local de GRASP.

Coeficientes:

$\alpha = 1/|V|$: coeficiente para garantizar voltajes compatibles.

B: matriz de ciclos fundamentales del grafo G (μ x m).

C: conjunto de ciclos fundamentales con respecto a un árbol arbitrario.

L: número muy grande.

M: matriz de incidencia del grafo G (n x m).

M^t: matriz de incidencia transpuesta del grafo G (m x n).

P: conjunto de ciclos elegibles, indexado por p.

W: conjunto de relaciones de demandas, indexado por r.

$\partial_{m,n}$: coeficiente igual a 1 si las relaciones de demandas m y n son «rivales». Esto significa que los caminos definidos para proteger las relaciones de demandas m y n no son mutuamente disjuntos (nodos o aristas).

Variables:

c_i: variable binaria. Toma valor 1 si el ciclo es generado por el ciclo fundamental i.

p_e: variable entera. Es una variable auxiliar que indica cuántos pares de ciclos fundamentales son seleccionados para generar el ciclo que incluyen a la arista e.

x_{eo}: variable binaria. Toma valor 1 si la arista e está en el ciclo generado. Los arcos son dirigidos y el índice o indica el sentido.

y_e: variable binaria. Toma valor 1 si la arista e es una cuerda (*straddle*) del ciclo generado.

z_k: variable binaria. Solo una de ellas toma valor 1 en cada ciclo generado para indicar el nodo raíz de los voltajes.

$volt_k$: variable continua. Indica el voltaje de cada nodo.

γ_r: igual a 1 si el ciclo protege la relación de demanda r, 0 si no.

φ_r: variable binaria que indica si la protección del p-ciclo sobre la demanda r es completamente *straddle* o no.

Formulación:

$$\min \sum_{e \in E} costo_e(x_{e0} + x_{e1}) \tag{4.50}$$

sujeto a

$$\begin{aligned}
B.c &= 2.p + x && (4.51)\\
x_0 + x_1 &\leq 1, \forall e \in E && (4.52)\\
\sum_{e=uv \in E} (x_{uv0} - x_{uv1}) &= 0, \forall u \in V && (4.53)\\
Mx &\leq 2 && (4.54)\\
\frac{1}{2}(x_0 + x_1)(\frac{1}{2}M^t M - 2I) &\geq y && (4.55)\\
\sum_{k \in V} z_k &= 1 && (4.56)\\
volt_h - volt_d &\geq \alpha x_{e,o} - (1 - x_{e,o}) - L z_h && (4.57)\\
& \forall e = (d,h) \in E, \forall o \in \{0,1\} && (4.58)\\
\partial_{m,n} + \gamma_m^p + \gamma_n^p &\leq 2, \forall m,n \in W^2 | m \neq n, \forall p \in P && (4.59)\\
\varphi_r &\leq \gamma_r, \forall r \in W && (4.60)\\
\gamma_r + \varphi_r &\leq x_e + 2.y_e, \forall e \in E_r, \forall r \in W && (4.61)\\
\gamma_r + \varphi_r &\geq d_r, \forall r \in W && (4.62)\\
& c_i \in \{0,1\}, \forall i \in C && (4.63)\\
& p_e \in \{0,\infty), \forall e \in E && (4.64)\\
& x_e \in \{0,1\}, \forall e \in E, o \in \{0,1\} && (4.65)\\
& y_e \in \{0,1\}, \forall e \in E && (4.66)\\
& z_k \in \{0,1\}, \forall k \in V && (4.67)\\
& volt_k \in \{0,\infty), \forall k \in V && (4.68)\\
& \gamma_r \in \{0,1\}, \forall r \in W && (4.69)\\
& \varphi_r \in \{0,1\}, \forall r \in W && (4.70)
\end{aligned}$$

4.6. Algoritmo *Branch-and-Price* para el problema FIPP

4.6.1. Motivación

El modelo más simple, existente en la literatura, para representar el problema FIPP es el descripto por Grover *et al.* [27] que mostramos en la sección 4.3 de este capítulo. En este modelo, las variables n^p representan la cantidad de veces que se utiliza un ciclo del grafo. Se trata de una formulación con una cantidad exponencial de variables. Para instancias pequeñas del problema (en cantidad de ciclos), el modelo es eficiente, ya que las relajaciones son muy buenas.

Un aspecto importante del problema es que las soluciones óptimas involucran muy pocos ciclos con respecto a la cantidad de ciclos totales.

Por otro lado, la resolución del problema SCA para p-ciclos mediante el algoritmo de *Branch-and-Price* que mostramos en la sección 3.7 nos permitió desarrollar las herramientas necesarias para resolver este problema con la misma técnica.

Esto nos llevó a preguntarnos qué eficiencia podríamos lograr resolviendo el modelo de Grover con un algoritmo de *Branch-and-Price*.

4.6.2. Bibliografía

En la literatura encontramos ejemplos de la utilización de algoritmos de *Branch-and-Price* para resolver problemas de diseño de redes de anillos como describen Henningsson *et al.* en el libro [64] capítulo 12. En la subsección 3.7.2 mencionamos algunas referencias para problemas muy relacionados.

En 2009 Grover *et al.* [28] proponen un tipo de p-ciclos para proteger fallas de nodo. En 2012 Jaumard *et al.* [36] también trabajaron centrándose en las fallas de nodos. Para eso, analizaron el problema FIPP y un nuevo tipo de p-ciclos llamados p-ciclos de nodos y

estudiaron la eficiencia de los distintos tipos de p-ciclos.

4.6.3. Problema maestro

Como mencionamos en la subsección 4.6.1, el modelo más simple, existente en la literatura, para representar el problema SCA sobre FIPP p-ciclos es el descripto por Grover *et al.* en [27] y que reproducimos en la sección 4.3 de este capítulo. En este modelo cada variable n^p representa la cantidad de veces que se utiliza un ciclo del grafo. Por lo tanto, se trata de una formulación con una cantidad exponencial de variables. Es un modelo que no se puede descomponer, pero que tiene muy buenas relajaciones lineales. Debido a los inconvenientes de este modelo (descriptos en la sección 4.3), nosotros utilizaremos una versión modificada como problema maestro para nuestro algoritmo de *Branch-and-Price*. Utilizaremos una variable para cada combinación posible de demandas compatibles. Es decir que nuestro modelo tendrá todavía mas columnas. A continuación describimos nuestro modelo.

Coeficientes:

PK: conjunto de ciclos elegibles (indexado por p) y sus combinaciones de demandas compatibles (indexado por k).

W: conjunto de relaciones de demandas, indexado por r.

x_r^{pk}: coeficiente igual a 1 si la relación de demanda r está en el p-ciclo p como *on-cycle*, 2 si todo el camino es por aristas que son cuerdas del p-ciclo (*straddles*) y 0 en otro caso.

Variables:

n^{pk}: cantidad de copias del ciclo p utilizadas como FIPP p-ciclo en la solución, cubriendo la combinación compatible de demandas k.

Formulación:

$$\min \sum_{pk \in PK} costo^{pk}.n^{pk} \tag{4.71}$$

sujeto a

$$\sum_{\forall pk \in PK} x_r^{pk}.n^{pk} \quad \geq \quad d_r, \forall r \in W \tag{4.72}$$

La restricción 4.72 garantiza que se cubran las demandas.

4.6.4. Ramificación

En la subsección 3.7.7 del capítulo anterior vimos algunos de los aspectos que determinan la eficiencia del algoritmo respecto de las decisiones de ramificación. En nuestra implementación, al momento de seleccionar el próximo nodo a resolver, elegimos el nodo de menor cota.

Las variables a elegir para ramificar son las n^{pk}. Estas copias pueden generar ineficiencia debido a la simetría que aporta que una relación de demanda que es compatible en distintas copias solo cambie de una a otra en la ramificación. Nosotros utilizamos ramificación fuerte que consiste en calcular, para cada una de las posibles variables fraccionarias, las cotas que tendrán sus dos problemas hijos y considerar aquella variable con mayor media entre las cotas de sus subproblemas. Es importante aclarar que las soluciones relajadas de nuestro problema no involucran muchas variables, esto hace que la ramificación fuerte no sea costosa en términos de procesamiento.

