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

Un algoritmo branch-and-cut para el routing and spectrum allocation problem

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

Lic. Marcelo Luis Bianchetti

Director: Dr. Javier Leonardo Marenco

Consejera de estudios: Dra. Flavia Bonomo

Lugar de trabajo: Universidad Nacional de General Sarmiento, Buenos Aires

Fecha de la defensa: 1 de Septiembre de 2022

Resumen

La fibra óptica flexible y en particular la tecnología llamada *grilla flexible* (flexgrid), especificada en el estándar ITU-T G.694.1, es una de las soluciones más prometedoras para lidiar con el enorme crecimiento del tráfico en redes muy grandes. En dicha especificación, el espectro de frecuencia de un cable de fibra óptica es dividido en canales más angostos, denominados *slots*. Cualquier secuencia consecutiva de *slots* puede ser usada como un solo canal. A la conexión que utiliza dicho canal a largo de una ruta a través de los nodos de la red se la denomina *lightpath*.

Dado un conjunto de demandas punto-a-punto, al problema de establecer los *lightpaths* para satisfacerlas se denomina *problema de Ruteo y Asignación de Espectro (RSA)*. Dada su relevancia, este problema ha sido estudiado intensivamente en la última década. Se demostró que pertenece a la clase $\mathcal{NP}$-dificil, y en la práctica se comprobó que es necesario aplicar técnicas computacionales no triviales para obtener soluciones de instancias reales.

En la última década, aplicando técnicas de programación entera se ha logrado resolver bien en la práctica una gran cantidad de problemas de optimización combinatoria. El principal objeto del presente trabajo es ampliar dicho campo, mediante la aplicación de técnicas de programación lineal entera sobre el RSA. El primer paso es explorar varios modelos de programación lineal entera para el RSA, analizando su efectividad en instancias conocidas. Recurrimos a varias técnicas de modelado, con el fin de encontrar formulaciones naturales de este problema. Una vez comparados estos modelos, analizamos en profundidad uno de los más prometedores para comprender sus características, fortalezas y debilidades, teniendo en cuenta sus propiedades combinatorias. A partir de los resultados de dicho estudio, desarrollamos procedimientos computacionales basados en programación lineal entera para el RSA, incluyendo el uso de planos de corte y heurísticas primales, con el objetivo final de resolver de manera ópima o casi óptima instancias reales de este problema.

Este trabajo también contiene el desarrollo de un generador de instancias basado en topologías reales, y la compilación bibliográfica más extensa hasta la fecha enumerando y clasificando los trabajos que tratan el RSA y sus variaciones más cercanas desde la óptica de la optimización combinatoria.

Keywords: programación lineal entera, investigación operativa, redes de fibra óptica, desigualdades válidas, heurísticas iniciales.

Abstract

A branch and cut algorithm for the routing and spectrum allocation problem

One of the most promising solutions to deal with huge data traffic demands in large communication networks is given by flexible optical networking, in particular the *flexible grid* (flexgrid) technology specified in the ITU-T standard G.694.1. In this specification, the frequency spectrum of an optical fiber link is divided into narrow frequency *slots*. Any sequence of consecutive slots can be used as a simple channel, and such a channel can be switched in the network nodes to create a *lightpath*.

In this kind of networks, the problem of establishing lightpaths for a set of end-to-end demands that compete for spectrum resources is called the *Routing and Spectrum Allocation problem* (RSA). Due to its relevance, this problem has been intensively studied in the last decade. It has been shown to be $\mathcal{NP}$-hard, and nontrivial computational techniques are to be applied in order to tackle the solution of practical instances.

Since integer programming techniques are known to provide successful practical approaches for several combinatorial optimization problems, the aim of this work is to further the exploration of such an issue, by applying integer linear programming techniques to RSA. The first objective is to explore several integer programming models for RSA, analyzing their effectiveness over known instances. We resort to several modeling techniques in order to find natural formulations of this problem. Having compared these models, then we make a detailed analysis of one of the most promising of them in order to understand its properties, strengths, and weaknesses, taking combinatorial properties into account. Based on these explorations, we develop integer linear programming based computational procedures for RSA including the use of cutting planes and primal heuristics, with the final objective of solving to optimality or near-optimality real-world instances of RSA.

This work also contains the development of an instance generator based on real topologies, and the most extensive survey to date listing and classifying the works that treat the RSA and its closest variations from the point of view of combinatorial optimization.

Keywords: integer linear programming, operational research, optical fiber networks, valid inequalities, initial heuristics.

Índice general

1 Introducción

Las redes ópticas representan una infraestructura crucial para nuestro sistema de comunicaciones moderno. Estas redes utilizan la luz como medio de comunicación entre un nodo emisor y un nodo receptor. El principal problema a resolver con esta tecnología es cómo satisfacer el ancho de banda requerido entre cada par de nodos. Durante las últimas décadas, los operadores han estimado un crecimiento continuo de más del 35 por ciento anual en las redes troncales, derivado de factores que incluyen internet móvil, video de alta resolución y un mayor uso de servicios basados en la nube. Esto parece que continuará con la demanda proveniente del internet de las cosas, el despliegue ubicuo de sensores y las comunicaciones punto a punto. Sin embargo, la tecnología no mejora tan rápidamente como para satisfacer el crecimiento de la demanda, por lo que, con cada avance que logra aumentar la capacidad de las redes, surge la necesidad de hacer un mejor uso de los recursos. El problema estudiado en el presente trabajo intenta resolver esta última tarea para la tecnología actual, como parte de un proyecto en colaboración con instituciones de Francia, Brasil, Ecuador, Chile y Argentina.

En el actual capítulo presentamos un breve resumen del camino seguido por las tecnologías utilizadas desde la aparición de la fibra óptica hasta el surgimiento del problema estudiado en esta tesis. Después de presentar formalmente este problema contando brevemente algunas de sus variantes más cercanas, realizamos una revisión teórica de las técnicas que utilizamos y revisamos brevemente la literatura relacionada. Finalmente, presentamos un estudio formal de su complejidad computacional.

1.1. Presentación del problema

Las tecnologías para dar solución a los problemas tanto de utilización de recursos como de ampliación de la capacidad de las redes han ido cambiando durante las últimas décadas [123]. En las siguientes secciones presentamos brevemente el camino recorrido desde la aparición de la fibra óptica hasta encontrarnos con el problema que estudiamos en detalle a lo largo de esta tesis.

1.1.1. Evolución de las tecnologías: de la WDM a la OFDM

Desde principios de los 90's, y a lo largo de dos décadas, la *multiplexación por división de longitud de onda* (WDM, del inglés *wavelength division multiplexing*) ha sido la tecnología más popular en las comunicaciones por fibra óptica. Esta tecnología, que reemplazó a la *jerarquía digital sincrónica* (SDH, del inglés *synchronous digital hierarchy*) la cual venía siendo utilizada desde finales de la década de 1980, combina múltiples longitudes de onda, que también llamaremos *wavelengths*, para transportar simultáneamente señales a través de una sola fibra óptica y, por lo tanto, permite una multiplicación de la capacidad de la red. Esta capacidad fue incrementada aún más mediante la implementación de un espaciado fijo del canal espectral, *i.e.*, una grilla fija [160].

Sin embargo, en esta tecnología suele haber una única forma de satisfacer una demanda o solicitud determinada. La tasa de bits del *wavelength* se selecciona de una grilla fija de frecuencias especificadas por la *unión internacional de telecomunicaciones* (UIT). De este modo tanto el alcance óptico como el espectro son fijos. Este problema de asignar frecuencias a demandas en redes fijas que se conoce como problema de *ruteo y asignación de wavelength* (RWA, del inglés *routing and wavelength assignment*) tiene su principal inconveniente en el hecho de que una demanda puede ocupar menos de un *wavelength* completo, o más de uno requiriendo múltiples *wavelengths*, desperdiciando capacidad en ambos casos. En general, esto conduce a un uso ineficaz de los recursos espectrales. Como se señala en [49], tiene sentido *dimensionar correctamente* el espectro para cada demanda en función de su tasa de bits y la distancia de transmisión (en lugar de forzar a todas las demandas a utilizar más espectro).

A fines de la década de 2000, en respuesta a esa objeción y a causa del crecimiento sostenido de los volúmenes de tráfico de datos, las *redes ópticas elásticas* (EON, del inglés *Elastic Optical Networks*) [49, 62] sugirieron aumentar la capacidad de transmisión a la vez que se hacía un mejor uso de los recursos mediante la utilización de una red flexible, abreviada como *flexgrid*. La Figura 1.1 [93] muestra la arquitectura de este tipo de redes, que consta de dos componentes principales:

- *bandwidth variable wavelength cross-connections* (BV-WXCs), que son los nodos de la red y permiten a la misma asignar recursos de manera flexible; y
- *bandwidth variable transponders* (BV-Transponders), que generan las señales ópticas de acuerdo con la demanda de tráfico, lo que les permite generar un uso eficiente del espectro disponible.

Las redes elásticas, basadas en la multiplexación por división de longitud de onda de Nyquist o multiplexación por división de frecuencia ortogonal óptica (OFDM, del inglés *orthogonal frequency-division multiplexing*) [171], pueden utilizar eficientemente el ancho de banda de la fibra óptica de una manera elástica dividiéndolo en cientos o incluso miles de *subportadoras* –conocidas como *slots de frecuencia* o simplemente *slots*– de ancho de banda fijo. Esto permite ofrecer diferentes velocidades de transmisión formando *canales* sobre múltiples subportadoras. Es posible obtener diferentes tasas de bits adaptando el formato de modulación al alcance óptico o aumentando el número de portadoras por canal. Diferentes subportadoras pueden adoptar una variedad de esquemas de modulación y dar como resultado diferentes tasas de bits. Cada canal puede conmutarse en los nodos de la red para crear un *lightpath* (es decir, una conexión óptica).

Siguiendo este enfoque, el espectro elástico se puede representar utilizando una serie de *slots* de frecuencia contiguos como se muestra en la Figura 1.2.

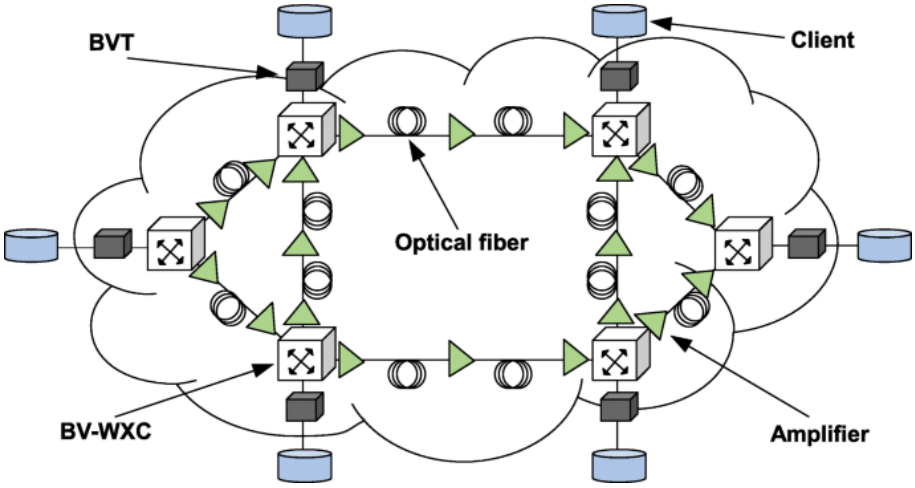


Fig. 1.1: Arquitectura de una red óptica elástica (EON) [93].

1.1.2. El problema de ruteo y asignación de espectro

Las redes ópticas elásticas parecen ser indispensables para lidiar con la siempre creciente demanda de ancho de banda debido a su gran cantidad de buenas propiedades, que incluyen tasa de datos y asignación de espectro flexible, baja atenuación y distorsión de señal, bajo requerimiento de energía, bajo uso de material, poco requerimiento de espacio, y bajo costo [26].

Dicha red traduce el problema de ruteo y asignación de *wavelengths* en el de *ruteo y asignación de espectro* (RSA, del inglés *routing and spectrum allocation*) [32, 62] que consiste en establecer *lightpaths* para un conjunto de demandas de extremo a extremo. En la versión simplificada estudiada en el presente trabajo se asume el mismo formato de modulación para cada demanda, por lo que éstas se expresan en términos de la cantidad de *slots* requeridos. Dado que los *lightpaths* están determinados por una ruta y un canal que satisface el volumen, el RSA implica encontrar una ruta y asignar *slots* a cada demanda. Para cumplir con la recomendación de la UIT, se deben respetar las siguientes restricciones:

- **continuidad**: los *slots* asignados a una determinada demanda deben ser iguales en todos los enlaces de la ruta correspondiente;
- **contigüidad**: los *slots* asignados a cada demanda deben ser contiguos;
- **no-solapamiento**: no se puede asignar un mismo *slot* a diferentes demandas si sus rutas comparten algún enlace.

La restricción de continuidad del espectro garantiza que, cuando no hay convertidores de espectro en la red, se asigne el mismo conjunto de subportadoras en todos los enlaces de un *lightpath*. La restricción de contigüidad del espectro asegura que las subportadoras asignadas a una conexión sean contiguas. Al mismo tiempo, la última restricción garantiza que cada par canal-enlace sea asignado a una sola demanda.

Formalmente, esta versión simplificada del problema puede expresarse de la siguiente manera. Recibimos un grafo dirigido $G = (V, E)$ representando la red de fibra óptica, un número fijo $\bar{s} \in \mathbb{Z}_+$ de *slots* disponibles, y un conjunto de *demandas* $D = \{d_i = (s_i, t_i, v_i)\}_{i=1}^{k}$, donde cada demanda d_i, $i = 1, \ldots, k$, está compuesta por un origen $s_i \in V$,

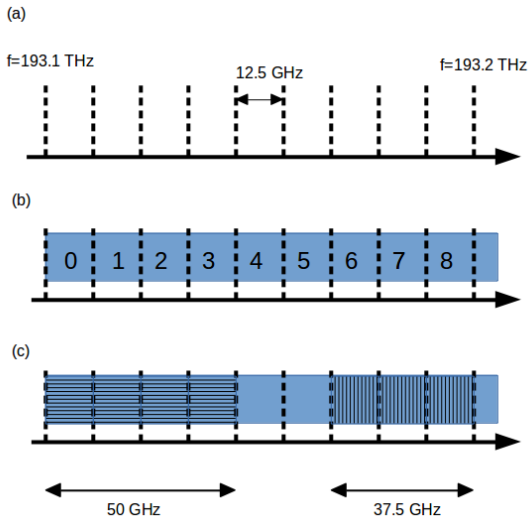


Fig. 1.2: (a) y (b) Espectro óptico dividido en *slots* de 12.5Ghz. (c) Canales formados por secuencias de *slots* consecutivos.

un destino $t_i \in V$, y un volumen $v_i \in \mathbb{Z}_+$. Si $d = d_i = (s_i, t_i, v_i) \in D$ es una demanda, para algún $i \in \{1, \ldots, k\}$, definimos $s(d) = s_i$, $t(d) = t_i$, y $v(d) = v_i$. Definimos como *lightpath* de una demanda $d_i = (s_i, t_i, v_i)$ a una tupla (l, r, p), donde $1 \leq l \leq l + v_i - 1 \leq r \leq \bar{s}$ y p es un camino simple (dirigido) en G de s_i a t_i.

En este escenario, el RSA consiste en establecer un *lightpath* asociado a cada demanda, de forma tal que estos *lightpaths* no se solapen. En otras palabras, cada demanda d_i, con $i = 1, \ldots, k$, debe tener asignado un camino p_i (visto como una secuencia de arcos) en G entre s_i y t_i y un intervalo $[l_i, r_i]$ de al menos v_i *slots* consecutivos en $[1, \bar{s}]$ de modo tal que si $p_i \cap p_j \neq \emptyset$ entonces $[l_i, r_i] \cap [l_j, r_j] = \emptyset$, para cualquier par de índices de demandas $i \neq j$.

La Figura 1.3 presenta un ejemplo de la operación de ruteo y asignación de espectro en una red de cuatro nodos. La instancia pide una asignación de una ruta y una secuencia de *slots* contiguos para tres demandas dadas de la siguiente manera: $d_1 = (v_1, v_3, 2)$, $d_2 = (v_2, v_4, 1)$ y $d_3 = (v_1, v_2, 3)$. En la solución propuesta, el camino más corto entre v_1 y v_3 está reservado para la demanda d_1, a la cual se le asignan los dos primeros *slots*. La demanda d_2 también necesita usar el arco v_2v_3 en su única ruta posible. Por lo tanto, a esta demanda se le asigna el *slot* 3 para evitar el solapamiento con d_1. Finalmente, el camino más corto para satisfacer la demanda d_3 es el arco v_1v_2, pero como este arco tiene sólo 2 *slots* libres, en la solución propuesta la demanda d_3 es satisfecha utilizando el rango de *slots* $[1, 3]$ en una ruta alternativa.