4.6.5. Proceso de acotación

En nuestra implementación aprovechamos la ramificación fuerte descripta en la subsección anterior para evaluar la cota que tendrán los nodos hijos y compararlas con la mejor solución entera disponible. En caso que la variable elegida para ramificar tenga uno o ambos nodos hijos con cotas mayores a la mejor solución entera se los poda. El gap informado por el algoritmo es obtenido recorriendo toda la lista de nodos pendientes de procesar para determinar la función objetivo mas pequeña (correspondiente al nodo padre) y hacer el cociente entre ella y el valor de la función objetivo de la mejor solución entera encontrada hasta ese momento.

4.6.6. *Pricing*

Los modelos para el problema de *pricing* que utilizamos son los descritos en la sección 4.3 y en la sección 4.4 con las siguientes modificaciones.

- El parámetro J se fija en el valor uno.
- La función objetivo de los problemas de *pricing* consiste en minimizar el costo neto del ciclo propuesto. Es decir que al costo del ciclo le restamos el beneficio de proteger cada relación de demanda. Entonces la función objetivo será:

$$z = \sum_{e \in E} c_e.x_e - \sum_{r \in W} \pi_r(\gamma_r + \varphi_r) \tag{4.73}$$

- La restricción de cubrimiento de demandas se elimina.
- Restricciones de ciclos y combinaciones de relaciones de demandas prohibidos.

Los valores duales a considerar (π_r) para el *pricing* son los relacionados a las restricciones 4.72.

El proceso de *Branch-and-Bound* aplicado al PMR se encarga de obtener la integralidad de la solución. En cada nodo cuya solución relajada sea fraccionaria ramificamos el problema en dos subproblemas. Supongamos que la variable c_{pk}, que representa las repeticiones del ciclo p para el conjunto de relaciones de demanda k en la solución del nodo padre, es la variable fraccionaria elegida para ramificar. El valor de dicha variable en la solución óptima relajada la representamos como c^*_{pk}. Entonces un nodo hijo incorpora la restricción $c_{pk} \leq \lfloor c^*_{pk} \rfloor$ y el otro nodo hijo incorpora la restricción $c_{pk} \geq \lceil c^*_{pk} \rceil$. En el primer caso debemos asegurarnos de que el problema de *pricing* no genere el ciclo p con combinación de demandas k como ciclo-conjunto de demandas recomendado. Por este motivo debemos agregar las *restricciones de ciclos y combinaciones de relaciones de demandas prohibidos*. Esto hace que para cada nivel del árbol de *Branch-and-Bound* se tenga que agregar una restricción extra al problema de *pricing* en uno de los nodos hijos.

Sea el ciclo c_p un ciclo prohibido de longitud λ, Θ_{c_p} el conjunto de relaciones de demandas asociadas a ese ciclo y sea x_e la variable binaria que indica si la arista e es parte del ciclo (*on-cycle*). La proposición 4.6.1 es trivialmente válida.

Proposición 4.6.1. *La restricción 4.74 prohibe que el ciclo-relación de demanda c_{pk} sea solución factible del modelo, sin excluir del espacio de soluciones factibles a ningún otro ciclo-relaciones de demandas.*

$$\sum_{e \in c_p} x_e \leq \lambda_{c_p} - 1 + N. \sum_{r \in \Theta_{c_{pk}}} (1 - \gamma_r) + N. \sum_{r \notin \Theta_{c_{pk}}} \gamma_r \tag{4.74}$$

El proceso de *pricing* es el cuello de botella de la implementación. Esto se debe a que lo resolvemos mediante un algoritmo *Branch-and-Cut* basado en el modelo de PLE y a medida que se avanza en la profundidad del árbol suma las restricciones de ciclos-demandas prohibidos. Esto nos lleva a pensar en formas de limitarlo. Si la suma del doble

de los beneficios es menor que el costo del ciclo de menor costo, entonces no tiene sentido intentar generar más columnas. El ciclo más barato se puede obtener mediante una única corrida del modelo de *pricing* (previa al proceso de *Branch-and-Price*).

En nuestra implementación exigimos al algoritmo de *pricing* que solo devuelva un *ciclo-demanda* cuya función objetivo sea menor a un parámetro f. Esto nos permite la opción de tomar varios ciclos y que todos ellos cumplan con la condición. El parámetro f resulta de mayor importancia respecto del tiempo total del algoritmo de *Branch-and-Price*. Este parámetro también influye mucho respecto del tiempo que requiere todo el proceso para demostrar la optimalidad de una solución.

Siendo que el problema de *pricing* depende de los valores de la solucion dual, en nuestra implementación esta alternativa aumentó notablemente la cantidad de iteraciones de *pricing* necesarias, empeorando el tiempo global del algoritmo.

A continuación detallamos la formulación del modelo de la subsección 4.3.1 adaptado al problema de *pricing* de *Branch-and-Price*.

Coeficientes:

$\alpha = 1/|V|$: coeficiente para garantizar voltajes compatibles.

Ω': conjunto de *ciclos-relaciones de demandas* prohibidos.

B: matriz de ciclos fundamentales del grafo G (μ x m).

C: conjunto de ciclos fundamentales con respecto a un árbol arbitrario.

K: conjunto de todos los ciclos del grafo.

L: número muy grande.

P: conjunto de ciclos elegibles, indexado por p.

M: matriz de incidencia del grafo G (n x m).

M^t: matriz de incidencia transpuesta del grafo G (m x n).

W: conjunto de relaciones de demandas, indexado por r.

Variables:

c_i: variable binaria. Toma valor 1 si el ciclo es generado por el ciclo fundamental i.

p_e: variable entera. Es una variable auxiliar que indica cuántos pares de ciclos fundamentales son seleccionados para generar el ciclo que incluyen a la arista e.

x_{eo}: variable binaria. Toma valor 1 si la arista e está en el ciclo generado. Los arcos son dirigidos y el índice o indica el sentido.

x_o: vector de variables x_{eo} de tamaño m.

x: suma de vectores de variables x_{eo}. Es decir, $x = x_0 + x_1$.

y_e: variable binaria. Toma valor 1 si la arista e es una cuerda (*straddle*) del ciclo generado.

z_k: variable binaria.

$volt_k$: variable continua.

γ_r: igual a 1 si el ciclo protege la relación de demanda r, 0 si no.

φ_r: variable binaria que indica si la protección del p-ciclo sobre la demanda r es completamente *straddle* o no.

Formulación:

$$\min \sum_{e \in E} c_e.x_e - \sum_{r \in W} \pi_r.\gamma_r \tag{4.75}$$

sujeto a

$$\begin{aligned}
B.c &= 2.p + x && (4.76)\\
x_{e0} + x_{e1} &\leq 1, \forall e \in E && (4.77)\\
\sum_{e=uv \in E} (x_{uv0} - x_{uv1}) &= 0, \forall k \in V && (4.78)\\
Mx &\leq 2 && (4.79)\\
\frac{1}{2}(x_0 + x_1)(\frac{1}{2}M^t M - 2I) &\geq y && (4.80)\\
\sum_{k \in V} z_k &= 1 && (4.81)\\
volt_h - volt_d &\geq \alpha x_{eo} - (1 - x_{eo}) - Lz_h && (4.82)\\
&\forall e = (d,h) \in E, \forall o \in \{0,1\} && (4.83)\\
\partial_{m,n} + \gamma_m^p + \gamma_n^p &\leq 2, \forall m,n \in W^2 | m \neq n, \forall p \in P && (4.84)\\
\varphi_r &\leq \gamma_r, \forall r \in W && (4.85)\\
\gamma_r + \varphi_r &\leq x_e + 2.y_e, \forall e \in E_r, \forall r \in W && (4.86)\\
\sum_{e \in c_p} x_e &\leq \lambda_{c_p} - 1 + N.\sum_{r \in K}(1 - \gamma_r) + N.\sum_{r \notin K} \gamma_r, \forall c_p, \in \Omega' && (4.87)
\end{aligned}$$

$$c_i \in \{0,1\}\,,\ \forall i \in C \tag{4.88}$$

$$p_e \in \{0,\infty)\,,\ \forall e \in E \tag{4.89}$$

$$x_e \in \{0,1\}\,,\ \forall e \in E, o \in \{0,1\} \tag{4.90}$$

$$y_e \in \{0,1\}\,,\ \forall e \in E \tag{4.91}$$

$$z_k \in \{0,1\}\,,\ \forall k \in V \tag{4.92}$$

$$volt_k \in \{0,\infty)\,,\ \forall k \in V \tag{4.93}$$

$$\gamma_r \in \{0,1\}\,,\ \forall r \in W \tag{4.94}$$

$$\varphi_r \in \{0,1\}\,,\ \forall r \in W \tag{4.95}$$

El problema de *pricing* depende de los valores de la solución dual del PMR. En nuestra implementación, utilizar el algoritmo de punto interior de CPLEX redujo levemente la cantidad de iteraciones de *pricing* necesarias, mejorando el tiempo global del algoritmo.

4.6.7. Heurística para obtener integralidad

En la mayoría de los nodos del árbol de *Branch-and-Bound* obtendremos soluciones fraccionarias. Mediante una heurística, podemos obtener rápidamente una o varias soluciones enteras. Trivialmente, este problema admite que una solución relajada de PMR se redondee hacía arriba para obtener una solución entera factible. Naturalmente en esta instancia pueden sobrar ciclos. Por este motivo la heurística recorre los ciclos calculando sus costos y demandas cubiertas. Luego ordena los ciclos por costo de más caro a más barato y los recorre en ese sentido para verificar si se lo puede retirar de la solución.

4.6.8. Heurísticas primales

Las heurísticas primales consisten en tomar columnas del PMR con costo reducido nulo e intentar convertirlas en columnas con costo reducido negativo. Estas modificaciones

se llevan a cabo mediante una heurística local similar a la utilizada en la implementación GRASP de la sección 4.5. Esta heurística consiste en recorrer las aristas del ciclo y a analizar la vecindad de ambos nodos para determinar si se puede ampliar el ciclo convirtiendo a la arista en cuestión en *straddle* y el costo reducido del ciclo pase a ser negativo. Luego revisamos si hay otras demandas que se puedan cubrir y revisamos que el *ciclo-demanda* generado no sea igual a uno ya existente en el PMR porque esto generaría problemas de simetría.

4.6.9. Solución inicial

Como ya adelantamos en la sección 4.5, a partir de nuestra implementación de un algoritmo GRASP, obtenemos un conjunto de ciclos buenos que garantizan una solución inicial factible de muy buena calidad. Si bien GRASP maneja una cantidad importante de ciclos, como solución inicial solo tomamos los ciclos de la mejor solución obtenida y los ciclos obtenidos por la búsqueda local exacta.

Igualmente a lo que ocurre en la subsección 4.6.8, es muy importante que GRASP no devuelva ciclos repetidos, porque eso generaría simetría en el árbol de *Branch-and-Bound*.

4.6.10. Pseudocódigo

A continuación detallamos el pseudocódigo del proceso de generación de columnas descripto en la subsección 3.7.3 adaptado a nuestra implementación.

1 **procedure** Generación_de_Columnas
2 Determinar una solución inicial factible del PMR.
3 Inicializar el pool de columnas como vacío.
4 Resolver el PMR actual.
5 Ejecutar heurística de integralidad.
6 Ejecutar heurística primal.
7 Eliminar columnas no básicas con costo reducido muy negativo del PMR.
8 Si el pool de columnas todavía contiene columnas con costo reducido positivo, seleccionar un subconjunto de ellas y agregarlos al PMR e ir al paso 4.
9 Vaciar el pool de columnas.
10 Llamar a la heurística de pricing para generar una o más columnas con costo reducido positivo. Si se generan columnas, agregarlas al pool e ir al paso 8.
11 Llamar al algoritmo exacto de pricing para probar la optimalidad o generar una o más columnas con costo reducido positivo. Si se generan columnas, agregarlas al pool e ir al paso 8.
12 **end** Generación_de_Columnas

4.6.11. Conclusiones del algoritmo *Branch-and-Price*

Si bien el problema FIPP es similar al SCA, las diferencias son grandes a la hora de implementar algoritmos *Branch-and-Price*. La formulación del PM en el caso FIPP tiene muchas más variables y su relajación lineal no es tan ajustada. También es más difícil obtener una buena solución inicial y el algoritmo de *pricing* es más lento. Sin embargo, los resultados experimentales que veremos en la próxima sección son aceptables. Por ejemplo, para la instancia más grande (VZ_US_PIP_021) llegamos al 6 % en menos de 5 minutos.

4.7. Resultados numéricos comparativos para el problema FIPP

A diferencia de lo que sucedió con SCA, en este caso el algoritmo más eficiente resultó ser el *Branch-and-Cut* basado en modelo de PLEM que obtuvo todas las soluciones óptimas en menos de 5 minutos. Con el algoritmo *Branch-and-Price* se obtuvieron buenos resultados, a menos de 1 % del óptimo para la instancia K_{12}, pero, para la instancias VZ_US_PIP_021, a menos del 6 % para el mismo límite de tiempo.

En la próxima subsección detallaremos las principales características de las instancias utilizadas, en la subsección 4.7.2 la parametrización de los algoritmos y, finalmente, en la subsección 4.7.3 los resultados detallados.

4.7.1. Instancias para FIPP

Las topologías de las redes de las instancias son las mismas que para SCA, la diferencia reside en las demandas. Aquí tampoco conocemos el valor óptimo del parámetro J que determinamos heurísticamente.

Las principales características de las instancias son:

Instancia	Nombre de la instancia.
Nodos	Cantidad de nodos.
Aristas	Cantidad de aristas.
Ciclos	Cantidad de ciclos.
Cant. Dem.	Cantidad de caminos demandados.
μ	Dimensión del espacio de ciclos (*i.e.* número ciclomático de G).
Óptimo	En los casos que se indica $\leq$, se trata de la mejor solución encontrada entre todos los algoritmos evaluados.
J	Parámetro J utilizado en los experimentos.

A continuación describimos sus valores.

Instancia	**Nodos**	**Aristas**	**Ciclos**	**Cant. Dem.**	μ	**Óptimo***	**J**
COST_239	11	26	3531	10	16	$ 20.405	5
RED_001	4	5	3	7	2	$ 15.400	6
RED_003	5	7	6	9	3	$ 16.500	6
RED_004	4	6	7	6	3	$ 42	3
Network_001	10	22	833	16	13	$ 83,4	4
K_8	8	28	8018	11	21	$ 18.583	4
K_9	9	36	62814	15	28	$ 10.278	4
K_{10}	10	45	556014	15	36	$ 9.111	4
K_{11}	11	55	5488059	18	45	$ 12.249	3
K_{12}	12	66	59740609	18	55	$ 9.142	3
VZ_US_PIP_001	13	39	106967	11	27	$ 13.450	4
VZ_US_PIP_002	12	35	37286	14	24	$ 19.650	7
VZ_US_PIP_003	12	35	37286	13	24	$ 18.660	6
VZ_US_PIP_004	12	35	37286	11	24	$ 14.920	4
VZ_US_PIP_005	11	32	15341	10	22	$ 7.913	3
VZ_US_PIP_006	11	32	15341	10	22	$ 10.748	2
VZ_US_PIP_007	11	32	15341	13	22	$ 11.903	3
VZ_US_PIP_008	11	32	15341	12	22	$ 12.634	3
VZ_US_PIP_009	11	32	15341	16	22	$ 16.308	4
VZ_US_PIP_010	11	32	15341	13	22	$ 15.378	3
VZ_US_PIP_011	10	27	3354	14	18	$ 25.084	6
VZ_US_PIP_012	9	24	1636	17	16	$ 13.217	3
VZ_US_PIP_013	8	17	167	11	10	$ 11.028	3
VZ_US_PIP_014	7	14	70	14	8	$ 16.858	6
VZ_US_PIP_015	6	10	18	6	5	$ 9.738	4
VZ_US_PIP_016	14	50	?	14	37	$ 23.620	6
VZ_US_PIP_017	14	56	?	14	43	$ 24.655	6
VZ_US_PIP_018	21	67	?	16	47	$ 43.390	6
VZ_US_PIP_019	23	73	?	14	51	$ 41.025	5
VZ_US_PIP_020	25	78	?	16	54	$ 36.585	6
VZ_US_PIP_021	27	85	?	15	59	$ 30.660	6

4.7.2. Algoritmos evaluados y parametrizaciones

Todos los algoritmos evaluados son los propuestos en este trabajo. Se implementaron en C++ de Visual Studio 2012, Concert Technologies e IBM ILOG CPLEX 12.6. El hardware es un procesador intel core i3, 2,20GHz (2 núcleos, 4 procesadores lógicos), 8Gb RAM corriendo bajo Windows 10 Pro.