1.1.3. Algunas variaciones del problema

En esta subsección se enumeran algunas de las variaciones más estudiadas del RSA. En primer lugar, las interferencias y deterioros de la capa física, como el ruido y la diafonía, hacen que la calidad de las señales ópticas se degrade a medida que éstas atraviesan la red. Las variantes más cercanas al problema tienen en cuenta este hecho y limitan el alcance de cada conexión. Esta versión se conoce como el problema de *asignación de espectro y*

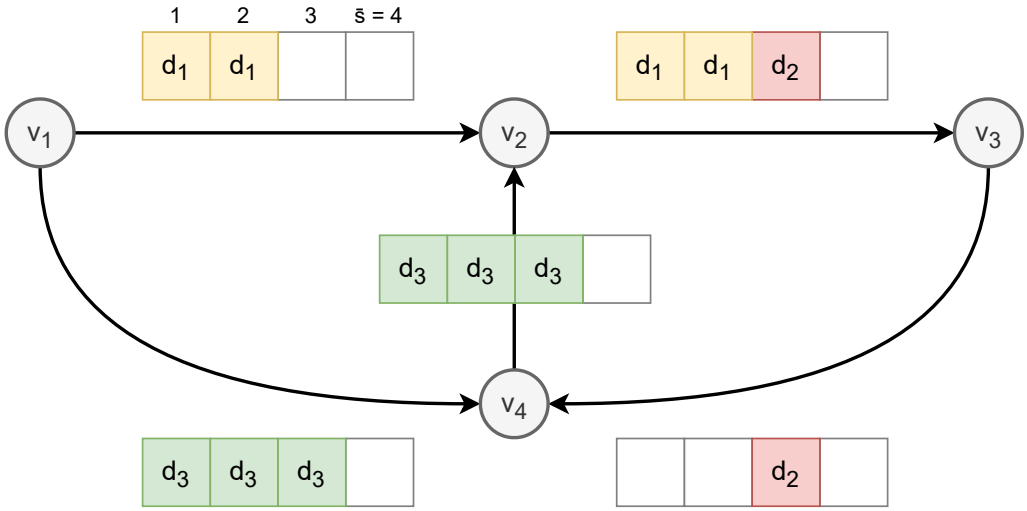


Fig. 1.3: Ejemplo del ruteo y asignación de espectro para las demandas $d_1 = (v_1, v_3, 2)$, $d_2 = (v_2, v_4, 1)$ y $d_3 = (v_1, v_2, 3)$.

ruteo restringido (C-RSA, del inglés *Constrained-RSA*) [7, 53, 137]. La versión del RSA presentada en la sección anterior también asume que cada demanda usa el mismo formato de modulación. Una variación más compleja, presentada por Christodoulopoulos et al. [31, 32], conocida como el problema de *ruteo, nivel de modulación y asignación de espectro* (RMSA, del inglés *routing, modulation level and spectrum allocation*), permite que cada demanda utilice diferentes tipos de modulación. La mayor parte del trabajo existente sobre el RMSA con *impairment-awareness* (conciencia de deterioro) en redes ópticas elásticas también se basa en limitar el alcance de transmisión [32, 62, 144, 155]. En estos trabajos, cada formato de modulación tiene un límite de alcance de transmisión asociado, el cual depende sólo de las degradaciones lineales, como se muestra en la Tabla 1.1. Se supone que la calidad de una ruta asignada a un formato de modulación particular es buena siempre que la longitud de la ruta no sea mayor que el alcance de transmisión correspondiente.

	BPSK	QPSK	8-QAM	16-QAM
Alcance [Km]	6300	3500	1200	600
Tasa de bits [Gbps]	50	100	150	200

Tab. 1.1: Alcance de transmisión y tasa de bits admitida por un transceptor para diferentes formatos de modulación.

Además, para evitar cualquier interferencia no lineal entre dos conexiones a las que se les asigna bandas de subportadoras adyacentes, se inserta una *banda de protección* (también conocida como *banda de guarda* o *espaciado de canal*) la cual está compuesta por varias subportadoras. Diversos trabajos estudian este enfoque. Sin embargo, el modelo de alcance de transmisión no tiene en cuenta las degradaciones no lineales (NLI, del inglés *nonlinear impairments*) que existen en la realidad, como la interferencia entre conexiones, que dependen del ruteo y la asignación de recursos de las conexiones en la misma ruta. Por tanto, el alcance y la banda de guarda pueden sobrestimar o subestimar las deficiencias de conexión. Algunos autores [97] consideran una relación señal/ruido (SNR, del inglés

signal-to-noise ratio) para cada conexión, la cual condensa las densidades espectrales de potencia (PSD, del inglés *power spectral densities*) de la señal, el ruido de emisión espontánea amplificada (ASE, del inglés *amplified spontaneous emission*) y el ruido de las degradaciones no lineales (NLI). Estos últimos ruidos son introducidos por cada conexión en todas las demás conexiones dependiendo de sus formatos de modulación y el espaciado entre sus frecuencias centrales.

La introducción de la dimensión espacial –a través de la disponibilidad de modos espaciales paralelos en la *multiplexación por división de espacio* (SDM, del inglés *space-division multiplexing*)– permite transportar *supercanales* (SChs, del inglés *super-channels*) utilizando *slots* no sólo en el dominio de la frecuencia (como en las EON), sino también distribuyéndolos en diferentes modos espaciales. En las redes ópticas, espectral y espacialmente flexibles, el RSA se convierte en el problema de *ruteo, modo espacial y asignación de espectro* (abreviado como RSSA, del inglés *ruteo, spatial mode, and spectrum allocation*), que agrega nuevas variables de decisión relacionadas con la selección de modos espaciales en los enlaces de la red por medio de los que se transmitirán las señales ópticas. La versión más simple de SDM asume el uso de manojos de fibras compuestos de fibras monomodo físicamente independientes. Las implementaciones de SDM más avanzadas, en cambio, se basan en *fibras multinúcleo* (MCF) o *fibras multimodo* (MMF).

Dependiendo del tipo de fibra SDM que se utilice, el RSSA puede tener un nombre y un significado ligeramente diferentes. Por ejemplo, si se utilizan MCF, el RSSA generalmente se conoce como el problema de *ruteo y asignación de núcleo y espectro* (RCSA, del inglés *routing, core, and spectrum allocation*) [47, 96]. Por cada variable adicional que es tenida en cuenta, como ser las relacionadas con el formato de modulación o la selección de la velocidad en baudios, es posible generar muchas otras variantes del RSSA. Ejemplos de estos problemas más específicos son el de *ruteo, selección de formato de modulación, núcleo y asignación de espectro* (RMCSA, del inglés *routing, modulation format, core, and spectrum allocation*) [95, 112] o el problema de *asignación de rutas, espectro, núcleo y/o modo* (RSCMA, del inglés *routing, spectrum, and core and/or mode assignment*). En las redes SDM con *pocas fibras de modo* (FMF) se estudia el problema de *ruteo y selección de formato de modulación, de velocidad en baudios y asignación de espectro* (RMBSA, del inglés *routing, modulation format, baud rate, and spectrum allocation*) [118, 142]. Finalmente, en las redes WDM de grilla fija con MCF [82], un problema común es el de *ruteo, y asignación de wavelength y núcleo* (RWCA, del inglés *routing, wavelength and spectrum assignment*). En las redes *multiplexor de división de longitud de onda de filtro* (FWDM, del inglés *filter wavelength division multiplexer*), debido a la asignación flexible de espectro, el problema de *asignación de rutas, wavelength y espectro* (RWSA) [105] además de la restricción de continuidad de los *wavelengths*, debe cumplir con las restricciones de continuidad y conflicto espectral.

También es común, al resolver el RSCA [96], tener en cuenta la diafonía entre núcleos, la cual se produce cuando las señales ópticas que utilizan el mismo espectro se propagan a través de núcleos adyacentes en MCF.

Se pueden agregar y conmutar múltiples conexiones de baja velocidad en un solo canal de alta capacidad para reducir el número de canales flexibles multiplexados mientras se transporta la misma cantidad de tráfico. Este procedimiento, análogo al que se aplica en los sistemas WDM, es también conocido como *preparación del tráfico* (*traffic grooming* en inglés) [59, 142].

Estos problemas generalmente se estudian para escenarios *unicast*, es decir, cuando

cada demanda tiene un solo origen y un destino. Sin embargo, hay algunos trabajos, por ejemplo [163], que asumen múltiples destinos por demanda, es decir, el escenario conocido como *manycast*, y hay también trabajos donde se considera que los mensajes se entregan a cualquiera de un grupo de nodos, generalmente al más cercano al origen, utilizando una asociación de uno-a-uno-de-muchos donde los datagramas se enrutan a cualquier miembro individual de un grupo de receptores potenciales que están todos identificados por la misma dirección de destino. Este último caso se conoce como *anycast* [3, 148, 149].

Estos problemas a su vez admiten variaciones según la forma en que llegan las solicitudes, el modo en el que se procesan, la vida útil de una demanda y su capacidad para aumentar o disminuir el ancho de banda solicitado. En este contexto, hay dos tipos principales de tráfico. Para cada uno de ellos, junto con los nodos de origen y destino de cada demanda, también se recibe el ancho de banda requerido, y se asume que la demanda tiene una duración indefinida dentro de la red. En el tráfico *estático* [14, 17, 18, 19, 22], también conocido como fase de *planificación* o fase *offline* del problema y generalmente utilizada para el diseño de redes, todas las solicitudes se conocen de antemano; mientras que en el tráfico *dinámico* [9, 12, 23], que también se conoce como fase *operativa* o fase *online*, las demandas llegan y se procesan inmediatamente. Asimismo, aunque es menos común, existe una tercera clase entre estas dos, denominada tráfico *semi-dinámico* [1, 10], en la cual la llegada de solicitudes sucede de a ráfagas. Tanto la clase dinámica como esta última cuentan con versiones en las que las demandas ya asignadas pueden solicitar la modificación de su ancho de banda durante el período de permanencia en la red.

El ruteo de camino único es el enfoque más común; sin embargo algunos trabajos utilizan el ruteo *multi-camino* [27, 119, 120] para mitigar la fragmentación del espectro, la cual es muy común en escenarios de tráfico dinámico. El primer ruteo trata de encontrar un único camino para satisfacer cada demanda mientras que el segundo permite dividir el pedido en varios caminos que, al sumarlos, satisfacen el volumen de la demanda.

Poder seguir operando ante una falla de un componente de la red también es un problema de gran importancia a causa de la pérdida de datos que provocaría [130, 140]. La incorporación de la *capacidad de supervivencia* en el RSA se encuentra actualmente en su etapa inicial y sólo se han dedicado unos pocos trabajos a este fin, principalmente por fallas de enlace, haciendo uso de metaheurísticas [25, 66, 152]. Hay dos formas principales de compartir recursos de respaldo tanto en WDM como en redes ópticas basadas en OFDM con asignación de ancho de banda elástica o fija: la protección mediante el uso de *ruta dedicada* [36, 69, 152] y la protección mediante el uso de *ruta compartida* [36, 83, 127].

1.2. Optimización combinatoria

La optimización combinatoria es un campo en la intersección de las matemáticas y las ciencias de la computación con un desarrollo impresionante en los últimos años, impulsado por la demanda de aplicaciones basadas en modelos discretos. Los problemas de optimización combinatoria surgen en una gran variedad de contextos en ciencia, ingeniería y administración. En tales problemas, el objetivo es encontrar una solución x^* dentro de un conjunto discreto $\mathcal{F}$ que optimice una función objetivo $c : \mathcal{F} \to \mathbb{R}$. En general, esto conduce a problemas $\mathcal{NP}$-difíciles que, sin embargo, es menester resolver en la práctica debido a su importancia para diversas aplicaciones. Una forma natural de estudiar problemas de optimización combinatoria es expresando todos los puntos $x \in \mathcal{F}$ como un conjunto de

soluciones de un sistema de desigualdades

$$\begin{aligned} Ax &\leq b \\ x &\in \mathbb{Z}^n \end{aligned} \tag{1.1}$$

que, combinado con una función objetivo lineal, resulta en un *programa lineal entero* (ILP, del inglés *integer linear program*). Esto constituye un marco de modelado bastante poderoso que proporciona una gran flexibilidad para expresar tanto los problemas clásicos de optimización combinatoria como, mediante ligeras modificaciones del sistema de desigualdades $Ax \leq b$, muchas de sus variantes que requieren incorporar restricciones laterales impuestas por sus aplicaciones prácticas modernas [15, 52, 99, 125, 126].

Aunque la programación lineal entera es $\mathcal{NP}$-difícil en general [48], la resolución computacional de programas lineales enteros ha avanzado considerablemente durante las últimas décadas. Existen muchos programas, denominados *solvers*, desarrollados exclusivamente para ILP, y los algoritmos generales implementados por estos *solvers* se pueden mejorar con la adición de conocimiento específico sobre el modelo que se está resolviendo. En particular, la identificación y estudio de desigualdades válidas, junto con los procedimientos de separación de estas desigualdades, ha permitido el diseño de programas eficientes para varios problemas de optimización combinatoria. Esto hace posible resolver instancias de tamaño real de estos problemas de forma óptima en tiempos aceptables. La búsqueda de desigualdades válidas asociadas a una formulación de ILP conduce naturalmente al estudio de los poliedros asociados

$$P(A, b) = \text{conv}\{x \in \mathbb{Z}^n : Ax \leq b\},$$

un campo conocido como *combinatoria poliedral*. Para muchos casos especiales se conoce una descripción completa de la cápsula convexa del conjunto solución $\mathcal{F}$ y esta descripción puede usarse para resolver en tiempo polinomial el problema de separación para el poliedro asociado a la formulación. Basado en el método del elipsoide, Grötschel, Lovász y Schrijver [52] demostraron que el problema de separación y el problema de optimización sobre un poliedro son polinomialmente equivalentes, es decir, si un problema es polinomialmente resoluble, también lo es el otro. A partir de esta equivalencia, existe una conjetura generalizada que sugiere que si un problema de optimización combinatoria puede resolverse en tiempo polinomial, entonces debería existir alguna formulación ILP del problema para la cual la cápsula convexa de sus soluciones admita una caracterización "elegante". Como se sabe que el RSA es $\mathcal{NP}$-difícil incluso en casos muy especiales, no podemos esperar que un sistema de restricciones "sencillo" sea suficiente para describir su conjunto de soluciones. El hecho de que el mismo problema de optimización combinatoria pueda ser codificado en general por diferentes ILP da lugar a la pregunta adicional de qué constituye una buena formulación y por lo tanto motiva a comparar formulaciones alternativas para el mismo problema.

Además, la *simetría* está presente en muchos problemas de optimización combinatoria y disminuye la eficiencia computacional de varios modelos si su conjunto de soluciones contiene soluciones factibles que parecen diferentes debido al modelo, pero de hecho son equivalentes [90]. Se sabe que el problema de la asignación del espectro tiene diversos niveles de simetría [89], y también el problema del ruteo está sujeto a simetría dado que las demandas pueden compartir origen y destino. En este contexto, es importante cuidar de que los modelos no generen un espacio de soluciones innecesariamente grande debido a soluciones simétricas.

1.3. Estado del arte

Dado que la EON ha sido ampliamente aceptada como la red de alta velocidad de la próxima generación, desde sus comienzos en 2008 los investigadores se han centrado en su arquitectura, el mecanismo de asignación de espectro y su alcance futuro.

Se han explorado diversos enfoques de soluciones en los últimos años, incluidos los abordajes directos que utilizan *solvers* comerciales de ILP. Éstos se han utilizado para resolver formulaciones de manera óptima, como por ejemplo en [1, 7, 8, 12, 22, 31, 32, 35, 163, 164, 165], o para resolver heurísticas basadas en ILP, generalmente utilizando conjuntos precomputados de caminos, como en [70, 72, 79, 108, 146]. También se ha propuesto una gran cantidad de heurísticas [132, 136, 147, 148, 155, 157, 158, 159, 163, 164, 165, 167], así como metaheurísticas [25, 66, 152], y en particular algoritmos genéticos [146, 176]. Es posible encontrar algunos pocos trabajos cuyos autores aplican descomposiciones Lagrangianas y técnicas de generación de columnas [21, 42, 43, 51, 58, 75, 121, 122, 147]. El enfoque más común implica descomponer el RSA en dos sub-problemas a resolver por separado; por un lado el ruteo y por otro la asignación de espectro (el SA) [1, 3, 7, 9, 10, 12, 33, 34]. Este último enfoque corresponde a la interpretación del RSA como un problema de coloreo por intervalos en un grafo de intersección de caminos de las distintas demandas, que se asemeja a un problema de asignación de ancho de banda ya estudiado en [86, 87, 88]. La aplicación de métodos similares permitiría resolver el problema de RSA mediante técnicas de *branch-and-cut-and-price*.

Una lista de las principales publicaciones ordenadas cronológicamente con un breve resumen de cada una –resaltando los datos más relevantes para nuestro trabajo– se puede consultar en el Anexo 8. Estas mismas publicaciones, clasificadas según diferentes criterios, se pueden ver en el Anexo 7.

Hasta donde sabemos, de acuerdo con la bibliografía existente, la mayoría de los trabajos realizados son heurísticos y, aunque varios autores presentaron algunos modelos matemáticos para resolver el RSA –o sus variantes– por optimalidad [142, 144, 146, 147, 150, 157], éstos han hecho muy poco uso del potencial que tienen las técnicas de programación lineal entera. De hecho, aparte de algunas propuestas de algoritmos de generación de columnas, y una sola publicación que propone técnicas de *pre-solve* [35], hemos encontrado muy pocos trabajos que presenten desigualdades válidas para mejorar el algoritmo *branch-and-cut* genérico de los *solvers*.

En julio de 2014 [75] Klinkowski et al. mejoran la formulación presentada por Ruiz et al. en 2013 [121] mediante el uso de una familia de desigualdades válidas, las *desigualdades clique*, que aseguran que a lo sumo uno solo de un conjunto de k lightpaths puede usarse cuando éstos se superponen. Sin embargo, los resultados fueron levemente mejores que los presentados en el trabajo mencionado anteriormente.

En junio [74] y en octubre [71] de 2015, Klinkowski et al. presentan una formulación de ILP basada en un modelo anterior [146] para la versión estática del RSA. Dado que este modelo utiliza una familia exponencial de variables que relaciona *lightpaths* precalculados con demandas, los autores proponen un enfoque de *branch-and-cut-and-price* para resolverlo y agregan una familia de planos de corte para mejorar la cota inferior. En junio de 2016 [73], agregan algunas desigualdades válidas más con el fin de mejorar dicho algoritmo y logran resolver instancias un poco más grandes. Sin embargo, dado que para los experimentos calculan previamente un subconjunto de rutas para cada demanda, las soluciones obtenidas no tienen garantía de optimalidad.

En septiembre de 2019 [53] Hadhbi, Y. et al. presentan un modelo de ILP basado en enlace-nodo y proponen un algoritmo *branch-and-cut* para resolver una variación del RSA considerando un límite para la longitud de las rutas e interpretando la topología como un grafo no dirigido. Las desigualdades que los autores proponen como planos de corte en este trabajo pertenecen a dos familias exponenciales y están basadas sólo en consideraciones de flujo, es decir, considerando únicamente a los nodos y aristas del grafo; éstas son denominadas *continuidad del camino* y *eliminación de ciclos*.