El modelo de PLEM se resolvió utilizando el algoritmo *Branch-and-Cut* de CPLEX. Las desigualdades válidas y los cortes de optimalidad propuestos están disponibles en todos los nodos como *User Cuts*. CPLEX posee una rutina para administrar automáticamente los cortes provistos por el usuario.

El algoritmo GRASP incluye en su búsqueda local el algoritmo *Branch-and-Cut* de CPLEX para el modelo de PLEM detallado en la subsección 4.5.3. Los experimentos para distintos límites de tiempo son independientes, por lo tanto, en el caso de GRASP, pueden obtener peores resultados con límites mayores.

El algoritmo *Branch-and-Price* también utiliza el algoritmo *Branch-and-Cut* de CPLEX para el modelo PLEM como *pricing* y para el modelo PLE en la heurística de *generación de columnas* que provee la primer solución entera. El modelo del PMR también se resuelve con CPLEX pero en este caso con el algoritmo de programación lineal continua.

Los parámetros de CPLEX son los siguientes:

Algoritmo	**RootAlg**	**NodeSel**	**MIPEmph**	**Prior.**	**SolLim**
PLEM	-	-	4	c, y	-
GRASP. B. Local	-	-	4	-	-
B&P. PMR	4	-	-	-	-
B&P. Pricing	-	-	4	-	1

Entendiendo que:

- **RootAlg**: Algoritmo de programación lineal continua. (-) *Default*; CPLEX elige. 4;

Algoritmo de punto interior.

- **NodeSel**: Selección de próximo nodo cuando realiza un *backtrack*. (-) *Default*; *Depth-first search*; 2 *Best-estimate search*.
- **MIPEmph**: Controla el balance entre velocidad, factibilidad, optimalidad y acotación en PLE. 3; *Emphasize moving best bound*. 4; *Emphasize finding hidden feasible solutions*.
- **Prior.**: Prioridad de variables.
- **SolLim**: Cantidad de soluciones a encontrar antes de detenerse.

Los parámetros se eligieron a partir de una extensa experimentación con las instancias más grandes (reales y artificiales).

El modelo de CP se ejecutó con las opciones por default, con la excepción de la fase de búsqueda que debió ser adaptada a las restricciones para evitar simetría, como se describió oportunamente.

Todos los experimentos se realizaron sobre las instancias descriptas anteriormente en 4.7.1 con límites de tiempos de cálculo de 5, 30 y 120 minutos y *Optimality Gap* en 0 %.

4.7.3. Resultados detallados

Tiempo límite: 5 minutos. Valores de la función objetivo resultante para cada instancia y algoritmo evaluado.

Instancia	PLEM	CP	GRASP	B&P
COST_239	**20.405**	20.815	27.765	20.855
RED_001	**15.400**	**15.400**	**15.400**	**15.400**
RED_003	**16.500**	**16.500**	**16.500**	**16.500**
RED_004	**42**	**42**	**42**	**42**
Network_001	**83,4**	**83,4**	186,1	**83,4**
K_8	**18.583**	18.781	26.228	18.769
K_9	**10.278**	10.464	15.529	**10.278**
K_{10}	**9.111**	10.377	19.487	9.271
K_{11}	**12.249**	14.997	30.555	12.274
K_{12}	**9.142**	10.494	27.070	**9.142**
VZ_US_PIP_001	**13.450**	**13.450**	23.600	13.800
VZ_US_PIP_002	**19.650**	19.800	34.780	20.290
VZ_US_PIP_003	**18.660**	20.665	28.045	19.660
VZ_US_PIP_004	**14.920**	16.240	30.910	15.050
VZ_US_PIP_005	**7.913**	**7.913**	16.917	**7.913**
VZ_US_PIP_006	**10.748**	**10.748**	24.207	**10.748**
VZ_US_PIP_007	**11.903**	12.357	29.457	12.222
VZ_US_PIP_008	**12.634**	12.803	28.829	13.129
VZ_US_PIP_009	**16.308**	17.572	37.467	17.118
VZ_US_PIP_010	**15.378**	15.548	34.632	15.548
VZ_US_PIP_011	**25.084**	25.413	40.843	26.079
VZ_US_PIP_012	**13.217**	**13.217**	25.040	**13.217**
VZ_US_PIP_013	**11.028**	**11.028**	14.563	**11.028**
VZ_US_PIP_014	**16.858**	**16.858**	17.753	**16.858**
VZ_US_PIP_015	**9.738**	**9.738**	12.998	**9.738**
VZ_US_PIP_016	**23.620**	-	39.635	26.410
VZ_US_PIP_017	**24.655**	-	34.615	26.870
VZ_US_PIP_018	**43.390**	-	65.950	44.685
VZ_US_PIP_019	**41.025**	-	82.590	44.975
VZ_US_PIP_020	36.760	-	60.470	**36.585**
VZ_US_PIP_021	**30.660**	-	56.290	32.305

Tiempo límite: 30 minutos. Valores de la función objetivo resultante para cada instancia y algoritmo evaluado.

Instancia	PLEM	CP	GRASP	B&P
COST_239	**20.405**	20.815	24.365	**20.405**
RED_001	**15.400**	**15.400**	**15.400**	**15.400**
RED_003	**16.500**	**16.500**	**16.500**	**16.500**
RED_004	**42**	**42**	**42**	**42**
Network_001	**83,4**	**83,4**	132,5	**83,4**
K_8	**18.583**	18.781	23.120	18.769
K_9	**10.278**	10.464	15.529	**10.278**
K_{10}	**9.111**	10.377	19.487	9.131
K_{11}	**12.249**	14.997	30.555	12.274
K_{12}	**9.142**	10.494	27.070	**9.142**
VZ_US_PIP_001	**13.450**	13.450	23.600	13.800
VZ_US_PIP_002	**19.650**	19.800	34.780	20.290
VZ_US_PIP_003	**18.660**	20.665	28.045	19.345
VZ_US_PIP_004	**14.920**	16.240	30.910	15.050
VZ_US_PIP_005	**7.913**	**7.913**	16.917	**7.913**
VZ_US_PIP_006	**10.748**	**10.748**	24.207	**10.748**
VZ_US_PIP_007	**11.903**	12.357	29.457	12.222
VZ_US_PIP_008	**12.634**	12.803	28.829	**12.634**
VZ_US_PIP_009	**16.308**	17.572	37.467	17.118
VZ_US_PIP_010	**15.378**	15.548	34.632	15.548
VZ_US_PIP_011	**25.084**	25.413	40.843	26.079
VZ_US_PIP_012	**13.217**	**13.217**	25.040	**13.217**
VZ_US_PIP_013	**11.028**	**11.028**	14.563	**11.028**
VZ_US_PIP_014	**16.858**	**16.858**	17.753	**16.858**
VZ_US_PIP_015	**9.738**	**9.738**	12.998	**9.738**
VZ_US_PIP_016	**23.620**	-	39.635	26.410
VZ_US_PIP_017	**24.655**	-	34.615	26.870
VZ_US_PIP_018	**43.390**	-	65.950	44.685
VZ_US_PIP_019	**41.025**	-	82.590	44.975
VZ_US_PIP_020	36.760	-	60.470	**36.585**
VZ_US_PIP_021	**30.660**	-	56.290	32.070

Tiempo límite: 120 minutos. Valores de la función objetivo resultante para cada instancia y algoritmo evaluado.

Instancia	PLEM	CP	GRASP	B&P
COST_239	**20.405**	**20.405**	20.860	**20.405**
RED_001	**15.400**	**15.400**	**15.400**	**15.400**
RED_003	**16.500**	**16.500**	**16.500**	**16.500**
RED_004	**42**	**42**	**42**	**42**
Network_001	**83,4**	**83,4**	92,8	**83,4**
K_8	**18.583**	18.781	23.120	18.769
K_9	**10.278**	10.464	15.529	**10.278**
K_{10}	**9.111**	10.377	19.487	**9.111**
K_{11}	**12.249**	14.997	30.555	12.274
K_{12}	**9.142**	10.494	27.070	**9.142**
VZ_US_PIP_001	**13.450**	**13.450**	23.600	13.800
VZ_US_PIP_002	**19.650**	19.800	34.780	20.150
VZ_US_PIP_003	**18.660**	20.665	28.045	19.345
VZ_US_PIP_004	**14.920**	16.240	30.910	15.050
VZ_US_PIP_005	**7.913**	**7.913**	16.917	**7.913**
VZ_US_PIP_006	**10.748**	**10.748**	24.207	**10.748**
VZ_US_PIP_007	**11.903**	12.357	29.457	12.222
VZ_US_PIP_008	**12.634**	12.803	28.829	**12.634**
VZ_US_PIP_009	**16.308**	17.572	37.467	17.118
VZ_US_PIP_010	**15.378**	15.548	34.632	15.548
VZ_US_PIP_011	**25.084**	25.413	40.843	26.079
VZ_US_PIP_012	**13.217**	**13.217**	25.040	**13.217**
VZ_US_PIP_013	**11.028**	**11.028**	14.563	**11.028**
VZ_US_PIP_014	**16.858**	**16.858**	17.753	**16.858**
VZ_US_PIP_015	**9.738**	**9.738**	12.998	**9.738**
VZ_US_PIP_016	**23.620**	-	39.635	26.410
VZ_US_PIP_017	**24.655**	-	34.615	26.870
VZ_US_PIP_018	**43.390**	-	65.950	44.685
VZ_US_PIP_019	**41.025**	-	82.590	44.975
VZ_US_PIP_020	36.760	-	60.470	**36.585**
VZ_US_PIP_021	**30.660**	-	56.290	31.165

4.8. Conclusiones sobre FIPP

Propusimos un nuevo modelo PLEM, un modelo de CP, una metaheurística GRASP con búsqueda local exacta y un algoritmo **Branch-and-Price** exacto. En este caso, la resolución del modelo PLEM (utilizando **Branch-and-Cut**) resultó ser la más eficiente, obteniendo la solución óptima (o menos del 1 %) dentro de los 5 minutos para todas nuestras instancias. El algoritmo CP resultó competitivo para instancias medianas y chicas, aunque directamente no encuentra soluciones para las instancias grandes. El algoritmo GRASP tiene dificultad para obtener resultados de calidad pero, por su naturaleza, siempre encuentra soluciones. La sorpresa se dió con el algoritmo *Branch-and-Price*, ya que su rendimiento es bueno, pero inferior al **Branch-and-Cut** del modelo PLEM. Esto se puede deber a que su PM tiene muchas más variables, las columnas iniciales no son tan buenas y su relajación lineal no es tan ajustada como en el caso SCA.

5. CONCLUSIONES GENERALES

En este trabajo tratamos los problemas de alocación de capacidades de repuesto (*Spare Capacity Allocation Problem*, SCA) para p-ciclos y FIPP p-ciclos, ambos NP-difíciles. Las topologías de redes de telecomunicaciones basadas en p-ciclos tienen una importancia creciente.

Para el primer problema (SCA) propusimos tres nuevos modelos de programación lineal entera (**PLE**) y entera mixta (**PLEM**) resueltos mediante el algoritmo **Branch-and-Cut** de CPLEX, dos modelos de programación por restricciones (**CP**), una metaheurística *Greedy Randomized Adaptive Search Procedures* (**GRASP**) con búsqueda local exacta y un algoritmo **Branch-and-Price** exacto.

Los modelos de CP, PLE y PLEM no requieren de la enumeración previa de ciclos, se basan en generar una cantidad J de ciclos. Estos no resultaron muy eficientes para las instancias difíciles del problema SCA pero si para el problema FIPP. Utilizamos estos modelos para la fase de búsqueda local de los algoritmos GRASP. Y con otras modificaciones menores (entre ellas fijar $J = 1$), estos modelos fueron usados como modelos para los algoritmos de *pricing* de los algoritmos *Branch-and-Price*. A su vez los algoritmos GRASP aportan las soluciones iniciales de los *Branch-and-Price* ya que su fase constructiva garantiza la disponibilidad de soluciones factibles.

Los algoritmos *Branch-and-Price* utilizan una heurística de generación de columnas como preproceso, que toma la solución inicial del GRASP y devuelve una solución entera de excelente calidad para el problema SCA, pero no tanto para el FIPP. Esta solución es entonces procesada por los algoritmos *Branch-and-Price* exactos.

Las dimensiones y las topologías de redes de EEUU (instancias reales) son de interés práctico ya que fueron obtenidas en base a información pública, interpretada gracias a nuestra experiencia laboral de 2007 a 2012 en el equipo *Global Planning Tools* de *Verizon Business*.

Es interesante ver como para dos problemas relativamente parecidos, las técnicas que mejor funcionaron no fueron las mismas. Para el problema SCA el mejor algoritmo resultó ser el *Branch-and-Price* con el que se obtuvieron excelentes resultados. Menos de 1,5 % del óptimo para las instancias artificiales y menos de 4,5 % para las reales. Para el problema FIPP el algoritmo **Branch-and-Cut** del modelo PLEM resultó ser el más eficiente, obteniendo la solución óptima dentro de los 5 minutos para todas nuestras instancias.

5.1. Trabajo futuro

La diferencia de rendimientos entre los distintos métodos propuestos para ambos problemas nos inducen a pensar que un estudio poliedral podría iluminar este aspecto. A su vez, permitiría desarrollar algoritmos *Branch-and-Cut* específicos, mejorar las soluciones iniciales y los algoritmos de *pricing*.

Respecto de los algoritmos GRASP, se puede trabajar en aumentar el grado de los k-intercambio evaluando grupos de ciclos e implementar técnicas más avanzadas como *Path-Relinking, Reactive GRASP, Cost Perturbations, Bias Functions, Intelligent Construction* y *Filtering Strategies*.

El algoritmo *Branch-and-Price* para FIPP tal vez permita un esquema de *branching* del tipo *Ryan y Foster*. Otras variantes podrían mejorar su eficiencia, como por ejemplo utilizar la heurística de generación de columnas con cierta frecuencia dentro del algoritmo *Branch-and-Price* o combinarlo con *Branch-and-Cut*.

Por último, creemos que es posible desarrollar heurísticas constructivas basadas en los algoritmos de generación de ciclos dado los recientes avances desarrollados por Birmelé *et al.* [12].

Apéndice

A. DETALLES DE IMPLEMENTACIÓN Y ALGORITMOS AUXILIARES

A.1. Generación de ciclos

El problema de enumerar todos los ciclos simples en un grafo ha sido muy estudiado. Hasta 2013, la solución más eficiente era la de Johnson [39] pero ese año Birmelé *et al.* [12] desarrollaron un algoritmo para grafos no dirigidos que es asintóticamente óptimo en el sentido de las operaciones necesarias.

En el presente trabajo implementamos el algoritmo de Johnson y el de Gibb (ver Lee [44]). Este último se basa en espacios de ciclos.

A continuación transcribimos el algoritmo de Johnson que utilizamos en los experimentos.

```
 1 CIRCUIT−FINDING ALGORITHM
 2 begin
 3   integer list array A(n), B(n); logical array blocked (n); integer s;
 4   logical procedure CIRCUIT (integer value v);
 5     begin logical f;
 6       procedure UNBLOCK (integer value u);
 7         begin
 8           blocked (u)← false;
 9           for w ∈ B(u) do
10             begin
11               delete w from B(u);
```

```
12              if blocked(w) then UNBLOCK(w);
13            end
14        end UNBLOCK
15      f ← false;
16      stack v;
17      blocked(v)← true;
18 L1: for w ∈ A_K(v) do
19        if w−s then
20          begin
21            output circuit composed of stack followed by s;
22            f ← true;
23          end
24        else if ¬blocked(w) then
25          if CIRCUIT(w) then f← true;
26 L2: if f then UNBLOCK(v)
27      else for w ∈ A_K(v) do
28          if v ∉ B(w) then put v on B(w);
29      unstack v;
30      CIRCUIT ← f;
31  end CIRCUIT;
32 empty stack;
33 s ← 1;
34 while s < n do
35  begin
36      A_K← adjacency structure of strong component K with least
37      vertex in subgraph of G induced by {s, s+1,..., n};
38      if A_K ≠ ∅ then
39        begin
```

```
40              s ← least vertex in V_k;
41              for i ∈ V_k do
42              begin
43                blocked(i) ← false;
44                B(i) ← ∅;
45              end;
46  L3:     dummy ← CIRCUIT(s);
47              s ← s+1;
48          end
49        else s ← n;
50    end
51  end;
```

A.2. Generación de ciclos fundamentales

Si T es un árbol de cobertura de un grafo G, y e es una arista de G que no pertenece a T, entonces el ciclo fundamental C_e definido por e es el ciclo simple formado por e junto al camino en T que conecta los nodos de e. Por lo tanto, podemos encontrar un conjunto de ciclos fundamentales a partir de un árbol de cobertura.