No encontramos ningún trabajo que utilice heurísticas como *warm-start* de un algoritmo exacto, es decir, que el segundo parta de una solución factible proporcionada por el primero. En [36] el autor hace algunos experimentos de agregar el resultado de un algoritmo genético como un *warm-start* de un modelo ILP, pero utilizando solamente un subconjunto de las rutas y canales posibles, con lo cual termina resolviendo heurísticamente el problema.

También notamos que, más allá de los datos experimentales –a menudo escasos– ofrecidos por cada publicación, ningún trabajo en la literatura estudia el conjunto de instancias utilizadas en la bibliografía. Los dos últimos artículos que describen un análisis sobre algunas de las topologías existentes datan de 2004 [56] y de 2011 [76].

1.4. Motivación de la tesis

Dado que en la literatura no hemos encontrado trabajos que logren resolver instancias reales del RSA en forma óptima, optamos por enfocarnos en una de las versiones más simples del problema. Con base en lo discutido en [62] y [70], donde se asegura que la dependencia de la longitud de la ruta comienza a jugar un papel cuando la misma está compuesta por más de diez nodos, en aras de la simplicidad, asumimos que el volumen de la demanda se puede mapear directamente a *slots* y el mismo puede ser constante a lo largo de todo el camino, considerándolo independiente de la longitud de éste; con lo cual descartamos el formato o nivel de modulación.

Dado que el objetivo del presente trabajo es adquirir conocimiento y dar los primeros pasos en la aplicación de técnicas de optimización combinatoria sobre el RSA, decidimos centrarnos en su versión estática. Ésta es más difícil que la dinámica, ya que tiene como objetivo optimizar conjuntamente el establecimiento de todas las conexiones y minimizar los recursos utilizados, pero esto mismo le da un carácter más combinatorio al problema, de igual forma que el problema de flujo con multiples productos es más difícil que el problema del camino mínimo en redes generales [144]. También asumimos que cada enlace es bidireccional, por lo que interpretamos la topología como un grafo dirigido, y utilizamos una única ruta para satisfacer cada demanda.

También optamos por la eliminación de las bandas de guarda, como suele hacerse en varios trabajos de la literatura, por ejemplo [131]. Este enfoque simplifica el problema y se puede adaptar fácilmente extendiendo la capacidad de la fibra y los requerimientos de cada demanda en ambas direcciones.

Como mostramos en la siguiente sección, esta simplificación del problema, sin embargo, pertenece a la clase $\mathcal{NP}$-difícil y sigue siendo muy complicado de resolver en la práctica.

1.4.1. Complejidad computacional del RSA

En esta sección, luego de recorrer la bibliografía mencionando las principales publicaciones que presentan algún tipo de resultado teórico sobre la complejidad del RSA o de

alguna de sus variaciones, demostramos formalmente que la versión de decisión del RSA pertenece a la clase $\mathcal{NP}$-completo y algunas consecuencias derivadas de este hecho.

En 1992 [28], Chlamtac y Ganz demostraron que el problema de decisión de la versión estática del problema de ruteo y asignación de *wavelengths*, llamado problema de *establecimiento estático de lightpaths* (SLE, del inglés *static lightpath establishment*), es $\mathcal{NP}$-completo, reduciendo el problema de coloreo al SLE. Unas décadas más tarde, en 2011 [158], Wang et al. esbozaron una breve demostración reduciendo una simplificación del RSA con rutas fijas –conocida como SRA– al RSA. En febrero de ese mismo año, Klinkowski y Careglio [70] y luego en agosto Klinkowski y Walkowiak [72], mostraron que el RSA es $\mathcal{NP}$-difícil en general al reducir el RWA al RSA. En su demostración señalan que el caso especial del RSA en el que el volumen de cada demanda es 1, es equivalente al RWA. Dos años después, en junio de 2013 [131] Shirazipourazad et al. mostraron que la versión de decisión del RSA estático es $\mathcal{NP}$-completo incluso cuando la topología es un anillo.

La demostración que presentamos a continuación es una adaptación de la presentada en 2011 [32], por Christodoulopoulos et al. cuando probaron que la versión particular del RSA que tiene en cuenta el nivel de modulación, *i.e.*, el RMLSA, es $\mathcal{NP}$-difícil.

Definimos una instancia del RSA como una terna $\mathcal{I} = (G, D, \bar{s})$ donde $G = (V, E)$ es el grafo dirigido con conjuntos de vértices y arcos V y E, respectivamente, D es el conjunto de demandas y $\bar{s} \in \mathbb{N}$ la capacidad máxima de slots por arco.

Proposición 1.4.1. *El problema de decidir si una instancia $\mathcal{I}$ es factible es $\mathcal{NP}$-completo.*

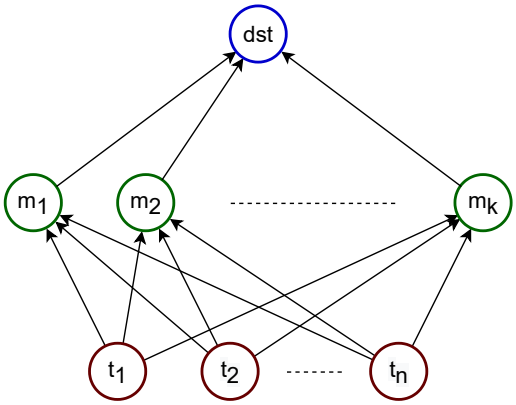


Fig. 1.4: Instancia del RSA que permite resolver el MSP.

Para probar la proposición, vamos a reducir una versión del problema de planificación con múltiples procesadores (MSP, del inglés *multiprocessor scheduling problem*) al RSA. En el MSP recibimos un conjunto de n tareas y un conjunto de k procesadores, llamémoslos $T = \{t_1, \ldots, t_n\}$ y $M = \{m_1, \ldots, m_k\}$ respectivamente. Para cada par de ellos también tenemos un tiempo $w(t_i, m_j) \in \mathbb{Z}^+$ con $i = 1, \ldots, n$ y $j = 1, \ldots, k$. El problema consiste en encontrar un *schedule* para T que asigne cada tarea a un procesador. La función objetivo busca minimizar el tiempo total de ejecución, conocido como *makespan*, es decir, el tiempo entre el inicio y el final de la ejecución de todas las tareas.

El problema de decisión asociado, que pertenece a la clase $\mathcal{NP}$-completo, consiste en determinar si dado un límite b existe una solución factible con un *makespan* inferior o igual a b.

Conocer el *makespan* mínimo sigue siendo $\mathcal{NP}$-difícil incluso dado un orden arbitrario entre tareas y teniendo la misma duración de cada una en cada procesador [65], es decir, $w(t, m_i) = w(t, m_j)$ por cada $t \in T$ y cada $i, j = 1, \ldots, k, i \neq j$. Entonces, podemos omitir m en la notación, es decir, $w : T \to \mathbb{Z}^+$.

Demostración. Dada una instancia cualquiera del MSP con T, M, w y un límite b, comenzamos nuestra reducción generando un grafo dirigido $G = (V, E)$ (ver, e.g., la Figura 1.4) donde el conjunto V está integrado por un nodo por cada procesador m_j, un nodo para cada tarea t_i, y un nodo dst, y donde E está conformado por un arco (t_i, m_j) por cada $i = 1, \ldots, n$ y $j = 1, \ldots, k$, y un arco (m_j, dst) por cada $j = 1, \ldots, k$. Entonces podemos definir el conjunto de demandas como $D = \{(t_i, dst, w(t_i)) : i = 1, \ldots, n\}$ y tomar $\bar{s} = b$.

En este contexto, definimos un *lightpath* para una demanda d como el par $l(d) = (p, r)$ con $p \subset V$ un camino que va desde el origen $s(d)$ al destino $t(d)$, y r el mayor *slot* utilizado por la demanda d a lo largo del camino p para satisfacer su volumen.

Sea $L = \{l(d_1), \ldots, l(d_n)\}$ el conjunto de *lightpaths* de una solución entera para una instancia $\mathcal{I} = (G, D, \bar{s})$. El único modo de satisfacer una demanda es utilizando un camino que atraviese el nodo m_h para algún $h \in \{1, \ldots, k\}$. De este modo, el *lightpath* que satisface la demanda d_i tiene que ser $l(d_i) = (p_i, r_i) \in L$, con $p_i = \{t_i, m_h, dst\}$. Lo cual implica que por cada tarea t_i tenemos un par formado por un procesador y un intervalo de tiempo: $(m_h, [r - w(t_i), r])$ dándonos un *schedule* que resuelve esta versión del MSP. Por lo tanto si esta instancia del RSA es factible con $\bar{s} = b$ *slots*, el MSP es factible con un *makespan* menor o igual a b.

Por otro lado, si el MSP es factible con un *makespan* menor o igual a b, deberíamos tener una asignación $(m_h, [r_i - w(t_i), r_i])$, con $r_i \leq b$ para cada t_i, usando el procesador m_h, por lo que podemos generar un conjunto de *lightpaths* $l(d_i) = (p_i, r_i) \in L$, con $p_i = (t_i, m_h, dst)$, tal que resuelva nuestra instancia del RSA usando a lo sumo b *slots*. Esto significa que nuestra instancia es factible con b *slots* si y sólo si la instancia original del MSP es factible.

La reducción entonces es correcta y podemos concluir que decidir si el RSA es factible usando b *slots* es un problema tan difícil como decidir si el MSP es factible con un *makespan* menor o igual a b. Por lo tanto, ambos problemas son $\mathcal{NP}$-completos. □

Corolario 1.4.1. *Calcular el mínimo $\bar{s}$ tal que una instancia $\mathcal{I} = (G, D, \bar{s})$ es factible es $\mathcal{NP}$-difícil.*

Sea $E^{fs}(G, \bar{s}, D, d) \subseteq E$ el conjunto de arcos usados en al menos una solución factible de la instancia $\mathcal{I} = (G, D, \bar{s})$ por la demanda $d \in D$. Es decir, $E^{fs}(G, \bar{s}, D, d)$ está compuesto por todo arco e tal que existe al menos una solución factible en la que d usa e. Con esta definición podemos enunciar el siguiente resultado.

Teorema 1.4.1. *Dado un digrafo G, la cantidad total de* slots *$\bar{s}$, el conjunto de demandas D y una demanda particular d, el problema de decidir si un arco e pertenece al conjunto $E^{fs}(G, \bar{s}, D, d)$ es Cook-$\mathcal{NP}$-completo.*

Demostración. Asumamos que es polinomial determinar si un arco e pertenece al conjunto $E^{fs}(G, \bar{s}, D, d)$ para toda demanda $d \in D$ en una instancia dada por la terna $(G, D, \bar{s})$. Entonces, podemos seleccionar una demanda particular d y, para cada $\bar{s} = 1, \ldots, \sum_{d' \in D} v(d')$, iterar sobre todos los arcos $e \in E$ buscando el mínimo $\bar{s}$ para el cual existe un arco $e \in E$ tal que e pertenece a $E^{fs}(G, \bar{s}, D, d)$. Con una cantidad de *slots* $\bar{s}$ igual a la suma de los volúmenes de las demandas, resolviendo el problema de camino mínimo para cada una de

ellas, tenemos una solución a la nueva instancia del problema, si para cada demanda existe un camino que une su par origen-destino. Entonces nuestro algoritmo se va a detener en alguna iteración j, devolviendo un valor para $\bar{s} \leq \sum_{d' \in D} v(d')$. Llamemos $\bar{s}_j$ a dicho valor y $\bar{s}^*$ al mínimo número de slots $\bar{s}$ tal que la instancia dada por $(G, D, \bar{s}^*)$ es factible. Dado que e pertenece a $E^{fs}(G, \bar{s}_j, D, d)$, la instancia dada por $(G, D, \bar{s}_j)$ es factible, entonces $\bar{s}^* \leq \bar{s}_j$. Asumamos que $\bar{s}^* < \bar{s}_j$. Como $\bar{s}^*$ alcanza para que la instancia dada por $(G, D, \bar{s}^*)$ sea factible, entonces para cada demanda d tenemos un camino p_d que conecta su origen con su destino y, por lo tanto al menos un arco $e \in p_d$ (dado que una demanda d debe tener $s(d) \neq t(d)$). Por lo tanto, dicho arco e debe pertenecer a $E^{fs}(G, \bar{s}^*, D, d)$ y nuestro algoritmo debería haber terminado en dicha iteración debido a que $\bar{s}^* < \bar{s}_j$. Si en lugar de una búsqueda exhaustiva usáramos búsqueda binaria sobre $\bar{s}$, tendríamos un algoritmo polinomial para hallar tal $\bar{s}^*$ resolviendo por lo tanto un problema $\mathcal{NP}$-completo. □

Teorema 1.4.2. *Dado un digrafo G, la cantidad de* slots *$\bar{s}$, el conjunto de demandas D y una demanda particular d, el problema de decidir si un arco e pertenece al conjunto $E^{fs}(G, \bar{s}, D, d)$ es Karp-$\mathcal{NP}$-completo.*

Demostración. Reduciremos el problema de decidir la factibilidad del RSA dada la cantidad de *slots* –el cual ha sido demostrado que es $\mathcal{NP}$-completo en la Proposición 1.4.1– al problema de decidir si un arco e pertenece al conjunto $E^{fs}(G, \bar{s}, D, d)$. Dada una instancia arbitraria $\mathcal{I}$ podemos generar una nueva instancia $\mathcal{I}'$ agregando un nodo nuevo b conectado a algún otro nodo existente $a \in V$ mediante un nuevo arco $e' = (a, b)$, y agregando además una nueva demanda $d' = (a, b, 1)$ al conjunto de demandas D de modo tal que nuestro nuevo problema sea ahora decidir si el arco e' pertenece a $E^{fs}(G' = (V \cup \{b\}, E \cup \{e'\}), \bar{s}, D \cup d', d')$. El nodo b no puede ser el destino de ninguna demanda excepto de la nueva, por lo que el arco e' no va a ser parte de ningún camino asignado a alguna demanda, salvo del camino trivial utilizado por el *lightpath* que satisface la demanda nueva. Como siempre será posible satisfacer dicha demanda, entonces nuestra nueva instancia tendrá respuesta *verdadero* si y sólo si es posible encontrar un ruteo y asignación de espectro a cada una de las demandas existentes en la instancia original $\mathcal{I}$. Consecuentemente, somos capaces de resolver un problema $\mathcal{NP}$-completo solucionando el problema de decidir si un arco e pertenece al conjunto $E^{fs}(G, \bar{s}, D, d)$. □

Análogamente a E^{fs}, definamos $\bar{S}(G, \bar{s}, D, d)$ como el conjunto de todos los *slots* que d puede utilizar en al menos una solución factible de una instancia $\mathcal{I} = (G, D, \bar{s})$. Con esta nueva definición podemos enunciar este otro teorema.

Teorema 1.4.3. *Dado un digrafo G, el número de* slots *$\bar{s}$, el conjunto de demandas D y una demanda particular d, el problema de decidir si un* slot *s pertenece a $\bar{S}(G, \bar{s}, D, d)$ es Karp-$\mathcal{NP}$-completo.*

Demostración. Dada una instancia $\mathcal{I}$ podemos generar una nueva instancia igual a $\mathcal{I}$ pero agregando un nuevo arco $e' = (a, b)$ que conecte un nodo existente $a \in V$ con un nuevo nodo b, y agregando una nueva demanda $d' = (a, b, 1)$ al conjunto D. Podemos entonces preguntar si un *slot* s pertenece al conjunto $\bar{S}(G' = (V \cup \{b\}, E \cup \{e'\}), \bar{s}, D \cup d', d')$. Como en una solución factible la demanda d' puede utilizar cualquier *slot* del arco (a, b), dado que ninguna otra demanda puede hacer uso de ellos (el arco no pertenece a ningún camino utilizado para satisfacer ninguna demanda existente), esta instancia reducida tendrá respuesta *verdadero* si y sólo si es posible encontrar una asignación de camino y espectro para

cada una de las demandas existentes en la instancia original $\mathcal{I}$. Por lo tanto, resolviendo dicha reducción seríamos capaces de dar solución a un problema $\mathcal{NP}$-completo. □

1.5. Contenido de la tesis

En la Sección 1.1 hemos revisado el camino seguido desde la aparición de la fibra óptica hasta el surgimiento del RSA. Luego, en la Sección 1.1.2 hemos presentado la versión más simple, que es la que estudiamos en esta tesis, por lo que la hemos definido formalmente. Asimismo, en la Sección 1.1.3 hemos enumerado las variantes más cercanas del problema que son tratadas a en la literatura, mientras que en la Sección 1.2 hemos hecho una presentación general de las técnicas utilizadas en el presente trabajo, seguido de una pequeña revisión del estado del arte en la Sección 1.3. Finalmente, en la Sección 1.4.1, hemos realizado un análisis de la complejidad del RSA y algunos problemas relacionados, probando formalmente que la versión de decisión del RSA pertenece a la clase $\mathcal{NP}$-completo junto con algunas consecuencias derivadas de este hecho.

El resto de este documento está organizado de la siguiente manera. En el Capítulo 2 estudiamos diversas formulaciones de ILP para el RSA. En la Sección 2.1 revisamos los modelos presentes en la literatura y clasificamos algunos de ellos de acuerdo con los criterios más comunes. En particular, analizamos los diferentes enfoques con los que intentan resolver las tres grandes restricciones del RSA, es decir, contigüidad, continuidad y no solapamiento, así como la función objetivo utilizada. En la Sección 2.2 describimos dos de estas formulaciones existentes que pertenecen a la categoría que nos interesa, es decir, las formulaciones compactas. En la Sección 2.3.1 enunciamos un teorema que caracteriza todas las soluciones para una versión muy reducida del RSA, el cual es utilizado en dos de las doce formulaciones que presentamos en la Sección 2.3. Para estas formulaciones, consideramos dos formas naturales de modelar soluciones factibles dentro de un enfoque de programación entera: o representamos el ruteo con un conjunto de variables y la asignación de *slots* con un segundo conjunto de variables, o bien representamos ambas decisiones con un único conjunto de variables. Para cada modelo presentamos algunas posibles variaciones. Al final de dicha sección se presenta una breve discusión donde se comparan los tamaños de los distintos modelos. En la Sección 2.4 estos doce modelos y variaciones propuestos se comparan con los extraídos de la literatura, usando un *solver* MILP genérico como una caja negra, primero en instancias con parámetros generados aleatoriamente en algunas topologías reales, y luego con instancias basadas en en datos reales. El capítulo se cierra con una serie de conclusiones presentadas en la Sección 2.5.

En el Capítulo 3 comenzamos el estudio de uno de los modelos con mejor desempeño presentado en el Capítulo 2. En particular, tratamos de mejorar los resultados obtenidos con una de las formulaciones compactas que utiliza un solo conjunto de variables, a saber, DSL-BF, aplicando técnicas habituales en programación lineal entera. En primer lugar, pretendemos estudiar la formulación y, explotando algunas de sus propiedades, presentar una serie de desigualdades válidas que pueden utilizarse como planos de corte en un algoritmo de tipo *branch-and-cut*. En la Sección 3.1 proponemos tres teoremas fundamentales provenientes de consideraciones de simetría que nos permiten generar varias familias de desigualdades válidas partiendo de otras. En la Sección 3.2 enumeramos diferentes características no deseadas que puede tener una solución factible, relacionadas con la restricción de ruteo, y presentamos varias familias de desigualdades válidas para mitigarlas. En la Sección 3.3 presentamos conjuntos de desigualdades válidas relacionadas con las restricciones

de contigüidad, la mayoría de ellas usando el teorema presentado en la Sección 2.3.1. La Sección 3.4 presenta varias familias de desigualdades válidas, ecuaciones y cortes de optimalidad relacionadas con la restricción de no solapamiento. Cerramos el capítulo en la Sección 3.5 con algunas conclusiones sobre estas más de sesenta familias propuestas.

Continuamos en el Capítulo 4 con la implementación de un algoritmo *branch-and-cut* usando las desigualdades de las familias presentadas en el Capítulo 3 como planos de corte. En la Sección 4.1 mostramos los procedimientos implementados para separar estas familias y estudiar sus complejidades temporales, mientras que en la Sección 4.2 presentamos diversas estrategias orientadas a seleccionar algunas de las desigualdades de estas familias. En la Sección 4.3 presentamos la experiencia computacional, en particular la calibración de los parámetros para cada procedimiento y una comparación entre las diferentes estrategias y un algoritmo genérico de tipo *branch-and-cut* y otro *branch-and-bound*. El set de instancias utilizado en estos experimentos y los siguientes se presenta en la Sección 4.3.1.

Los resultados obtenidos con el algoritmo *branch-and-cut* sugieren que partiendo de una solución factible de buena calidad, el algoritmo podría obtener mejores resultados más rápidamente. Por lo tanto, en el Capítulo 5 se propone una heurística primal. En la Sección 5.1 revisamos brevemente el estado del arte relacionado con heurísticas tanto para el RSA como para el SA. Luego, en la Sección 5.2, presentamos formalmente la heurística propuesta, así como sus variaciones y detalles de implementación. Dado que el modelo estudiado permite soluciones con ciclos y caminos espurios, ha sido beneficioso dotar a las heurísticas propuestas de un algoritmo capaz de mejorar esas soluciones. Dicho algoritmo se explica en la Sección 5.2.1. Los resultados computacionales obtenidos, tanto en los experimentos realizados para la calibración de los algoritmos como en los llevados a cabo para comparar el desempeño del procedimiento completo, se presentan en la Sección 5.3.

Finalmente el Capítulo 6 presenta algunas conclusiones y diversas lineas posibles de trabajo a futuro.

El presente trabajo contiene también tres apéndices. El Apéndice 7 presenta una revisión de la literatura existente sobre el RSA clasificada según diversos criterios, en particular la variante exacta del problema tratado en la publicación, el enfoque utilizado y el objetivo buscado. Para aquellos trabajos que utilizan modelos de programación lineal entera, hemos discriminado el tipo de formulación presentado así como las variables empleadas. El Apéndice 8 muestra una cantidad representativa de los trabajos relacionados con esta tesis ordenados cronológicamente. Finalmente, el Apéndice 9 describe los detalles de implementación del generador de instancias desarrollado con el fin de llevar a cabo los experimentos de esta tesis.

2

Modelos de programación lineal entera para el problema de ruteo y asignación de espectro

En el presente capítulo, luego de examinar los modelos existentes en la literatura del RSA, exploramos diversas formulaciones alternativas de programación entera para este problema basadas en diferentes ideas de modelado que, a nuestro leal saber y entender, no se han explorado en trabajos anteriores. Este capítulo se basa principalmente en la publicación [14] realizada en colaboración con Federico Bertero.

Cabe señalar que todas las formulaciones presentadas modelan la misma variación del RSA, a pesar de que cada una de ellas tiene asociado un politopo diferente debido a que recurren a distintos conjuntos de variables y restricciones. Esta es una práctica muy común en ILP: generalmente es posible proporcionar más de una formulación para un problema dado y su desempeño práctico es muy difícil de predecir de antemano de manera precisa. Esto lleva a plantear modelos alternativos de ILP para el mismo problema y a evaluar empíricamente los tiempos de ejecución a fin de comparar dichas formulaciones desde un punto de vista computacional. Se acostumbra también emplear la misma función objetivo en todos los modelos considerados, con el fin de lograr una comparación justa.

Como consecuencia de tales experimentos, es habitual elegir la formulación con mejor rendimiento, ya sea en términos del tiempo de ejecución necesario para alcanzar el óptimo o en términos del *gap* de optimalidad alcanzado después de un límite de tiempo preestablecido. Incluso cuando no es posible lograr soluciones óptimas en tiempos de ejecución razonables, las formulaciones con los mejores *gaps* de optimalidad proporcionan un buen punto de partida para el desarrollo de heurísticas, ya que las soluciones factibles obtenidas por ellas tienen las mejores garantías de optimalidad. Estas consideraciones hacen apropiado el desarrollo de formulaciones alternativas y su evaluación empírica, que es el objetivo principal de este capítulo.

La Figura 2.1 presenta la notación utilizada a lo largo de este capítulo, incluyendo los conjuntos, las constantes y las variables.

Conjuntos:

V	conjunto de nodos de la red
$E \subseteq V^2$	conjunto de arcos (enlaces de la red)
D	conjunto de demandas
$\delta^-(j) \subseteq E$	conjunto de arcos entrantes al nodo j
$\delta^+(j) \subseteq E$	conjunto de arcos salientes del nodo j
$\delta(j)$	$\delta^-(j) \cup \delta^+(j)$

Constantes:

$\bar{s} \in \mathbb{N}$	número de *slots* disponibles por arco
$S = \{1, \ldots, \bar{s}\}$	conjunto de *slots* disponibles por arco
$v(d) \in \mathbb{N}$	volumen de la demanda d
$s(d) \in V$	origen de la demanda d
$t(d) \in V$	destino de la demanda d

Variables:

$y_{de} \in \{0,1\}$	igual a 1 si el arco e es asignado a la demanda d, y 0 en caso contrario
$l_d \in S$	el menor *slot* asignado a la demanda d
$r_d \in S$	el mayor *slot* asignado a la demanda d
$p_{dd'} \in \{0,1\}$	igual a 1 si para las demandas d y d', $r_d < l_{d'}$, y 0 en caso contrario
$n_{dd'} \in \{0,1\}$	igual a 1 si los caminos asignados a las demandas d y d' comparten algún arco y $r_d < l_{d'}$, 0 en caso contrario
$r_{ds} \in \{0,1\}$	igual a 1 si s es el último *slot* asignado a la demanda d, y 0 en caso contrario
$l_{ds} \in \{0,1\}$	igual a 1 si s es el primer *slot* asignado a la demanda d, y 0 en caso contrario
$x_{ds} \in \{0,1\}$	igual a 1 si la demanda d usa el *slot* s, y 0 en caso contrario
$u_{des} \in \{0,1\}$	igual a 1 si la demanda d usa el *slot* s en el arco e, y 0 en caso contrario
$l_{des} \in \{0,1\}$	igual a 1 si la demanda d usa el *slot* s en el arco e, y s es el primer *slot* asignado a d, y 0 en caso contrario
$a_{ds} \in \{0,1\}$	igual a 1 si la demanda d usa *slots* mayores a s, y 0 en caso contrario
$b_{ds} \in \{0,1\}$	igual a 1 si la demanda d usa *slots* menores a s, y 0 en caso contrario

Fig. 2.1: Notación utilizada a lo largo del presente capítulo.

2.1. Recorriendo la literatura

Dado que la programación entera es un enfoque natural para abordar este tipo de problemas, se han propuesto varias formulaciones para el RSA como un programa entero [31, 32, 36, 70, 121, 146, 159]. La clasificación más común según el enfoque de modelado es en cuatro categorías dependiendo de las variables utilizadas para establecer las restricciones de continuidad y las variables empleadas para garantizar las restricciones de contigüidad y no solapamiento [68].

2.1.1. Formulaciones enlace-ruta

Una gran cantidad de los modelos existentes utiliza una formulación del tipo *enlace-ruta* –también llamada *arista-ruta* o incluso *arco-ruta*– en la cual, para cada demanda, las variables están asociadas a todas las rutas que conectan su origen con su destino o bien a todos los *lightpaths* que la satisfarían.

- Goścień et al. [51], Klinkowski et al. [71, 73, 74], Velasco et al. [146] y Zotkiewicz et al. [176], presentan un modelo clásico que incluye el conjunto de todas las asignaciones *lightpath*-demanda disponibles, y una variable binaria z_{dl} para cada demanda d y cada *lightpath* l asociado, utilizada para determinar si la demanda d utiliza el *lightpath* l.

- Rottondi et al. [118] y Velasco et al. [146] presentan una segunda formulación donde para cada demanda d se consideran todas las combinaciones posibles de ruta-canal, y una variable binaria x_{pcd} para cada ruta p y canal c asociados con la demanda d, la cual representa el uso tanto de p como de c por la demanda d.

- Archambault et al. [8], Klinkowski et al. [68, 71, 73, 74], Muhammad et al. [94, 95, 96], Xuan et al. [164, 165], Zotkiewicz et al. [176], entre otros, presentan una tercera y más comúnmente utilizada formulación, en la cual la noción de *lightpath* es dividida en la noción de asignación de rutas y la designación de espectro. Estos modelos emplean una variable binaria f_{pd} para cada demanda d y cada posible ruta p, y diversas variables utilizadas para representar los *slots* asignados a cada demanda.

Para encontrar soluciones óptimas con la ayuda de formulaciones enlace-ruta, deben tenerse en cuenta todas los caminos posibles para cada demanda. Como el número de rutas crece exponencialmente con el tamaño de la red, los modelos explícitos son demasiado grandes para realizar los cálculos, por lo que es menester aplicar técnicas de generación de columnas. Sin embargo, los resultados computacionales de por ejemplo [122] muestran que el tamaño de las instancias que se pueden resolver de esa manera es bastante limitado.

2.1.2. Formulaciones enlace-nodo

Una alternativa a las formulaciones del tipo enlace-ruta son las formulaciones conocidas como de tipo *enlace-nodo* –también llamada como *arista-nodo* o *arco-nodo*–, las cuales conducen a modelos menos intuitivos para el ruteo, pero que tienen la ventaja de que la cantidad de variables utilizadas para el este subproblema de encontrar un camino para cada demanda crece sólo polinomialmente con el tamaño de la instancia.

- Bertero et al. [14], Cai et al. [22], Dao Than [36], Hadhbi et al. [53], Klinkowski [68], Velasco et al. [146], Zotkiewicz et al. [176], entre otros, presentan formulaciones que hacen uso de una variable binaria y_{de} la cual representa si la demanda d utiliza el enlace e o no. Pero mientras que en [14, 22, 53] estas variables son parte de formulaciones compactas, es decir, formulaciones con un número polinomial de variables, en [36, 68, 146], por otro lado, también se utiliza un juego exponencial de variables para el resto de restricciones del RSA.

- Una importante cantidad de trabajos propone familias de variables más sofisticadas que y_{de}. Muhammad et al. en [94], por ejemplo, acude a una variable $n_{dd'e}$ para

decir que el primer *slot* utilizado por una demanda d en el arco e es menor que el menor *slot* utilizado por otra demanda d' en el mismo enlace. Paul [108], a su vez, utiliza la familia de variables $m_{dd'e}$ para establecer si dos demandas distintas d y d' comparten o no un arco e en sus caminos. Araújo et al. [7], Klinkowski et al. [68] y Velasco et al. [146, 147] presentan la variable h_{cde} para expresar si a una demanda d en el enlace e se le asigna un rango de *slots* contiguos c.

Ortogonal a la distinción enlace-camino y enlace-nodo, las formulaciones se pueden clasificar de acuerdo a la forma en que establecen las restricciones de contigüidad y no solapamiento.

2.1.3. Formulaciones basadas en canales

Cuando el modelo hace uso de un conjunto precalculado de canales espectrales (*i.e.*, *slots* o subportadoras), donde cada canal se identifica por un subconjunto contiguo de *slots* necesarios para satisfacer una demanda particular, se dice que dicha formulación está *basada en canales*.

- Los modelos presentados por Dao et al. [36], Klinkowski et al. [68] y Tornatore et al. [142] usan una familia de variables k_{cd} que representa la selección de cierto canal c para la demanda d.
- Araújo et al. [7], Klinkowski et al. [68] y Velasco et al. [146, 147] agregan un subíndice al conjunto de variables anterior, *i.e.*, h_{cde}, para indicar si el canal c es usado por la demanda d en un determinado enlace e.

Al igual que con enlace-ruta, todos los posibles canales deben ser tenidos en cuenta para obtener el óptimo. Como la cantidad de estas combinaciones sin ser exponencial suele ser grande, este tipo de formulaciones también requieren aplicar técnicas de generación de columnas.

2.1.4. Formulaciones basadas en *slots*

Una alternativa al cálculo previo de los canales es modelar la distribución del espectro asignando explícitamente todos los *slots*, o únicamente el inicial a cada demanda, y, agregando diversas restricciones, evitar las posibles colisiones entre las distintas demandas que compiten por el uso del espectro. Éstas se conocen como formulaciones *basadas en slots*.

- Una posible forma de relacionar cada demanda d con cada *slot* s, es definiendo una variable x_{ds} para representar si la demanda d utiliza el *slot* s o no, como por ejemplo en los modelos presentados por Patel et al. [104, 105], Li et al. [83] y Bertero et al. [14].
- Cai et al. [22], Christodoulopoulos et al. [30, 31, 32, 33, 34], Klinkowski et al. [68], Muhammad et al. [95, 96] y Bertero et al. [14], entre otros, proponen una familia de variables enteras l_d que indican el menor *slot* utilizado por la demanda d.
- Cai et al. [22] y Bertero et al. [14] agregan una variable r_d para determinar también el mayor *slot* utilizado por d, lo cual es útil cada vez que el máximo volumen utilizado por una demanda no está fijo en el modelo.

- También es posible expresar ambas opciones con familias de variables binarias, i.e, l_{ds} (resp. r_{ds}), que toma valor 1 si y sólo si s es el menor (resp. mayor) *slot* utilizado por la demanda d. Hasta donde sabemos, el único trabajo que recurre a este tipo de variables es [14].

Luego de haber analizado más de 120 trabajos relacionados con el RSA y sus distintas variantes, hemos recopilado en la Tabla 7.7 del Apéndice 7 una gran cantidad de los artículos que presentan formulaciones y los hemos clasificado de acuerdo a los distintos enfoques de modelado, mientras que en las Tablas 7.9 y 7.10, presentamos una lista de las variables utilizadas.

También son muy comunes los enfoques híbridos; por ejemplo, los modelos presentados por Archambault et al. [8], Colares et al. [35], Hadhbi et al. [53], Li et al. [82, 83], Patel et al. [104, 105] y Xuan et al. [163, 165], entre otros, utilizan la variable u_{des} que relaciona demandas con *slots* y con arcos, la cual, como probamos en el presente capítulo, basta por sí sola para modelar el RSA con formulaciones arco-nodo basadas en *slots*. También es común encontrar la combinación opuesta. Como mencionamos, se pueden precomputar los *lightpaths* de cada demanda y tener una sola variable z_{ld} que los relacione. Asimismo, es viable definir una familia de variables que agrupen distintos conjuntos de *lightpaths*, con características similares, por ejemplo, por el menor *slot* utilizado, como hacen las llamadas *configuraciones* presentadas por Enoch y Jaumard en [42, 58]. La cantidad de variables en este último caso crece exponencialmente, por lo que nuevamente se hace necesario recurrir a técnicas de generación de columnas. En la literatura se pueden hallar formulaciones más complejas que pertenecen a tres e incluso a las cuatro categorías antedichas, como el modelo presentado en [164] que utiliza la variable b_{pdes}, la cual representa que el *slot* s en el arco e del camino p es utilizado por la demanda d, o las formulaciones propuestas en [142, 157, 158, 159], que determinan si existe un *lightpath* entre los nodos i y j usando el *slot* s en el enlace e para satisfacer la demanda d, por medio de la variable binaria a_{ijdes}.

2.1.5. Función objetivo

Tratándose del problema de RSA estático, diversas funciones objetivo pueden ser de interés para el operador de la red. Entre las más comunes se pueden enumerar las que buscan minimizar alguno de los siguientes valores.

- Espectro máximo: el número de *slots* requerido en cada arco de la red para ser capaz de satisfacer todas las demandas (también llamado *nivel de congestion* [36]). También puede encontrarse como *intervalo de espectro mínimo*, *i.e.*, la diferencia entre el máximo y el mínimo *slot* utilizado entre todos los arcos.
- Espectro general: el número total de *slots* asignados en todos los enlaces de red.
- Espectro promedio: la utilización promedio del espectro por enlace.
- Costo de la red: la cantidad de transceptores espectrales o espaciales, el número de fibras usadas, la cantidad de nodos ópticos *express* o nodos *add/drop*, los amplificadores SDM, el número de multiplexores y demultiplexores, el consumo de energía, entre otros.
- Longitud de las rutas: la suma de todas las longitudes de las rutas asignadas. Podría interpretarse como un caso particular de costo de red.