En el presente trabajo, determinamos diferentes árboles de cobertura (dependiendo del algoritmo), y luego obtenemos conjuntos de ciclos fundamentales asociados. Para obtener el camino dentro de T entre los nodos de una arista no en T, utilizamos el algoritmo de camino más corto de Dijkstra.

En 2009, Kavitha et al. [41] presentan un *survey* sobre bases de ciclos.

A.3. Verificación de la existencia de *subtours*

```
1  procedure VerificarSubtours
2    Calculo grados de los nodos
3    for each v ∈ V
4      if grado[v] > 2 then
5        return TRUE
6      endif
7    pilaNodos ← ∅
8    tour ← 0
9    for each e ∈ Solución
10     if ciclo[u] = 0
11       tour ← tour + 1
12       ciclo[u] ← tour
13       ciclo[v] ← tour
14       pilaNodos ← pilaNodos ∪ u, v
15       while pilaNodos ≠ ∅
16         w ← pilaNodos.pop()
17         for each f = (w, x)|f ∈ Solución
18           if ciclo[x] = 0
19             pilaNodos ← pilaNodos ∪ x
20             ciclo[x] = tour
21   if tour > 1
22     return TRUE
23   else
24     return FALSE
```

El algoritmo implementado consiste en dos pasos. El primero calcula los grados de los nodos lo que permite determinar que no haya *subtours* que tengan nodos en común. El

segundo paso utiliza una pila para marcar los nodos del ciclo y determinar la cantidad de ciclos presentes en la solución.

B. INSTANCIAS

B.1. COST239

La instancia COST239 representa una red de Europa muy utilizada en la bibliografía. El grafo posee 11 nodos, 26 aristas y 3531 ciclos.

1 Copenhagen
2 Londres
3 Ámsterdam
4 Berlín
5 Bruselas
6 Luxemburgo
7 Praga
8 Paris
9 Zurich
10 Viena
11 Milán

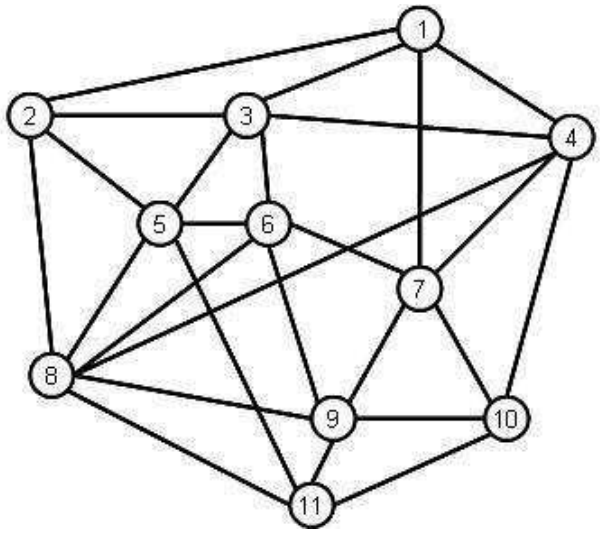

Fig. B.1: Cost239

Arista	Demanda	Costo
Copenhagen-Londres:	0	1310
Copenhagen-Ámsterdam:	5	760
Copenhagen-Berlín:	7	390
Copenhagen-Praga:	0	740
Londres-Ámsterdam:	14	550
Londres-Bruselas:	11	390
Londres-Paris:	13	450
Ámsterdam-Berlín:	26	660
Ámsterdam-Bruselas:	17	210
Ámsterdam-Luxemburgo:	9	390
Berlín-Praga:	37	340
Berlín-Paris:	11	1090
Berlín-Viena:	0	660
Bruselas-Luxemburgo:	18	220
Bruselas-Paris:	13	300
Bruselas-Milán:	0	930
Luxemburgo-Praga:	8	730
Luxemburgo-Paris:	2	400
Luxemburgo-Zurich:	18	350
Praga-Zurich:	24	565
Praga-Viena:	16	320
Paris-Zurich:	8	600
Paris-Milán:	8	820
Zurich-Viena:	5	730
Zurich-Milán:	22	320
Viena-Milán:	3	820

B.2. VZ_US_PIP_001

La instancia VZ_US_PIP_001 representa una red de EEUU obtenida a partir de información pública. El grafo posee 13 nodos, 39 aristas y 106.967 ciclos.

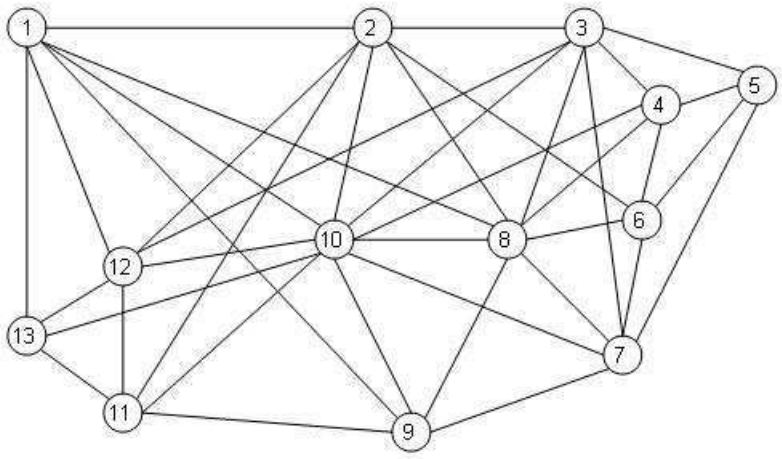

Fig. B.2: VZ_US_PIP_001

Arista	Demanda	Costo	Arista	Demanda	Costo
1-2:	1	1310	4-8:	4	320
1-8:	4	760	4-10:	3	600
1-9:	5	390	5-6:	3	820
1-10:	0	740	5-7:	2	730
1-12:	4	550	6-7:	10	320
1-13:	5	390	6-8:	1	820
2-3:	4	450	7-8:	4	310
2-6:	9	660	7-9:	5	450
2-8:	7	210	7-10:	0	680
2-10:	5	390	8-9:	4	590
2-11:	11	340	8-10:	5	210
2-12:	3	1090	9-10:	9	490
3-4:	5	660	9-11:	7	1780
3-5:	10	220	10-11:	11	2110
3-7:	5	300	10-12:	3	3450
3-8:	0	930	10-13:	5	3610
3-10:	7	730	11-12:	10	660
3-12:	2	400	11-13:	5	110
4-5:	8	350	12-13:	7	210
4-6:	10	565			

C. GLOSARIO DE ACRÓNIMOS

AE	Apriori Efficiency
BLSR	Bi-directional Line Switched Ring
CERC	Cubrimiento - Expansión Rutas - Combinación
CIDA	Capacitated Iterative Design Algorithm
CP	Constraint Programming
DFS	Depth First Search
DSL	Digital Subscriber Line
FEC	Fundamentales - Expansión - Combinación
FIPP	Failure Independent Path Protecting
GRASP	Greedy Randomized Adaptive Search Procedures
IP	Internet Protocol
ISP	Internet Service Provider
JCA	Joint Capacity Allocation
MIP	Mixed Integer Programming
MTZ	Miller-Tucker-Zemlin
OADM	Optical Add Drop Multiplexor
OC-1	Optical Carrier (51,84 Mbit/s)
OC-3	Optical Carrier (155,52 Mbit/s)
OC-12	Optical Carrier (622,08 Mbit/s)
OC-24	Optical Carrier (1244,16 Mbit/s)
OC-48	Optical Carrier (4976,64 Mbit/s)
OC-192	Optical Carrier (9.953,28 Mbit/s)
OC-768	Optical Carrier (39.813,12 Mbit/s)

OSPF	Open Shortest Path First
PLE	Programación Lineal Entera
PLEM	Programación Lineal Entera Mixta
PM	Problema Maestro
PMR	Problema Maestro Restringido
LRC	Lista Restringida de Candidatos
SBPP	Shared Backup Path Protection
SCA	Spare Capacity Allocation Problem
SDH	Synchronous Digital Hierarchy
SHR	Self-Healing Rings
SLA	Straddling Link Algorithm
SONET	Synchronous Optical Networking
TSP	Traveling Salesman Problem
TS	Topological Score

Bibliografía

[1] T. Achterberg, T. Koch, A. Martin. «Branching rules revisited». *Operations Research Letters* 33 (2005) 42 54. 2005.