Sin embargo, en muchos casos estamos interesados en más de una métrica, como por ejemplo la cantidad de transpondedores junto con las longitudes de onda utilizadas, u otras combinaciones de las métricas anteriores [36, 75, 79, 82, 83, 144]. En un problema de optimización multiobjetivo, suele ocurrir que ninguna solución logra simultáneamente la optimización de todas las métricas. En este contexto es útil considerar el frente de soluciones *no dominado* o *Pareto*, que es el conjunto de soluciones que no se pueden mejorar en un objetivo sin deteriorarse en al menos uno de los demás (ver Figura 2.2).

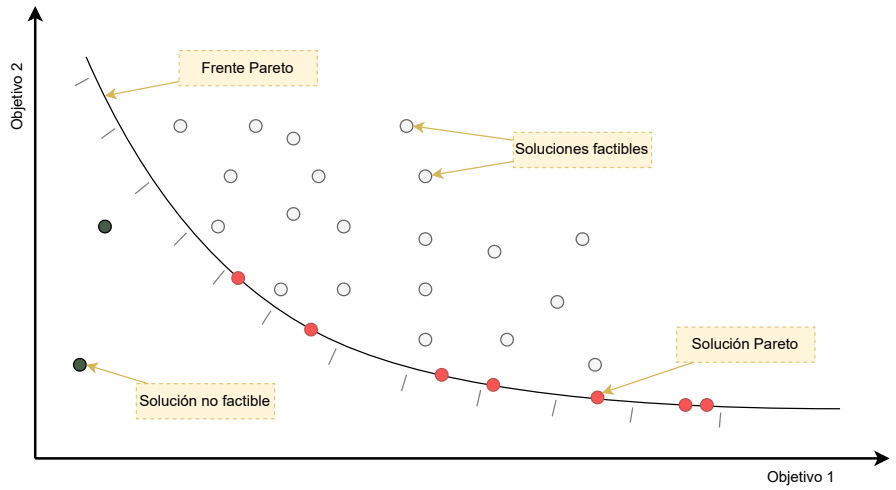


Fig. 2.2: El frente de soluciones Pareto de un problema de optimización multiobjetivo.

La Tabla 7.4 del Apéndice 7 presenta un estudio basado en más de 120 trabajos clasificados por sus funciones objetivo.

El consumo de energía tiene un efecto profundo en el costo operativo de una red de transporte óptica [4, 85, 151, 174]. Especialmente en las grandes redes centrales, donde los recursos se aprovisionan para atender el tráfico en las horas pico, se consume una cantidad significativa de energía incluso durante las horas de menor actividad [11, 85, 98, 143, 174]. El consumo de energía de dichas redes se puede reducir ajustando dinámicamente la capacidad de los recursos [11, 37, 85, 98, 143, 151]. Si las rutas son seleccionadas cuidadosamente, es posible identificar y apagar algunos enlaces innecesarios, reduciendo así el consumo de energía.

Una forma de buscar indirectamente este objetivo es reduciendo la longitud de las rutas usadas para satisfacer las demandas. Según la bibliografía consultada, de los objetivos más habituales, el que busca minimizar el máximo espectro utilizado parece ser el más difícil de abordar en la práctica, mientras que a nuestro juicio el que busca minimizar la longitud del camino asignado a cada demanda es uno de los más sencillos. Si la capacidad de los arcos es suficientemente grande, asignar la ruta más corta a cada demanda da como resultado una solución óptima. Del mismo modo, si la capacidad $\bar{s}$ es suficientemente ajustada, una solución a este problema es, en cierto modo, un punto de partida para el otro.

Esto último, sumado a que el objetivo principal del presente trabajo es comprender más profundamente el RSA con la idea de dar un punto de partida a estudios poliedrales en los que la función objetivo pasa a un segundo plano, es lo que nos motivó a buscar minimizar la longitud de las rutas asignadas a cada demanda, al igual que en los trabajos

[53, 79, 91, 100, 107, 136], entre otros.

2.2. Formulaciones existentes

Dado que estamos interesados en obtener soluciones óptimas globales con formulaciones compactas de ILP, en esta sección presentamos dos formulaciones compactas, tomadas de la literatura mencionada, que no utilizan subconjuntos predefinidos de rutas. Para que la comparación computacional sea justa, adaptamos dichos modelos para que utilicen la misma función objetivo que las formulaciones de ILP introducidas en el presente trabajo.

2.2.1. Formulación enlace-nodo basada en *slots* (NLS)

Esta formulación es una de las presentadas en [176], con los nombres de variables adaptados a los que utilizamos en este trabajo. Los autores asocian arcos y demandas con el fin de definir caminos, y la relación entre demandas y *slots* es modelada particionando los *slots* disponibles para cada demanda en tres grupos distintos: los *slots* menores a los utilizados, los utilizados, y los mayores a los utilizados. Con el fin de obtener dichos grupos, para cada demanda $d \in D$ y cada arco $e \in E$, se usa la variable y_{de} de modo tal que $y_{de} = 1$ si y sólo si la ruta asociada a la demanda d usa el arco e. También, para cada demanda $d \in D$, los autores recurren a las variables binarias a_{ds}, x_{ds}, b_{ds}, de forma tal que $a_{ds} = 1$ si y sólo si la demanda d usa *slots* mayores que s, $b_{ds} = 1$ si y sólo si la demanda d usa *slots* menores que s, y $x_{ds} = 1$ si y sólo si la demanda d usa el *slot* s. Con esta configuración, el siguiente modelo de ILP es una formulación para el RSA.

$$\min \quad \sum_{d \in D} \sum_{e \in E} y_{de} \tag{2.1}$$

$$\text{s.t.} \quad \sum_{e \in \delta^+(s(d))} y_{de} - \sum_{e \in \delta^-(s(d))} y_{de} = 1 \qquad \forall d \in D \tag{2.2}$$

$$\sum_{e \in \delta^-(j)} y_{de} - \sum_{e \in \delta^+(j)} y_{de} = 0 \qquad \forall d \in D, \forall v \in V \setminus \{s(d), t(d)\} \tag{2.3}$$

$$y_{de} + x_{ds} + y_{d'e} + x_{d's} \leq 3 \qquad \forall d, d' \in D,\ d \neq d', \forall e \in E, \forall s \in S \tag{2.4}$$

$$a_{ds} \geq a_{d,s+1} \qquad \forall d \in D, \forall s \in S \setminus \{\bar{s}\} \tag{2.5}$$

$$b_{ds} \geq b_{d,s-1} \qquad \forall d \in D, \forall s \in S \setminus \{1\} \tag{2.6}$$

$$x_{ds} + a_{ds} + b_{ds} = 1 \qquad \forall d \in D, \forall s \in S \tag{2.7}$$

$$\sum_{s \in S} x_{ds} \geq v(d) \qquad \forall d \in D \tag{2.8}$$

$$x_{ds}, a_{ds}, b_{ds} \in \{0, 1\} \qquad \forall d \in D, \forall s \in S \tag{2.9}$$

$$y_{de} \in \{0, 1\} \qquad \forall d \in D, \forall e \in E. \tag{2.10}$$

La función objetivo (2.1) busca minimizar el numero de enlaces utilizados. Las restricciones siguientes, *i.e.*, (2.2) y (2.3), expresan las restricciones clásicas de conservación de flujo. Las restricciones (2.4) aseguran que no exista un *slot* compartido por dos demandas que usen el mismo arco. Mientras las restricciones (2.5) a (2.7) garantizan que los canales estén formados por *slots* consecutivos, la restricción (2.8) se encarga de la satisfacción de las demandas.

2.2.2. Formulación enlace-nodo basada en canales (NL-CA)

Esta formulación es una adaptación de la presentada en [146]. La misma está basada en la asignación de canales, con lo cual remueven la restricción de contigüidad del espectro, intentando reducir la complejidad del problema. En esta formulación se precomputan todos los canales posibles conformados por *slots* consecutivos. Para hallar una solución óptima es necesario considerar para cada demanda todos los canales de este conjunto capaces de satisfacerla (*i.e.*, para cada demanda d es menester recorrer todos los canales de $v(d)$ *slots* consecutivos que puedan ser definidos sobre el conjunto S). En aras de la consistencia, presentamos este modelo en términos de las variables utilizadas en este trabajo, por lo que reemplazamos las variables w utilizadas en [146] por las variables l consideradas en este trabajo, sin por ello alterar el modelo. Dicho intercambio es posible dado que asignar el canal $[s, \ldots, s + v(d) - 1]$ a la demanda d es equivalente a decir que s es el primer *slot* asignado a d. Respecto a la función objetivo, removimos las variables z –utilizadas para indicar enlaces ocupados– junto con una familia de restricciones que asigna el valor 1 a dichas variables cada vez que un enlace es usado, por lo que la comparación con los modelos presentados en este trabajo creemos que es justa. En estos términos, el siguiente modelo de ILP es la adaptación de la formulación para el RSA presentada en [146].

$$\min \quad \sum_{d \in D} \sum_{e \in E} \sum_{s=1}^{\bar{s}-v(d)+1} l_{des} \tag{2.11}$$

$$\text{s.t.} \quad \sum_{e \in \delta(v)} \sum_{s=1}^{\bar{s}-v(d)+1} l_{des} = 1 \qquad \forall d \in D,\ \forall v \in \{s(d), t(d)\} \tag{2.12}$$

$$\sum_{e \in \delta(v)} \sum_{s=1}^{\bar{s}-v(d)+1} l_{des} \leq 2 \qquad \forall d \in D,\ \forall v \notin \{s(d), t(d)\} \tag{2.13}$$

$$\sum_{e' \in \delta(v) \setminus \{e\}} l_{de's} \geq l_{des} \qquad \begin{array}{r} \forall d \in D,\ \forall s \in \{1, \ldots, \bar{s} - v(d) + 1\}, \\ \forall v \in V \setminus \{s(d), t(d)\},\ \forall e \in \delta(v) \end{array} \tag{2.14}$$

$$l_{des} \in \{0, 1\} \qquad \forall d \in D,\ \forall e \in E,\ \forall s \in S \tag{2.15}$$

La función objetivo (2.11) pretende minimizar el número de enlaces utilizados. Las restricciones (2.12)-(2.14) resuelven el ruteo a través de la topología y asignan a cada demanda el primer *slot* utilizado. Las restricciones (2.12) aseguran que cada demanda tenga un único *slot* designado como primer *slot* tanto en los enlaces salientes del origen como los entrantes al destino. Finalmente las restricciones (2.13) y (2.14) realizan el ruteo en los nodos intermedios y fuerzan la restricción de continuidad del espectro en dichos nodos.

2.3. Modelos propuestos en esta tesis

El RSA consiste en el establecimiento de una ruta y una secuencia de *slots* para cada demanda, de modo tal que se respeten las restricciones dadas. Como mencionamos anteriormente, en la actualidad existen cuatro categorías posibles dentro de un enfoque de programación entera en función de las variables utilizadas. Todos los modelos propuestos en el presente trabajo pertenecen a la categoría de enlace-nodo basada en *slots*. Dentro

de esta clase en particular, consideramos dos formas naturales de modelar una solución: o bien representamos el ruteo con un conjunto de variables y la asignación de *slots* con un segundo conjunto, o representamos ambas decisiones con un solo conjunto de variables. Primero enunciamos los modelos que utilizan un conjunto de variables para la asignación de demandas a enlaces y otro para la asignación de demandas a *slots*, y luego presentamos las formulaciones que utilizan un único conjunto de variables para la asignación de demandas a enlaces y *slots*.

Dentro de cada categoría, también consideramos dos modos distintos de lidiar con la asignación *demanda-canal*. Por un lado, hacemos una asignación explícita del par *demand-slot* utilizando variables que indican si cada *slot* es usado o no por cada demanda. Por otro lado, hacemos una asignación implícita de *demanda-slot-rango* haciendo uso de variables que indican el primero y/o el último *slot* utilizado por cada demanda. Como el volumen de las demandas está fijo, si por ejemplo s es el primer *slot* usado por la demanda d, entonces los *slots* $s, \ldots, s + v(d) - 1$ serán también asignados a la demanda d.

De acuerdo a los comentarios previos, podemos agrupar nuestras formulaciones ILP en cuatro familias –siempre dentro de la categoría enlace-nodo basada en slots–. Por un lado, las familias que recurren a diferentes conjuntos de variables para el ruteo y para la asignación de espectro son clasificadas como formulaciones de asignación *demanda-rango* (DR) o como formulaciones de asignación *demanda-slot* (DS), dependiendo de la distinción mencionada en el párrafo anterior. Por otro lado, las familias que contienen un único conjunto de variables son clasificadas como formulaciones de asignación *demanda-slot-enlace* (DSL) o como formulaciones de asignación *demanda-rango-enlace* (DRL), nuevamente dependiendo de la distinción mencionada en el párrafo anterior. Dadas sus variables, las formulaciones de la bibliografía presentadas anteriormente, *i.e.*, NLS y NL-CA, pueden ser clasificadas en las familias *demanda-slot* (DS) y *demanda-rango-enlace* (DRL), respectivamente.

En cada una de las siguientes subsecciones presentamos una de las familias mencionadas anteriormente con una formulación de ILP base asociada y un conjunto de modificaciones posibles para dicha formulación base. Cada una de estas modificaciones genera un nuevo modelo que será probado y analizado en la Sección 2.4. Sin embargo, antes de comenzar a presentar los nuevos modelos, presentamos un teorema que será usado en varios de ellos y en los capítulos siguientes.

2.3.1. Teorema de unos consecutivos

Fijemos una demanda $d \in D$ y un arco $e \in E$ usado por d. Llamemos x_s a la variable binaria que toma valor 1 si y sólo si el *slot* $s \in S$ es usado por d. Por comodidad, renombremos $h = v(d)$ para denotar el volumen de la demanda d, con $1 \leq h \leq \bar{s}$, y consideremos las siguientes restricciones.

$$\sum_{\substack{s \in S: \\ s \equiv i \ (h)}} x_s \ = \ 1 \qquad \forall i \in \{1, \ldots, h\} \tag{2.16}$$

$$\sum_{\substack{s \in \{1,\ldots,i\}: \\ s \equiv i \ (h)}} x_s \ \geq \ \sum_{\substack{s \in \{1,\ldots,i-1\}: \\ s+1 \equiv i \ (h)}} x_s \qquad \forall i \in \{1, \ldots, \bar{s} + 1 - h\}. \tag{2.17}$$

Teorema 2.3.1. *Si $x^* \in \{0,1\}^{\bar{s}}$, las restricciones (2.16) y (2.17) aseguran que hay exactamente h elementos consecutivos en el vector binario $(x^*_1, \ldots, x^*_{\bar{s}})$ tomando el valor 1.*

Demostración. Utilizando las restricciones (2.17) y una propiedad de las restricciones (2.16) podemos probar que hay al menos h variables consecutivas tomando el valor 1, y haciendo uso de (2.16) podemos mostrar que esas h variables son las únicas que toman dicho valor, *i.e.*, el resto de los elementos del vector binario x^* valen 0.

Para $i = 1, \ldots, \bar{s} + 1 - h$, sea C_i la restriccion (2.17) asociada al índice i, y sean LHS_i y RHS_i el lado izquierdo y derecho de C_i, respectivamente, *i.e.*,

$$C_i \ : \ LHS_i \ \geq \ RHS_i.$$

Como el término $RHS_1 = 0$ y como el conjunto de índices de LHS_i es igual al conjunto de índices de RHS_{i+1} para todo i en $\{1, \ldots, s - h\}$, es decir,

$$LHS_{\bar{s}+1-h} \geq RHS_{\bar{s}+1-h} = LHS_{\bar{s}-h} \geq \cdots \geq RHS_2 = LHS_1 \geq RHS_1 = 0, \tag{2.18}$$

entonces podemos afirmar que

$$LHS_{i+1} \ \geq \ LHS_i \ \forall i \in \{1, \ldots, \bar{s} - h\}. \tag{2.19}$$

Sea $L_i = \{s : s \in \{1, \ldots, i\}, \ s \equiv i \ (h)\}$ el conjunto de índices de LHS_i. El índice i es el mayor elemento en L_i, *i.e.*, $i = \text{máx}\{i' : x^*_{i'} \in LHS_i\}$. Asimismo como $i \notin L_{i+1}, \ldots, i \notin L_{i+h-1}$, para $q = 0, \ldots, h-1$ e $i = 1, \ldots, \bar{s} - q$, concluímos que el único índice mayor a i en L_{i+q} es $i + q$, *i.e.*,

$$\{i' : i' \geq i, \ i' \in L_{i+q}\} = \{i + q\}. \tag{2.20}$$

Ahora usemos esos resultados para probar que cualquier solución factible que satisface las restricciones (2.16) y (2.17) tiene al menos h variables consecutivas valiendo 1. Sea A_i el lado izquierdo de la restriccion (2.16) asociada al índice i, y sea $\alpha_i = \{s : s \in S, \ s \equiv i \ (h)\}$. Dado que $L_i \subseteq \alpha_i$ entonces las restricciones (2.16) implican:

$$LHS_i \ \leq A_i = 1 \quad \forall i \in \{1, \ldots, \bar{s} + 1 - h\}. \tag{2.21}$$

Sea p el menor índice tal que $x^*_p = 1$. Sea i^* el menor índice tal que $x^*_p \in LHS_{i^*}$. A causa de (2.21) y como $x^*_p = 1$ entonces $LHS_{i^*} = 1$. De igual modo, debido a (2.19) y (2.21), $LHS_k = 1$ para cada $k \geq i^*$. En particular,

$$LHS_k = 1 \quad \forall k \in \{i^*, \ldots, i^* + h - 1\}.$$

Por la definición de p, tenemos $x^*_{p'} = 0$ para todo $p' < p$. Esto implica que en cada LHS_k, tal que $i^* < k < i^* + h$, solamente las variables con índices mayores a p pueden tomar el valor 1; pero a causa de (2.20), algunas de estas son $\{x^*_{p+1}, \ldots, x^*_{p+h-1}\}$. Por lo tanto, tenemos al menos h variables consecutivas tomando valor 1.