[2] R. Anbil, R. Tanga y E.L. Johnson. «A global approach to crew-pairing optimization». *IBM Systems Journal*. 1992.

[3] R. Asthana, Y.N. Singh y W.D. Grover. «p-Cycles: An overview», *Communications Surveys & Tutorials*, IEEE , vol.12, no.1, pp.97-111. First Quarter 2010.

[4] A. Atamtürk y D. Rajan. «Partition inequalities for capacitated survivable network design based on directed p-cycles». *Discrete Optimization* Vol. 5, Issue 2, Pages 415-433. 2008.

[5] A. Atamtürk y D. Rajan. «A new model for designing survivable networks». *Department of Industrial Engineering and Operations Research University of California, Berkeley.*

[6] A. Atamtürk y O. Günlük. «Network design arc set with variable upperbounds». *Networks. Vol. 50 , Issue 1* Pages: 17 - 28. 2007.

[7] A. Atamtürk. «On capacitated network design cut-set polyhedra». *Mathematical Programming* Springer. Berlin. Heidelberg. Vol. 92, No. 3. 2002.

[8] D. Baloukov, W. D. Grover y A. Kodian. «Toward jointly optimized design of failure independent path-protecting p-cycle networks». *Journal of Optical Networking*, Vol. 7, Issue 1, pp. 62-79. 2008.

[9] C. Barnhart, E. Johnson, G. Nemhauser, M. W. P. Savelsbergh y P. Vance. «Branch and Price: Column Generation for Solving Huge Integer Programs». School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta. 1994.

[10] M. Batista y M. Belén. «Optimización metaheurística para la planificación de redes WDM». Departamento de Estadística, Investigación Operativa y Computación. Universidad de la Laguna. 2003.

[11] G.J. Bezem y J. van Leeuwen. «Enumeration in graphs». *Technical Report* RUU-CS-87-07, Utrecht University, 1987.

[12] E. Birmele, R. Ferreira, R. Grossi, A. Marino, N. Pisanti, R. Rizzi y G. Sacomoto. «Optimal Listing of Cycles and st-Paths in Undirected Graphs». *ACM-SIAM Symposium on Discrete Algorithms*. 2013.

[13] R.E. Bixby, J.W. Gregory, I.J. Lustig, R.E. Marsten y D.F. Shanno. «Very large-scale linear programming: a case study in combining interior point and simplex methods». *Operations Research*. 1992.

[14] I. Charon y O. Hudry. «The noising method: A new method for combinatorial optimization». *Operations Research Letters*, 14:133-137. 1993.

[15] E. Delgadillo. «Modelos y algoritmos para diseño de redes de comunicaciones con requisitos de supervivencia». Tesis de licenciatura en Ciencias de la Computación. FCEyN UBA. 2013.

[16] R. Diestel. «Graph Theory». Springer-Verlag New York 1997. 2000.

[17] J. Doucette, D. He, W.D. Grover y O. Yang. «Algorithmic approaches for efficient enumeration of candidate p-cycles and capacitated p-cycle network design», *Design of Reliable Communication Networks*, 2003. (DRCN 2003). Proceedings. Fourth International Workshop on , vol., no., pp. 212- 220, 19-22. Oct. 2003

[18] J. Doucette y W.D. Grover. «Capacity Design Studies of span-restorable mesh networks with shared-risk link group (SRLG) effects». *Optical Networking and Communications Conference (OptiComm 2002)*, pages 25-38, Boston, MA. Julio/Agosto 2002.

[19] M. Ferreira y T. Gomes. «Candidate cycles generation for p-cycle calculation considering SRLGs». *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, 3rd International Congress on. págs 1 - 8. 2011.

[20] M. R. Garey y D. S. Johnson. «Computers and Intractability: A Guide to the Theory of NP-Completeness». W. H. Freeman and Company, New York. 1979.

[21] B. Gavish y S. C. Graves. «The Travelling Salesman Problem and Related Problems». Operations Research Center Working Paper; OR 078-78. 1978.

[22] D. Gensch. «An industrial application of the Traveling Salesman Subtour Problem». *AIIE Transactions*, 10(4):362-370. 1978.

[23] A. J. Glenstrup. «Optimised Design and Analysis of All-Optical Networks». PhD thesis. Research Center Com. 2002.

[24] J. L. González. «Optimización heurística y redes neuronales en dirección de operaciones e ingeniería». *Adenso Diaz Editor*. Editorial Paraninfo. 1996.

[25] Gurobi. [Online] http://www.gurobi.com/

[26] W.D. Grover. «Mesh-based Survivable Networks: Options for Optical", MPLS, SONET and ATM Networking», Prentice-Hall, Aug. 2003.

[27] W. D. Grover y A. Kodian. «Failure-independent path-protecting p-cycles: Efficient and simple fully preconnected optical-path protection». *Journal of Lightwave Technology* , Vol. 23, No. 10. October 2005.

[28] W. D. Grover y D. Onguetou. «A new approach to node-failure protection with span-protecting p-cycles». *11th International Conference on Transparent Optical Networks (ICTON)*, pp. 1-5. 2009.

[29] W.D. Grover y D. Stamatelakis. «Cycle-oriented distributed preconfiguration: ring-like speed with mesh-like capacity for self-planning network restoration», *Communications*, 1998. ICC 98. Conference Record. 1998 IEEE International Conference on, vol.1, no., pp.537-543 vol.1, 7-11. Jun 1998.

[30] C. G. Gruber y D. A. Schupke. «Capacity-efficient Planning of Resilient Networks with p-Cycles». Munich University of Technology. Institute of Communication Networks. Munich, Germany. 2002.

[31] M. Henningsson y K. Holmberg. «A ring generation problem based on the traveling salesman sub tour problem». Research Report LiTH-MAT-R-2003-19, Department of Mathematics, Linköping Institute of Technology, Sweden. 2003.

[32] H. Hwang, S. Y. Ahn, Y. H. Yoo y S. K. Chong. «Multiple shared backup cycles for survivable optical networks». in Proc. ICCCN01, Scottsdale, AZ, pp. 284289. Oct. 2001.

[33] J. D. Horton. «A polynomial-time algorithm to find the shortest cycle basis of a graph». SIAM J. COMPUT. Vol. 16, No. 2. April 1987.

[34] IBM, IBM ILOG CPLEX optimizer version 12.6.1, 2014. [Online] http://www-01.ibm.com/ software/commerce/optimization/cplex-optimizer/

[35] B. Jaumard, C. Rocha, D. Baloukov y W.D. Grover. «A column-generation approach for design of networks using path-protecting p-cycles». *Proceedings of 6th Workshop on Design of Reliable Communication Networks*. 2007.

[36] B. Jaumard, H. Li y C. Rocha. «Design of Efficient Node p-cycles in WDM Mesh Networks». *Les Cahiers du GERAD*. GERAD. 2012.

[37] B. Jaumard, H. A. Hoang y D. T. Kien. «Robust FIPP p-cycles against dual link failures». *Telecommunication Systems*, Vol. 56, Issue 1, pp 157-168. 2014.

[38] B. Jaumard y H. A. Hoang . «A new flow formulation for FIPP p-cycle protection subject to multiple link failures». *Ultra Modern Telecommunications and Control Systems and Workshops* (ICUMT), 3rd International Congress on. págs 1-7. 2011.

[39] D. B. Johnson. «Finding all the elementary circuits of a directed graph». *SIAM Journal Computing*. Vol. 4, No. 1. 1975.

[40] D. Jungnickel. «Graphs, Networks and Algorithms». Springer-Verlag Berlin Heidelberg. 1999.

[41] T. Kavitha, C. Liebchen, K. Mehlhorn, D. Michail, R. Rizzi, T. Ueckerdt, K. A. Zweig. «Cycle Bases in Graphs Characterization, Algorithms, Complexity, and Applications». Computer Science Review Vol. 3, Issue 4, Pages 199243. 2009.

[42] A. Kodian, A. Sack, W. D. Grover. «p-cycle network design with hop limits and circumference limits». *Broadband Networks, 2004. BroadNets 2004. Proceedings.* First International Conference on , vol., no., pp. 244- 253, 25-29 Oct. 2004.

[43] A. Kodian, W. D. Grover. «Failure-independent path-protecting p-cycles efficient and simple fully preconnected optical-path protection». *Journal of Lightwave Technology* 23 (10) 3241-3259. 2005.