Consideremos ahora la unión de los conjuntos $\{s \in S : \ s \equiv i \ (h)\}$ para $i \in \{1, \ldots, h\}$. Dado que todos sus elementos son disjuntos, entonces,

$$\bigcup_{i=1}^{h} \{s \in S : \ s \equiv i \ (h)\} = \{s \in S : \ s \equiv i \ (h), \ i \in \{1 \ldots h\}\} = S.$$

Esto implica que hay a lo sumo h variables tomando valor 1, *i.e.*,

$$\sum_{i \in \{1, \ldots, h\}} \ \sum_{\substack{s \in S: \\ s \ \equiv \ i \ (h)}} x^*_s \ \leq \ h.$$

Dado que hay al menos y a lo sumo h variables consecutivas tomando valor 1, hay exactamente h variables tomando dicho valor, con lo cual concluimos la demostración. □

2.3.2. Formulaciones de asignación demanda-rango (DR)

Formulación base (DR-BF)

En este primer modelo asociamos arcos con demandas para definir los caminos, y modelamos el primero y el último *slot* reservado para cada demanda. Con este fin, para cada demanda $d \in D$ y cada arco $e \in E$, utilizamos la variable binaria y_{de} de modo tal que $y_{de} = 1$ si y sólo si el camino asociado a la demanda d usa el arco e. Asimismo, para cada demanda $d \in D$ usamos las variables enteras $l_d, r_d \in S$, de forma tal que el intervalo de *slots* asignado a la demanda d está dado por $[l_d, r_d]$. Finalmente, en orden a garantizar que los intervalos asociados a cada par de demandas con rutas no disjuntas no se solapen, recurrimos a la variable binaria $p_{dd'}$, de modo tal que para dos demandas d y d', si $r_d < l_{d'}$ entonces $p_{dd'} = 1$. En este escenario, el siguiente modelo de ILP es una formulación del RSA.

$$\min \quad \sum_{d \in D} \sum_{e \in E} y_{de} \tag{2.22}$$

$$\text{s.a.} \quad \sum_{e \in \delta^-(j)} y_{de} - \sum_{e \in \delta^+(j)} y_{de} = \begin{cases} -1 & \text{if } j = s(d) \\ 1 & \text{if } j = t(d) \\ 0 & \text{en otro caso} \end{cases} \qquad \forall j \in V, \forall d \in D \tag{2.23}$$

$$p_{dd'} + p_{d'd} = 1 \qquad \forall d, d' \in D,\ d \neq d' \tag{2.24}$$

$$\bar{s}(3 - p_{dd'} - y_{de} - y_{d'e}) \geq r_d - l_{d'} + 1 \qquad \forall d, d' \in D,\ d \neq d',\ \forall e \in E \tag{2.25}$$

$$r_d - l_d + 1 = v(d) \qquad \forall d \in D \tag{2.26}$$

$$1 \leq l_d \leq r_d \leq \bar{s} \qquad \forall d \in D \tag{2.27}$$

$$y_{de} \in \{0, 1\} \qquad \forall d \in D,\ \forall e \in E \tag{2.28}$$

$$p_{dd'} \in \{0, 1\} \qquad \forall d, d' \in D,\ d \neq d' \tag{2.29}$$

$$l_d, r_d \in \mathbb{Z} \qquad \forall d \in D. \tag{2.30}$$

Con la función objetivo (2.22) lo que intentamos es minimizar el número total de arcos en todos los caminos asignados a las demandas. Las restricciones (2.23) aseguran que las variables y asociadas a cada demanda, en caso de no valer 0, determinan un camino desde el origen hasta el destino de la demanda, imponiendo las restricciones de conservación de flujo a cada nodo de la red. Es interesante notar que (2.23) no previenen la formación de ciclos espurios. Dado que la aparición de estas estructuras no afecta a la factibilidad (las cuales, a su vez, serán eliminadas por función objetivo en el óptimo), mientras que la adición de familias de restricciones (la mayoría exponenciales) enfocadas en eliminar ciclos pueden afectar los tiempos de resolución, no incluímos restricciones que explícitamente prohiban dichas situaciones.

Las restricciones (2.24) y (2.25) garantizan que si las demandas d y d' comparten un arco, entonces $r_d < l_{d'}$, o bien $r_{d'} < l_d$, evitando así el solapamiento. De hecho, si las demandas d y d' comparten el arco e, entonces (2.24) implica que $p_{dd'} + y_{de} + y_{d'e} = 3$ ó $p_{d'd} + y_{de} + y_{d'e} = 3$, imponiendo por lo tanto la condición deseada. Finalmente, con las restricciones (2.26) y (2.27) se obliga a que el número de *slots* asignados a cada demanda satisfaga su volumen requerido.

Límite con un *slot* (DR-OSB)

Las restricciones (2.26) nos permiten eliminar las variables r, ya que éstas pueden ser escritas en términos de las variables l. Por lo tanto, puede obtenerse una re-estructuración de la formulación previa mediante el reemplazo de las restricciones (2.25) y (2.26) por las desigualdades

$$l_{d'} \geq l_d + v(d) - \bar{s}(3 - p_{dd'} - y_{de} - y_{d'e}) \qquad \forall d, d' \in D,\ d \neq d',\ \forall e \in E. \tag{2.31}$$

Variables alternativas que incluyen orden (DR-AOV)

Podría ser útil reemplazar las variables p por variables que ayuden a garantizar que no se solapen los intervalos asociados con pares de demandas cuyas rutas no sean disjuntas en arcos. La variable $n_{dd'}$ toma valor 1 si los caminos asignados a d y d' comparten al menos un arco y $r_d < l_{d'}$. Asignando valores a las variables de forma tal que se satisfagan (2.23), (2.26), (2.27), (2.28), (2.30), y las siguientes restricciones, obtenemos una solución factible del RSA.

$$n_{dd'} + n_{d'd} \geq y_{de} + y_{d'e} - 1 \qquad \forall d, d' \in D,\ d \neq d',\ \forall e \in E \tag{2.32}$$

$$r_d + 1 \leq l_{d'} + \bar{s}(1 - n_{dd'}) \qquad \forall d, d' \in D,\ d \neq d' \tag{2.33}$$

$$n_{dd'} \in \{0, 1\} \qquad \forall d, d' \in D,\ d \neq d'. \tag{2.34}$$

Las restricciones (2.32) especifican que si dos demandas distintas d y d' comparten algún enlace (*i.e.*, existe algún e con $y_{de} = y_{d'e} = 1$) entonces o la variable $n_{dd'} = 1$ o bien la variable simétrica $n_{d'd} = 1$. Las restricciones (2.33) imponen que si $n_{dd'} = 1$ entonces debe cumplirse que $r_d + 1 \leq l_{d'}$, implicando que los *slots* asignados a d' tienen índices mayores que los *slots* asignados a d. Por otro lado, si la variable $n_{d'd} = 1$ entonces debe ocurrir lo opuesto. Como ambos eventos son disjuntos, no podemos tener $n_{dd'} = 1$ y $n_{d'd} = 1$ de manera simultánea, con lo que garantizamos que cualquier par de demandas diferentes que compartan al menos un enlace no serán asignadas al mismo *slot*, cumpliendo así con la restricción de no solapamiento.

Asignación binaria de *slots* (DR-BSA)

Una alternativa para la asignación de *slots* a demandas a través de las variables enteras l y r, es el uso de variables binarias. En la presente variación implementamos esta asignación mediante el uso de la variable binaria l_{ds} para cada demanda $d \in D$ y cada *slot* $s \in \{1, \ldots, \bar{s} - v(d) + 1\}$, de modo tal que $l_{ds} = 1$ si s es el primer *slot* asignado a la demanda d, y $l_{ds} = 0$ en caso contrario. Si $l_{ds} = 1$, entonces los *slots* $s, \ldots, s + v(d) - 1$ son asignados a la demanda d. El uso de estas variables nos permite enunciar una nueva formulación que utiliza, además de (2.23) y (2.29), las siguientes restricciones:

$$\sum_{s=1}^{\bar{s}-v(d)+1} l_{ds} = 1 \qquad \forall d \in D \tag{2.35}$$

$$\sum_{s'=s}^{s+v(d)-1} l_{d's'} \leq 3 - y_{de} - y_{d'e} - l_{ds} \qquad \begin{array}{r} \forall d, d' \in D,\ d \neq d', \forall e \in E, \\ \forall s \in \{1, \ldots, \bar{s} - v(d)\} \end{array} \tag{2.36}$$

$$l_{ds} \in \{0, 1\} \qquad \forall d \in D,\ \forall s \in S. \tag{2.37}$$

Las restricciones (2.35) aseguran que el intervalo de asignado a cada demanda tiene exactamente un *primer slot*. Para representar la restricción de no solapamiento, las restricciones (2.36) garantizan que si s es el primer *slot* asignado a la demanda d, entonces los siguientes $v(d)-1$ *slots* en los arcos usados por d deben ser también asignados a d. Esto se logra forzando a que ninguno de ellos sea el primer *slot* de alguna otra demanda que comparta algún enlace con d.

Restricciones más ajustadas (DR-SC)

Las restricciones (2.36) nos aseguran, en una única expresión, que los *slots* asignados a la demanda d sean consecutivos. Podemos formular la misma restricción reemplazando la suma en (2.36) por $v(d)$ nuevas desigualdades, como se muestra a continuación.

$$l_{d'(s+i)} \leq 3 - y_{de} - y_{d'e} - l_{ds} \qquad \begin{array}{r} \forall d, d' \in D,\ d \neq d',\ \forall e \in E, \\ \forall s \in \{1, \ldots, \bar{s} - v(d) + 1\}, \\ \forall i \in \{0, \ldots, v(d) - 1\}. \end{array} \tag{2.38}$$

Las restricciones (2.38) imponen que si $y_{de} + y_{d'e} + l_{ds} = 3$ entonces $l_{d's'} = 0$ para $s' = s, \ldots, s + v(d) - 1$. En otras palabras, si dos demandas diferentes d y d' comparten un arco e, y s es el primer *slot* usado por d, entonces ningún *slot* en $\{s, \ldots, s + v(d) - 1\}$ puede ser el primer *slot* de d'.

2.3.3. Formulaciones de asignación demanda-*slot* (DS)

Formulación base (DS-BF)

Ésta es la primera de la segunda familia de modelos, que representa explícitamente la asignación de cada *slot* a cada demanda en lugar de asginar rangos de *slots* mediante el uso de las variables l y r. Para ello utilizamos una variable binaria x de la siguiente manera. Para cada demanda $d \in D$ y cada *slot* $s \in S$, la variable binaria x_{ds} toma el valor 1 si la demanda d usa el *slot* s, y x_{ds} toma el valor 0 en caso contrario. Para asociar arcos con demandas, seguimos utilizando las variables binarias y al igual que en los modelos anteriores. Por este motivo, en esta primera familia podemos hacer nuevamente uso de las restricciones de flujo (2.23) para garantizar que los arcos utilizados formen caminos entre el origen y el destino de alguna demanda. En este escenario, el siguiente modelo de ILP es una formulación para el RSA.

$$\min \quad \sum_{d \in D} \sum_{e \in E} y_{de} \tag{2.39}$$

s.t. (2.23)

$$y_{de} + x_{ds} \leq 3 - y_{d'e} - x_{d's} \qquad \forall d, d' \in D,\ d \neq d',\ \forall e \in E,\ \forall s \in S \tag{2.40}$$

$$\sum_{s'=1+s-v(d)}^{s} x_{ds'} \geq v(d)(x_{ds} - x_{d,s+1}) \qquad \forall d \in D,\ \forall s \in \{v(d), \ldots, \bar{s}\} \tag{2.41}$$

$$x_{d,\bar{s}+1} = 0 \qquad \forall d \in D \tag{2.42}$$

$$\sum_{s \in S} x_{ds} \geq v(d) \qquad \forall d \in D. \tag{2.43}$$

$$y_{de} \in \{0, 1\} \qquad \forall d \in D,\ \forall e \in E \tag{2.44}$$

$$x_{ds} \in \{0, 1\} \qquad \forall d \in D,\ \forall s \in S \cup \{\bar{s} + 1\}. \tag{2.45}$$

Las restricciones (2.23) son las restricciones de conservación de flujo utilizadas en los modelos anteriores para definir un camino desde el origen hasta el destino de cada demanda. Para asegurar que si dos demandas comparten un arco entonces no comparten *slot*, necesitamos que al menos una de las variables y_{de}, x_{ds}, $y_{d'e}$ y $x_{d's}$ no esté activada, *i.e.*, no tome el valor 1, y las restricciones (2.40) fuerzan esa condición. A su vez, las restricciones (2.41) y (2.42) garantizan la contiguidad de *slots*. Las restricciones (2.41) aseguran que si un *slot* s es asignado a la demanda d pero el *slot* vecino $s+1$ no es asignado a d, entonces los $v(d)-1$ *slots* a la izquierda de s deben ser también asignados a d. En orden a simplificar la notación, asumimos la existencia de la variable ficticia $x_{d,\bar{s}+1}$ la cual toma valor 0. Finalmente, con el fin de satisfacer el volumen requerido por cada demanda, incluímos las restricciones (2.43), las cuales fuerzan que la cantidad de *slots* asignados a cada demanda sea al menos igual al volumen requerido por la misma.

Restricciones de contigüidad alternativas (DS-ACC)

Si consideramos los slots de a pares, las restricciones (2.41) se pueden reemplazar por las siguientes:

$$x_{ds} + x_{ds'} \;\leq\; x_{d,s+1} + 1 \qquad \forall s, s' \in S,\ s' > s,\ \forall d \in D. \tag{2.46}$$

Las restricciones (2.46) imponen que si $x_{ds} - x_{d,s+1} = 1$ entonces s debe ser el último *slot* asignado a la demanda d y por lo tanto $x_{ds'} = 0$ para $s' > s$. Esto nos asegura que todos los *slots* asignados a la demanda son consecutivos.

Restricciones de contigüidad más ajustadas (DS-SCC)

Estas nuevas restricciones parten de otra forma de enunciar las restricciones (2.41).

$$\sum_{\substack{s \in S: \\ s \,\equiv\, j \ (v(d))}} x_{ds} \;=\; 1 \qquad \forall d \in D,\ \forall j \in \{1, \ldots, v(d)\} \tag{2.47}$$

$$\sum_{\substack{s \in \{1,\ldots,i\}: \\ s \,\equiv\, i \ (v(d))}} x_{ds} \;\geq\; \sum_{\substack{s \in \{1,\ldots,i-1\}: \\ s+1 \,\equiv\, i \ (v(d))}} x_{ds} \qquad \forall d \in D,\ \forall i \in \{1, \ldots, \bar{s} + 1 - v(d)\}. \tag{2.48}$$

Debido al Teorema 2.3.1, las restricciones (2.47) y (2.48) aseguran que para cada demanda d y cada arco e hay exactamente $v(d)$ variables consecutivas en el vector x^* tomando valor 1, y que las variables restantes toman valor 0.

2.3.4. Formulaciones demanda-*slot*-enlace (DSL)

Formulación base (DSL-BF)

Como fue comentado anteriormente, es posible modelar el ruteo y la asignación de espectro con una única familia de variables. Para cada demanda $d \in D$, cada arco $e \in E$ y cada *slot* $s \in S$, introducimos la variable binaria u_{des}, de modo tal que $u_{des} = 1$ si y sólo si la demanda d utiliza el *slot* s sobre el arco e. Por razones técnicas, como con el modelo DR-BF presentado en la Sección 2.3.2, aquí también consideramos la variable ficticia $u_{de,\bar{s}+1}$ que siempre toma valor 0 para cada demanda $d \in D$ y cada arco $e \in E$.

En este marco, la formulación DSL-BF para el RSA está dada por el siguiente programa entero.

$$\min \quad \sum_{d\in D}\sum_{e\in E}\sum_{s\in S} u_{des}/v(d) \tag{2.49}$$

$$\text{s.t.} \quad \sum_{e\in\delta^-(j)} u_{des} - \sum_{e\in\delta^+(j)} u_{des} = 0 \qquad \forall d\in D,\, \forall j\in V\backslash\{s(d),t(d)\},\, \forall s\in S \tag{2.50}$$

$$\sum_{e\in\delta^+(s(d))}\sum_{s\in S} u_{des} \geq v(d) \qquad \forall d\in D \tag{2.51}$$

$$\sum_{e\in\delta^-(s(d))}\sum_{s\in S} u_{des} = 0 \qquad \forall d\in D \tag{2.52}$$

$$\sum_{d\in D} u_{des} \leq 1 \qquad \forall e\in E,\, \forall s\in S \tag{2.53}$$

$$u_{de,\bar{s}+1} = 0 \qquad \forall d\in D,\, \forall e\in E \tag{2.54}$$

$$v(d)(u_{des} - u_{de,s+1}) \leq \sum_{s'=f}^{s} u_{des'} \qquad \begin{array}{r}\forall d\in D,\, \forall e\in E,\, \forall s\in S,\\ f=\text{máx}\{1, s-v(d)+1\}\end{array} \tag{2.55}$$

$$u_{des}\in\{0,1\} \qquad \forall d\in D,\, \forall e\in E,\, \forall s\in S\cup\{\bar{s}+1\}. \tag{2.56}$$

La función objetivo (2.49) pide que la longitud total de los caminos asociados a los *lightpaths* sea mínima. Esto a su vez prohíbe la aparición de ciclos y caminos espurios en cualquier solución óptima. Las restricciones (2.50) imponen la conservación de flujo para cada demanda en cada nodo de la red, excepto para los nodos origen y destino asociados a la demanda. Las restricciones (2.51) aseguran que el *lightpath* asignado a cada demanda tenga mínimamente el número de *slots* requerido por la misma, mientras que las restricciones (2.52) prohíben arcos entrantes al nodo origen de cada demanda. Las restricciones (2.53) garantizan que dos *lightpaths* diferentes no se solapen. Finalmente, las restricciones (2.55) aseguran la contigüidad de los *slots*: si $u_{des} - u_{de,s+1} = 1$ entonces s es el último *slot* utilizado por la demanda d en el arco e, y en este caso pedimos a todas las variables u_{dei}, con $i = s - v(d) + 1, \ldots, s$, que estén activas. Estas restricciones aprovechan el hecho de que la variable ficticia $u_{de,\bar{s}+1}$ toma valor 0 para $d\in D$ y $e\in E$.

Notar que al tomar $f = \text{máx}\{1, s - v(d) + 1\}$ en lugar de simplemente $f = v(d)$ en las restricciones (2.55), prohibimos que una demanda d sea satisfecha utilizando varios caminos cuando en éstos se utilizan *slots* con subíndices menores que $v(d)$.