[44] B. C. Lee. «Algorithmic approaches to circuit enumeration problemns and applications». FTL REPORT R82-7. Department of Aeronautics and Astronautics. MIT. 1982.

[45] C. Liu y L. Ruan. «Finding good candidate cycles for efficient p-Cycle network design». *13th International Conference on Computer Communications and Networks*. ICCCN 2004. Proceedings. 2004.

[46] Y. Liu, D. Tipper y P Siripongwutikorn. «Approximating Optimal Spare Capacity Allocation by Successive Survivable Routing». Department of Information Science and Telecommunications. University of Pittsburgh. 2004.

[47] M. E. Lübbecke y J. Desrosiers. «Selected Topics in Column Generation». *Operations Research*. Vol. 53, No. 6, NovemberDecember 2005, pp. 10071023. 2005.

[48] V. Mak y T. Thomadsen. «Polyhedral combinatorics of the cardinality constrained quadratic knapsack problem and the quadratic selective travelling salesman problem». *Journal of Combinatorial Optimisation*. 11.421-434. 2006

[49] P. Mateti y N. Deo. «On algorithms for enumerating all circuits of a graph». *SIAM Journal on Computing*. Vol. 5, No. 1. March 1976.

[50] J. Mittenthal y C.E. Noon. «An insert-delete heuristic for the traveling salesman subset-tour problem with one additional cosntraint». *Journal of the Operational Research Society*, 43, 277-283. 1992.

[51] D. P. Onguetou y W. D. Grover. «Approaches to p-cycle network design with controlled optical path lenghts in the restored network state». *Journal of Optical Networking*, Vol. 7, Issue 7, pp. 673-691. 2008.

[52] D. P. Onguetou y W. D. Grover. «p-cycle network design: From fewest in number to smallest in size». *Design and Reliable Communication Networks*. DRCN 2007. 6th International Workshop on , vol., no., pp.1-8, 7-10 Oct. 2007.

[53] D. P. Onguetou. «p-Cycles: New Solutions for Node Protection, Transparency, and Large Scale Network Design». PhD Thesis. Department of Electrical and Computer Engineering. University of Alberta. 2011.

[54] K. Paton. «An algorithm for finding a fundamental set of cycles of a graph», Comm. ACM 12, pp. 514-518. 1969.

[55] A. Pecorari e I. Loiseau. «Algorithms and Models for Survivable Network Design using P-Cycles». *LAND-TRANSLOG*. Puerto Varas. Chile. 2011.

[56] A. Pecorari e I. Loiseau. «Algorithms and Models for P-Cycle Design Without Cycle Enumeration». *EPIO*. Argentina. 2012.

[57] A. Pecorari e I. Loiseau. «Survivable network design using FIPP p-cycle». *ALIO - INFORMS*. Buenos Aires, Argentina. 2010

[58] A. Pecorari e I. Loiseau. «Algoritmos de Generación de Columnas para el problema de Diseño de Redes de Telecomunicaciones Basadas en FIPP p-ciclos». *JAIIO*. Buenos Aires, Argentina. 2014.

[59] A. Pecorari e I. Loiseau. «Models and heuristics for p-cycle networks design». *CLAIO-SBPO*. Rio de Janeiro, Brazil. 2012.

[60] D. Rajan y A. Atamtürk. «Survivable network design: routing of flows and slacks». *Telecommunications Network Design and Management, G. Anandalingam and S. Raghavan (eds.)*, 65-82, Kluwer Academic Publishers. 2002.

[61] A. Ranjbar y C. Assi. Availability-aware design in FIPP p-cycles protected mesh networks. *Optical Network Design and Modeling. ONDM 2008. International Conference on*. 2008.

[62] V. V. Bapeswara Rao y V. G. K. Murti. «Enumeration of all circuits of a graph». *Proceedings of the IEEE*. 1969.

[63] M. G.C. Resende y C. C. Ribeiro. «Greedy Randomized Adaptive Search Procedures». *State of the Art Handbook in Metaheuristics*, F. Glover and G. Kochenberger, eds., Kluwer. 2002.

[64] M.G.C. Resende y Panos M. Pardalos Editors. «Handbook of Optimization on Telecommunications». *Springer*. 2005.

[65] F. Rossi, P. van Beek y T. Walsh (Editores). «Handbook of Constraint Programming». 2006.

[66] C. Rocha, B. Jaumard. «Revisiting p-cycles. FIPP p-cycles vs. Shared Link Path Protection». *17th International Conference on Computer Communications and Networks (ICCCN)*. 2008.

[67] C. Rocha, B. Jaumard, T. Stidsen. «Efficient computation of FIPP p-cycles». *Telecommunication Systems*. 2012.

[68] C. Rocha, B. Jaumard y T. Stidsen. «Obtaining lower bounds for the FIPP p-cycle problem». *XLII SBPO*. 2009.

[69] D. M. Ryan y B. A. Foster. «An integer programming approach to scheduling». A. Wren, ed. *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*. North-Holland, Amsterdam, The Netherlands, 269280. 1981.

[70] M. Ryu y H. Park. «Survivable network design using restricted p-cycle». *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 918-927. 2003.

[71] J. J. Salazar González. «Programación matemática». Diaz de Santos. 2001.

[72] D. A. Schupke. «An ILP for optimal p-cycle selection without cycle enumeration». *in Proc. of the Eighth Working Conference on Optical Network Design and Modelling (ONDM)*. February 2004.

[73] SCIP. Zuse Institute Berlin. [Online] http://scip.zib.de/

[74] M.C. Sinclair. «Minimum cost topology optimisation of the COST 239 European optical network». Dept. of Electronic Systems Engineering, University of Essex. 1995.

[75] M. Sol y M.W.P. Savelsbergh. «A Branch and Price Algorithm for the Pickup and Delivery Problem with Time Windows». *Report COC94-06*. Georgia Institute of Technology. Atlanta. Georgia. 1994.

[76] A. Smutnicki y K. Walkowiak. «A heuristic approach to working and spare capacity optimization for survivable anycast streaming protected by p-cycles». Telecommunication Systems. May 2014, Vol. 56, Issue 1, pp 141-156. 2014.

[77] A. Smutnicki y K. Walkowiak. «A new approach to optimization of p-cycle protected Multicast optical networks». *Ultra Modern Telecommunications and Control Systems and Workshops* (ICUMT), 5th International Congress on. págs 74 - 81. 2013.

[78] A. Smutnicki y K. Walkowiak. «Joint working and spare capacity assignment for anycast streaming in survivable networks protected by p-Cycles». *Ultra Modern Telecommunications and Control Systems and Workshops* (ICUMT), 3rd International Congress on. págs 1-6. 2011.

[79] A. Smutnicki y K. Walkowiak. «Optimal results for Anycast-Protecting p-Cycles problem». *Ultra Modern Telecommunications and Control Systems and Workshops* (ICUMT) International Congress on. págs 511 - 517. 2010.

[80] T. Stidsen y T. Thomadsen. «Joint optimization of working and p-cycle protection capacity». *Technical report*. Informatics and Mathematical Modelling, Technical University of Denmark, DTU. 2004.

[81] R. Tarjan. «Enumeration of the elementary circuits of a directed graph». Department of Computer Science. Cornell University. September 1972.

[82] A. Westerlund. «Decomposition schemes for the traveling salesman subtour problem». Linköping Studies in Science and Technology. Theses No 939. Licentiate thesis, LiU-TEK-LIC-2002:12. 2002.

[83] L.A. Wolsey. «Integer Programming». Ed. John Wiley & Sons, New York. 1998.

[84] B. Wu, K. L. Yeung y P. Ho. «ILP formulations for p-cycle design without candidate cycle enumeration». *IEEE ICC '07, Glasgow, UK, Jun. 2007, and the IEEE GlobeCom '07*, Washington D. C., USA, Nov. 2007.

[85] B. Wu, K. L. Yeung y P. Ho. «Preconfigured structures for survivable WDM networks». *Proceedings Of The 2008 International Conference On Advanced Infocomm Technology*, Icait '08. 2008.

[86] Z. Zhang, W. Zhong y B. Mukherjee. «A heuristic method for design of survivable WDM networks with p-cycles». *Communications Letters, IEEE* , vol.8, no.7, pp. 467-469. July 2004.

[87] H. Zhang y O. Yang. «Finding protection cycles in DWDM networks». *In Proceedings of IEEE International Conference in Communications (ICC 2002)*, vol. 5, págs 2756-2760, NY. Abril/Mayo 2002.