Restricciones alternativas de satisfactibilidad y contigüidad (DSL-ASCC)

Las restricciones (2.54) y (2.55), que aseguran la contigüidad y satisfactibilidad de las demandas, pueden ser reemplazadas por las siguientes restricciones alternativas.

$$v(d)\; u_{des} \leq \sum_{s'\in S} u_{des'} \qquad \forall d\in D,\, \forall e\in E,\, \forall s\in S \tag{2.57}$$

$$u_{des_1} + u_{des_2} \leq u_{de(s_1+1)} + 1 \qquad \forall d\in D,\, \forall e\in E,\, \forall s_1, s_2\in S,\, s_2 > s_1. \tag{2.58}$$

Si la demanda d usa el *slot* s en el arco e, entonces $u_{des} = 1$ y las restricciones (2.57) garantizan que el número de *slots* usado por d en e es mayor o igual a $v(d)$. Por lo tanto, estas restricciones aseguran que cada demanda es satisfecha. Más aún, si la demanda d

tiene asignados los *slots* s_1 y s_2 (con $s_1 < s_2$) en el arco e, entonces todo *slot* en el rango $\{s_1, \ldots, s_2\}$ debe también ser asignado a d en el arco e, y esto es forzado por las restricciones (2.58). Es interesante notar que el modelo DSL-BF permite tener uno o más conjuntos de al menos $v(d)$ *slots* contiguos satisfaciendo una demanda d, a diferencia de este modelo, que fuerza que todos los *slots* utilizados por cada demanda sean contiguos. Esto no es un problema en ninguno de los dos casos, dado que las soluciones que utilizan más slots de los necesarios no pueden ser óptimas. Para no degradar el desempeño de la resolución preferimos no agregar más restricciones a los modelos.

Agregando límites a los *slots* (DSL-ASB)

En lugar de utilizar las variables enteras l_d y r_d para representar el intervalo de *slots* asignado a la demanda d, en este modelo recurrimos a las variables binarias asociadas con cada demanda y cada *slot*. Para $d \in D$ y $s \in S$, usamos la variable binaria l_{ds} (resp. r_{ds}) de forma tal que $l_{ds} = 1$ (resp. $r_{ds} = 1$) si s es el primer (resp. último) *slot* asignado a d. En este marco, la formulación DSL-ASB usa las restricciones (2.50), (2.51), (2.52), (2.53), (2.56) del modelo DSL-BF, las restricciones (2.57) de la variación DSL-ASCC, las restricciones de integralidad (2.37) de la formulación DR-BSA, y las siguientes restricciones:

$$\sum_{s \in S} s(r_{ds} - l_{ds}) = v(d) - 1 \qquad \forall d \in D \qquad (2.59)$$

$$\sum_{s \in S} l_{ds} = 1 \qquad \forall d \in D \qquad (2.60)$$

$$\sum_{s \in S} r_{ds} = 1 \qquad \forall d \in D \qquad (2.61)$$

$$\sum_{s'=1}^{s} l_{ds'} \geq u_{des} \qquad \forall d \in D, \forall e \in E, \forall s \in S \qquad (2.62)$$

$$\sum_{s'=s}^{\bar{s}} r_{ds'} \geq u_{des} \qquad \forall d \in D, \forall e \in E, \forall s \in S \qquad (2.63)$$

$$r_{ds} \in \{0, 1\} \qquad \forall d \in D, \forall s \in S. \qquad (2.64)$$

Si s_1, s_2 son el menor y el mayor *slot* asignados a la demanda d, entonces $l_{ds_1} = r_{ds_2} = 1$ mientras $l_{ds'} = 0$ para todos los *slots* s' diferentes a s_1, y $r_{ds'} = 0$ para todos los *slots* s' distintos de s_2. Entonces el lado izquierdo de las restricciones (2.59) se reduce a $s_2 - s_1$, implicando que el rango asignado a la demanda satisface su volumen. Las restricciones (2.60) y (2.61) garantizan que cada demanda tenga exactamente un primer y un último *slot*. Finalmente, las restricciones (2.62) y (2.63) relacionan las variables u con las variables l y r, asegurando que cualquier *slot* usado por una demanda pertenzca al rango especificado por las variables l y r correspondientes.

2.3.5. Formulaciones demanda-rango-enlace (DRL)

Formulación base (DRL-BF)

En lugar de explícitamente asignar *slots* a demandas y arcos, en esta formulación representamos el primer *slot* asignado a cada demanda en cada enlace. Para tal fin, usamos la variable binaria l_{des} para cada $d \in D$, $e \in E$, y $s \in \{1, \ldots, \bar{s} - v(d) + 1\}$, de modo tal

que $l_{des} = 1$ si y sólo si la demanda d usa el *slot* s en el enlace e, y s es el primer *slot* asignado a d. En este contexto, el siguiente modelo ILP es una formulación para el RSA.

$$\min \quad \sum_{d\in D}\sum_{e\in E}\sum_{s\in S} l_{des} \tag{2.65}$$

$$\text{s.t.} \quad \sum_{e\in\delta^-(j)} l_{des} - \sum_{e\in\delta^+(j)} l_{des} = 0 \qquad \forall d \in D,\ \forall s \in S,\ \forall j \in V\backslash\{s(d), t(d)\} \tag{2.66}$$

$$\sum_{e\in\delta^-(s(d))}\sum_{s\in S} l_{des} = 0 \qquad \forall d \in D \tag{2.67}$$

$$\sum_{e\in\delta^+(s(d))}\sum_{s\in S} l_{des} = 1 \qquad \forall d \in D \tag{2.68}$$

$$\sum_{d'\in D\backslash\{d\}}\sum_{s'=s}^{f} l_{d'es'} \le \bar{s}\,(1 - l_{des}) \qquad \forall d \in D,\ \forall e \in E,\ \forall s \in S,\ f = \text{mín}\{\bar{s},\ s + v(d) - 1\}. \tag{2.69}$$

$$l_{des} \in \{0, 1\} \qquad \forall d \in D,\ \forall e \in E,\ \forall s \in S. \tag{2.70}$$

En esta formulación tratamos de minimizar la función objetivo (2.65), buscando reducir el número total de arcos utilizados en el ruteo. Las restricciones (2.66), (2.67) y (2.68) aseguran la existencia de una ruta factible para cada demanda, eliminan ciclos que comiencen y terminen en el origen, y garantizan que cada demanda tenga asignado exactamente un primer *slot*, respectivamente. Las restricciones (2.69) establecen que si una demanda d tiene asignado a s como su primer *slot* en el enlace e, entonces ni s ni los siguientes $v(d) - 1$ *slots* pueden ser asignados a ninguna otra demanda d' en el mismo arco, evitando así el solapamiento de *slots*.

2.3.6. Discusión

En las secciones anteriores, hemos considerado diversas formulaciones para la misma variación del RSA. Una primera característica a tener en cuenta a la hora de compararlas es el tamaño del modelo, es decir, el número de variables y restricciones empleadas por cada formulación. La Tabla 2.1 muestra el número de variables y restricciones de las formulaciones propuestas en las Secciones 2.2 y 2.3. Agrupamos estos valores por cada una de las familias de modelos, ya que todos los modelos dentro de la misma familia presentan el mismo orden de magnitud para variables y restricciones.

Familia	Variables	Restricciones
Demanda-rango (DR)	$\Theta(\|D\|^2)$	$\Theta(\|D\|^2\|E\|)$
Demanda-*slot* (DS)	$\Theta(\|D\|\bar{s})$	$\Theta(\|D\|^2\|E\|\bar{s})$
Demanda-*slot*-enlace (DSL)	$\Theta(\|D\|\|E\|\bar{s})$	$\Theta(\|D\|\|E\|\bar{s})$
Demanda-rango-enlace (DRL)	$\Theta(\|D\|\|E\|\bar{s})$	$\Theta(\|D\|\|E\|\bar{s})$

Tab. 2.1: Número de variables y de restricciones de las formulaciones presentadas en la Sección 2.3.

El número de restricciones asociadas a la familia DS hace que sea poco práctico generar modelos de ILP para instancias grandes. Como ejemplo, con $\bar{s} = 100$, $|D| = 100$ y

$|E| = 10$, estaríamos tratando de manejar aproximadamente 10 millones de restricciones, mientras que con las familias DR, DSL y DRL, tenemos aproximadamente 100.000 restricciones. En la Sección 2.4 resolvemos instancias de diferentes tamaños para mostrar el comportamiento detallado de todas las formulaciones. Los tamaños de los conjuntos de variables no son tan diferentes entre las distintas familias como ocurre con los conjuntos de restricciones. Con la misma configuración anterior, *i.e.*, $\bar{s} = 100$, $|D| = 100$ y $|E| = 10$, las formulaciones DR y DS tienen aproximadamente 10.000 variables, mientras que DSL y DRL tienen aproximadamente 100.000 variables. Como veremos en las siguientes secciones, estos valores asociados a las familias influyen en la generación del modelo, en algunos casos con consecuencias críticas.

En general, no es posible realizar un análisis teórico sobre el desempeño práctico de un modelo de ILP, y esto motiva el análisis empírico de la Sección 2.4. Aunque el tamaño de una formulación no es un buen predictor del rendimiento de un *solver* de IP, si un modelo tiene una gran cantidad de variables o restricciones, puede ser difícil o imposible de generar para instancias de tamaño mediano o grande.

Sin embargo, es habitual considerar cuán ajustada es la relajación lineal en comparación con la cápsula convexa de las soluciones factibles. En este sentido, por ejemplo, las restricciones (2.23) son las conocidas restricciones de conservación de flujo y, aisladas, generan una cápsula convexa entera, que es una propiedad muy deseable en este tipo de modelos. Este hecho nos motivó a buscar formulaciones que incluyeran este tipo de restricciones. Aunque no generan cápsulas enteras, se sabe que algunas restricciones proporcionan relajaciones más fuertes que las versiones alternativas, como por ejemplo las restricciones (2.38) contra las restricciones (2.36) en los modelos DR-SC y DR-BSA, respectivamente, o las restricciones (2.46) versus las restricciones (2.41) en los modelos DS-ACC y DS-BF, respectivamente. En estos casos, los modelos con relajaciones más fuertes son más atractivos desde una perspectiva teórica.

2.4. Experimentos computacionales con las formulaciones de ILP

En esta sección reportamos los resultados de nuestra comparación experimental de los modelos presentados en las Secciones 2.2 y 2.3. Por su parte, la Sección 2.4.1 explora sistemáticamente el rendimiento de los modelos utilizando diferentes parámetros al variar el número de *slots* y el número de demandas. Cada modelo de ILP se probó con ocho topologías de red tomadas de la literatura utilizando diferentes combinaciones de demandas y conjuntos de *slots*. A su vez, la Sección 2.4.2 reporta los resultados computacionales sobre instancias reales.

Los modelos fueron implementados en OPL studio y resueltos con el *solver* IBM ILOG Cplex 12.6, en una computadora Intel de 2.10 GHz con núcleo i5.

Topologías de red

Cada topología de red T está dada por un grafo dirigido $G_T = (V_T, E_T)$ donde V_T y E_T son los conjuntos de nodos y de enlaces, respectivamente. La información de cada grafo G_T utilizado para estos experimentos puede observarse en la Tabla 2.2. La tercera columna de dicha tabla denota el *coeficiente de dispersión* –también llamado *arcs-density*–

del grafo G_T, el cual está dado por

$$\Delta_T = \frac{|E_T|}{|E(K_{|V_T|})|},$$

donde $|E(K_{|V_T|})| = |V_T|(|V_T| - 1)$ (*i.e.*, la cantidad de arcos del digrafo completo de $|V_T|$ vértices). Asimismo, la cuarta y quinta columnas de dicha tabla presentan, respectivamente, el mínimo y el máximo grado de los vértices de cada topología, calculados como la cantidad de arcos que entran o que salen de cada uno de ellos.

Topología T	$\|V_T\|$	$\|E_T\|$	Δ_T	$\text{mín}_{v \in V_T}\, \delta(v)$	$\text{máx}_{v \in V_T}\, \delta(v)$
11n-52m-COST239	11	52	0.47	8	12
14n-42m-NSF	14	42	0.23	4	8
14n-46m-DT	14	46	0.25	4	12
19n-76m-EON19	19	76	0.22	4	12
21n-64m-DT	21	64	0.15	4	6
21n-70m-Spain	21	70	0.17	4	8
22n-70m-BT	22	70	0.15	6	8
21n-78m-UKNet	21	78	0.19	4	14

Tab. 2.2: Cantidad de nodos y de arcos, coeficiente de dispersión y grado máximo y mínimo de los nodos de cada topología utilizada.

2.4.1. Exploración del espacio de parámetros

Con el fin de comparar el rendimiento de las distintas formulaciones de ILP, las evaluamos utilizando 96 instancias generadas a partir de las ocho topologías de red mencionadas anteriormente. Para cada una de estas topologías utilizamos los valores $\{10,\ 20,\ 40\}$ y $\{2|D|,\ 2|D| + 10,\ 2|D| + 20,\ 2|D| + 30\}$ para $|D|$ y $\bar{s}$, respectivamente. Cada demanda $d \in D$ tiene asociado un volumen, generado con distribución uniforme dentro de S, y un par de nodos origen y destino, generados también con distribución uniforme en el conjunto V. Una instancia, entonces, se define mediante una topología de red específica, un conjunto de demandas a satisfacer y una cantidad de *slots* disponibles, la cual depende del número de demandas.

En la Tabla 2.3 resumimos los resultados de este primer conjunto de experimentos. La primera columna corresponde a la formulación ILP utilizada. Debido a la gran cantidad de pruebas que hemos realizado en esta sección, el *tiempo límite* fue fijado en 600 segundos, *i.e.*, 10 minutos, de forma tal que si la solución óptima no es alcanzada dentro de ese lapso, la ejecución se detiene. Si el tamaño del modelo generado es tal que OPL studio no es capaz de manejarlo, se reporta una excepción de *Memoria* y se pasa al siguiente experimento. Asimismo, dado que el volumen, origen y destino de las demandas son generados de forma aleatoria, la instancia creada puede ser *no factible*. La segunda, tercera y cuarta columna de la Tabla 2.3 muestran el porcentaje de instancias resueltas en forma óptima, de instancias no resueltas dentro del plazo establecido, y de instancias para las cuales se obtuvo una excepción por falta de *Memoria*, respectivamente, para cada formulación. Las instancias *no factible* no son tenidas en cuenta al calcular estos porcentajes. La última columna corresponde al mayor tamaño del conjunto D para el cual se pudo obtener el óptimo.

Con el objeto de mostrar un análisis más exhaustivo, elegimos las formulaciones con mejor desempeño en cada una de las familias y presentamos un comportamiento detallado

Formulación	% Resueltas	% Tiempo límite	% Memoria	Mayor $\|D\|$ resuelto
DR-BF	86	14	0	40
DR-OSB	85	15	0	40
DR-AOV	90	10	0	40
DR-BSA	37	4	59	20
DR-SC	22	0	78	10
DS-BF	34	18	48	10
DS-ACC	38	3	59	20
DS-SCC	46	2	52	20
DSL-BF	46	52	2	20
DSL-ASCC	17	17	64	10
DSL-ASB	21	32	47	10
DRL-BF	35	9	56	10

Tab. 2.3: Compendio de resultados para las formulaciones de ILP.

de las mismas. No tomamos un representante de la familia DRL por sus similitudes en cuanto a las características a evaluar con la familia DSL. Elegimos las formulaciones DR-AOV, DS-SCC y DSL-BF, y las Figuras 2.3, 2.4 y 2.5 comparan su promedio de variables, restricciones y tiempos de resolución, respectivamente. Los resultados se muestran de acuerdo a los tamaños de las instancias (combinación de $|D|$ y $\bar{s}$), y el promedio se realiza sobre todas las instancias, es decir, la suma se calcula sobre todas las instancias de cada tamaño dividida por el número total de instancias de cada tamaño.

En estas figuras, la barra asociada a la formulación DS-SCC ya no aparece a partir de la instancia $(40, 90)$, debido a que las formulaciones no se pudieron generar a causa de su tamaño. Lo mismo ocurre con la formulación DSL-BF a partir de la instancia $(80, 160)$.

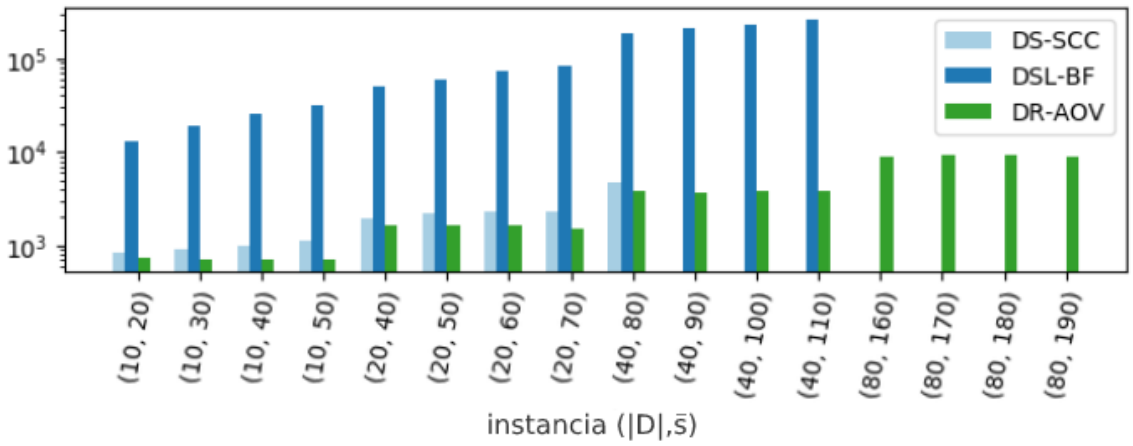


Fig. 2.3: Número promedio de variables para los modelos DS-SCC, DSL-BF y DR-AOV.

La Figura 2.3 muestra que el modelo DSL-BF tiene una gran cantidad de variables. Esto se debe a la forma en que se representa la asignación en la familia a la que pertenece dicha formulación. Como en el resto de modelos de la familia DSL, para la asignación de demandas a enlaces y *slots*, se utiliza un conjunto de variables que tiene un tamaño

igual a $|D||E|\bar{s}$. Las otras familias de modelos representan por un lado la asignación de demandas a enlaces y por otro la asignación de demandas a *slots*, con lo cual el número de variables es considerablemente menor. En el caso del modelo DS-SCC, encontramos un número teórico de variables igual a $|D|(|E| + \bar{s})$, que es menor que el monto del modelo DR-AOV, siendo este último $|D||E| + 2|D| + |D|^2$.

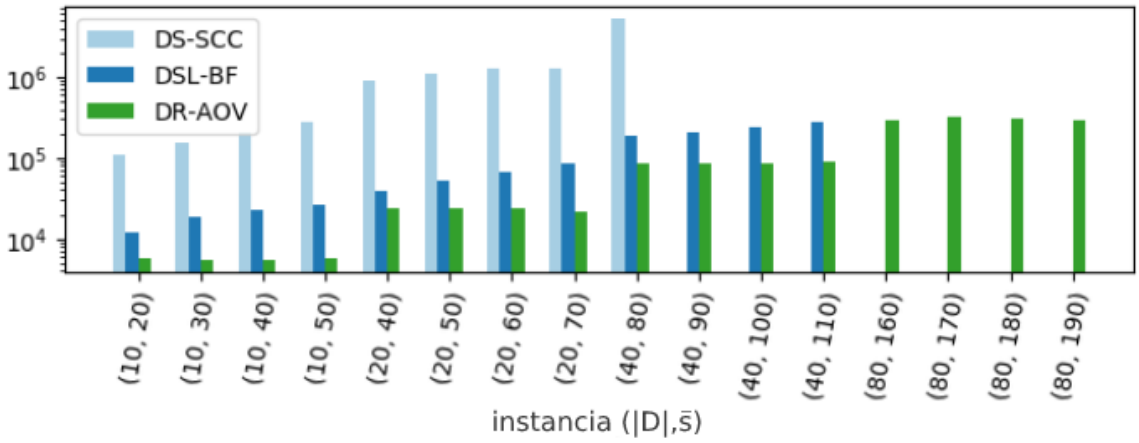


Fig. 2.4: Número de restricciones promedio de los modelos DS-SCC, DSL-BF y DR-AOV.

La Figura 2.4 muestra el gran número de restricciones que contiene el modelo DS-SCC. Esto se debe a la familia de restricciones agregada para garantizar la propiedad de no solapamiento, la cual solicita que a cada par de demandas no se le asigne el mismo *slot* en ningún enlace (dando un total de $|D|^2|E|\bar{s}$ restricciones). Esta gran cantidad de restricciones, que es un orden de magnitud mayor que la cantidad de variables del modelo DSL-BF, provoca que a partir de un cierto número de demandas y *slots* (tamaño del conjunto D y valor de $\bar{s}$), los modelos asociados ya no puedan ser generados por OPL studio, produciendo excepciones por falta de memoria.

En ambos aspectos, el modelo DR-AOV fue el mejor en esta primera etapa de evaluación. En cuanto al número de variables, se aprecia una diferencia que puede alcanzar hasta dos órdenes de magnitud comparado con el modelo DSL-BF y una pequeña diferencia comparado con el modelo DS-SCC. Cabe señalar que, como estamos usando variables enteras para la asignación de demandas con *slots*, el número de variables no aumenta a medida que crece $\bar{s}$. En términos del número de restricciones, es posible ver una diferencia que puede alcanzar hasta dos órdenes de magnitud en comparación con el modelo DS-SCC y de un orden de magnitud en comparación con el modelo DSL-BF. Al igual que con el número de variables, el número de restricciones no se altera modificando $\bar{s}$, esto se debe a que ninguna restricción depende de este valor.

Finalmente, como podemos ver en la Figura 2.5, el modelo DR-AOV es el único que podría resolver todas las instancias y el que tiene el mejor tiempo de ejecución. El modelo DS-SCC fue el modelo con el menor número de instancias resueltas de manera óptima, lo cual se debe, como ya se ha mencionado, a que en muchos casos ni siquiera se pueden generar las formulaciones a causa de su tamaño. Por otro lado, el modelo DSL-BF tiene un número menor de instancias resueltas de mandera sub-óptima, lo cual se debe a las excepciones por límite de tiempo, que probablemente puedan reducirse sustancialmente

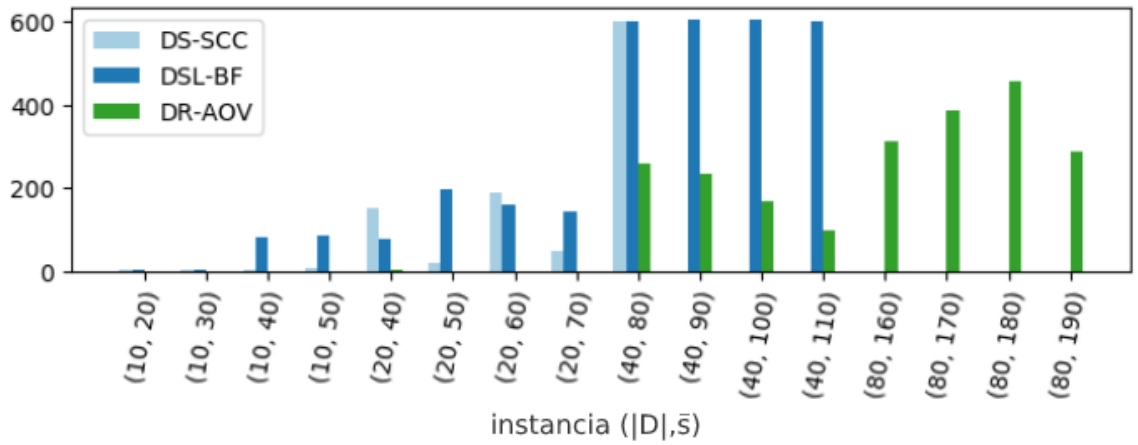


Fig. 2.5: Tiempo de resolución promedio para los modelos DS-SCC, DSL-BF y DR-AOV.

estableciendo un tiempo de resolución máximo más alto.

2.4.2. Comparación de rendimientos en escenarios reales

A lo largo de esta sección mostraremos los resultados de los experimentos computacionales que hemos realizado con todas las formulaciones de las Secciones 2.2 y 2.3 en escenarios reales. Utilizamos *22n-70m-BT* (*British Telecom*) y *14n-42m-NSF* como las topologías para las instancias, buscando cubrir diversos coeficientes de dispersión. Consideramos *slots* de $12.5GHz$ y dos tamaños de espectros ópticos: $400GHz$ y $4000GHz$, los cuales pueden ser generados utilizando un diodo laser y un LED como origen óptico, respectivamente. Por lo tanto para esta segunda etapa de experimentos computacionales, el valor $\bar{s}$ puede ser 32 ó 320. Para cada elemento en el conjunto de demandas, el ancho de banda requerido es 10, 40, ó $100Gbps$. Esto es, para cada demanda $d \in D$, el volumen asociado $v(d)$ fue generado con distribución uniforme en $\{1, 2, 4\}$, y el origen y destino asociado también fue generado con distribución uniforme dentro del conjunto V. El tamaño de dicho conjunto D depende de la topología y de $\bar{s}$: para la red NSF con $\bar{s} = 32$, utilizamos 30, 50 y 80 como valores para $|D|$ mientras que para un $\bar{s} = 320$, usamos los valores 100, 150 y 180; para la topología *British* los valores usados para $|D|$ son 30, 50 y 70 cuando $\bar{s}$ es 32, y 30, 50 y 70 cuando el $\bar{s}$ es 320. Utilizamos distintos valores para $|D|$ dependiendo de la topología y del $\bar{s}$ con el objeto de mantener la dificultad y el tamaño de las instancias bajo control. Por un lado, las instancias generadas con la red *British*, la capacidad $\bar{s} = 32$ y la cantidad de demandas $|D| = 80$ son imposibles de resolver con cualquiera de los modelos considerados, incluso fijando el tiempo límite en 1 hora. Por otro lado, la red NSF tiene 14 nodos, por lo que no podemos generar más de 182 demandas sin repetir pares origen-destino.

Las Tablas 2.4 y 2.5 muestran el desempeño de las formulaciones en los experimentos que utilizan las topologías NSF y *British Telecom*, respectivamente. Para esta segunda etapa de evaluación fijamos el tiempo límite en 1800 segundos, *i.e.*, 30 minutos, de manera tal que si una ejecución excede dicha cota, utilizamos la etiqueta *Tiempo*. Asimismo, si no es posible para OPL studio generar la formulación a causa del tamaño de la instancia,

hacemos uso del caracter "-". Nuevamente, las formulaciones BF, OSB, y AOV de la familia DR junto con DSL-BF presentan los mejores resultados, esta vez teniendo en cuenta los tiempos de resolución y los tamaños de las intancias que pueden ser manejadas sin obtener excepciones por memoria. Para todas las formulaciones de la familia DS y la mayoría de la DSL, OPL studio no puede generar los modelos a resolver. Como podemos ver en la Tabla 2.1, esto se debe a la gran cantidad de restricciones que tiene esta formulación.

	$\bar{s} = 32$			$\bar{s} = 320$		
Formulación	$\|D\| = 30$	$\|D\| = 50$	$\|D\| = 80$	$\|D\| = 100$	$\|D\| = 150$	$\|D\| = 180$
DR-BF	1	5	18	21	100	109
DR-OSB	2	6	18	30	76	296
DR-AOV	1	3	6	13	42	365
DR-BSA	33	129	594	-	-	-
DR-SC	51	179	-	-	-	-
DS-BF	17	50	-	-	-	-
DS-ACC	20	65	-	-	-	-
DS-SCC	14	49	-	-	-	-
DSL-BF	3	4	22	243	Tiempo	Tiempo
DSL-ASCC	18	29	161	-	-	-
DSL-ASB	11	21	Tiempo	-	-	-
DRL-BF	13	26	104	Tiempo	Tiempo	Tiempo
NLS	12	46	162	-	-	-
NL-CA	24	64	Tiempo	Tiempo	Tiempo	Tiempo

Tab. 2.4: Tiempos computacionales –en segundos– de las formulaciones de ILP en los experimentos reales con NFS como la topología de red (14 nodos y 21 enlaces bidireccionales).

Comparadas con las formulaciones tomadas de la literatura y sobre este conjunto de experimentos, la mayoría de nuestras formulaciones presentan mejores tiempos de solución y fueron capaces de manejar instancias más grandes. Para el modelo NLS, con la topología NSF y $\bar{s} = 32$, los resultados de los experimentos son aproximadamente los mismos que los obtenidos por nuestras formulaciones. Sin embargo, cuando el valor de $\bar{s}$ crece hasta 320 ya no le es posible a OPL studio siquiera generar los modelos para cualquier tamaño del conjunto D, debido a que las restricciones de no-solapamiento tienen un gran número de elementos en este caso. A modo de ejemplo, y como fue mencionado con anterioridad, con $\bar{s} = 320$, $|D| = 100$ y $|E| = 30$, estamos tratando de manejar un conjunto de aproximadamente 100 millones de restricciones. Para el modelo NL-CA, únicamente hemos podido obtener la solución óptima para dos instancias, para todos los demás casos se llegó al límite de tiempo.

2.5. Conclusiones

En este capítulo hemos presentado varios modelos de programación entera que resuelven la versión estática del RSA. Hemos probado con diferentes familias de variables y restricciones, obteniendo diversos resultados.

Como las pruebas computacionales se realizaron utilizando OPL studio y el *solver* de programación entera como una caja negra, los experimentos de la Sección 2.4 brindan para cada modelo una idea aproximada del tamaño de la instancia que es capaz de

Formulación	$\bar{s} = 32$ $\lvert D\rvert = 30$	$\lvert D\rvert = 50$	$\lvert D\rvert = 70$	$\bar{s} = 320$ $\lvert D\rvert = 100$	$\lvert D\rvert = 150$	$\lvert D\rvert = 200$
DR-BF	2	11	261	51	285	923
DR-OSB	2	9	19	81	349	Tiempo
DR-AOV	2	5	9	23	60	122
DR-BSA	64	1200	Tiempo	-	-	-
DR-SC	201	Tiempo	Tiempo	-	-	-
DS-BF	51	335	Tiempo	-	-	-
DS-ACC	37	1148	Tiempo	-	-	-
DS-SCC	65	362	Tiempo	-	-	-
DSL-BF	10	27	151	Tiempo	Tiempo	Tiempo
DSL-ASCC	48	72	Tiempo	-	-	-
DSL-ASB	59	240	Tiempo	-	-	-
DRL-BF	27	50	656	Tiempo	Tiempo	-
NLS	26	285	Tiempo	-	-	-
NL-CA	Tiempo	Tiempo	Tiempo	Tiempo	Tiempo	Tiempo

Tab. 2.5: Tiempos computacionales –en segundos— de las formulaciones de ILP en experimentos reales con la topología *22n-70m-BT* (20 nodos y 35 enlaces bidireccionales).

resolver sin el uso de técnicas más sofisticadas de programación lineal entera. Nuestros experimentos computacionales, tanto sobre instancias reales como sobre instancias generadas aleatoriamente, sugieren que los modelos DR-BF, DR-OSB y DR-AOV superan a las otras formulaciones resolviendo casi el 90 % de las instancias propuestas sin problemas de memoria y obteniendo valores óptimos en las instancias más grandes, incluso un orden de magnitud más rápido que los demás. Esto puede deberse a la forma en que se representan las asignaciones de intervalos y enlaces en estas formulaciones, a saber, la asignación de intervalos a las demandas está representada implícitamente.

Del resto de modelos, la formulación DSL-BF es la de mejores resultados. Aunque no puede resolver las instancias más grandes, este modelo ha logrado resolver más instancias que las otras formulaciones y en tiempos razonables, incluso para algunas instancias requirió menos tiempo que la formulación DR-BF.

Dado que esta tesis forma parte de un proyecto que busca estudiar los modelos más prometedores para el RSA, en los siguientes capítulos nos concentraremos en particular en esta segunda formulación. El modelo DSL-BF, a pesar de tener un gran número de variables, arrojó resultados bastante alentadores y parece muy interesante desde un punto de vista teórico. En particular, vamos a intentar mejorar su rendimiento aplicando técnicas habituales de programación lineal entera. El resto de las formulaciones están siendo a su vez estudiadas como parte del mencionado proyecto utilizando diversos enfoques. En particular para la formulación DR-BF se han iniciado estudios poliedrales que se encuentran en una etapa preliminar.

3

Desigualdades e igualdades válidas y cortes optimales para el modelo DSL-BF

En capítulos anteriores hemos presentado y estudiado varios modelos de programación entera para el problema de ruteo y asignación de espectro. Ahora, intentaremos mejorar los resultados obtenidos con la formulación DSL-BF aplicando técnicas comúnmente utilizadas en programación lineal entera. En este capítulo en particular pretendemos estudiar la formulación y, explotando algunas de sus propiedades, presentar una serie de desigualdades válidas que podrían usarse como planos de corte en un algoritmo *branch-and-cut*.

A continuación presentamos nuevamente el modelo que estudiamos en este y los siguientes capítulos. La familia de variables binarias $u \in \{0,1\}^{|D||E|\bar{s}}$ se define para cada demanda $d \in D$, cada arco $e \in E$, y cada *slot* $s \in S$, de modo tal que $u_{des} = 1$ si y sólo si la demanda d usa el *slot* s sobre el arco e. Por razones técnicas, también consideramos la variable ficticia $u_{de,\bar{s}+1}$ que siempre toma el valor 0, por cada $d \in D$ y cada $e \in E$. En este escenario, la formulación DSL-BF para el RSA está dada por el siguiente programa entero.

$$\min \quad \sum_{d\in D}\sum_{e\in E}\sum_{s\in S} u_{des}/v(d) \tag{3.1}$$

$$\text{s.t.} \quad \sum_{e\in\delta^-(j)} u_{des} - \sum_{e\in\delta^+(j)} u_{des} = 0 \qquad \forall d \in D,\, \forall j \in V\backslash\{s(d),t(d)\},\, \forall s \in S \tag{3.2}$$

$$\sum_{e\in\delta^+(s(d))}\sum_{s\in S} u_{des} \geq v(d) \qquad \forall d \in D \tag{3.3}$$

$$\sum_{e\in\delta^-(s(d))}\sum_{s\in S} u_{des} = 0 \qquad \forall d \in D \tag{3.4}$$

$$\sum_{d\in D} u_{des} \leq 1 \qquad \forall e \in E,\, \forall s \in S \tag{3.5}$$

$$u_{de,\bar{s}+1} = 0 \qquad \forall d \in D,\, \forall e \in E \tag{3.6}$$

$$v(d)(u_{des} - u_{de,s+1}) \leq \sum_{s'=f}^{s} u_{des'} \qquad \begin{array}{c} \forall d \in D,\ \forall e \in E,\ \forall s \in S, \\ f = \text{máx}\{1, s - v(d) + 1\} \end{array} \tag{3.7}$$

$$u_{des} \in \{0,1\} \qquad \forall d \in D,\ \forall e \in E,\ \forall s \in S \tag{3.8}$$

Definimos como $RSA(G, D, \bar{s})$ a la cápsula convexa de soluciones factibles para la formulación DSL-BF (3.2)-(3.8). Una *desigualdad válida* (resp. *igualdad válida*) es una desigualdad (resp. igualdad) lineal en u satisfecha por todos los puntos en $RSA(G, D, \bar{s})$. Un *corte optimal* (resp. *igualdad optimal*) es una desigualdad (resp. igualdad) inválida que es cumplida por al menos una solución óptima. Decimos que una desigualdad remueve una solución, cuando dicha solución no satisface la desigualdad. En particular, ninguno de los cortes optimales propuestos en el presente capítulo remueve soluciones óptimas.

El politopo $RSA(G, D, \bar{s})$ parece ser muy dificil de estudiar, dado que incluso caracterizar su dimensión no es algo sencillo (como sugieren las igualdades válidas de las siguientes secciones). A causa de esto, en este capítulo exploramos familias de desigualdades válidas sin demostrar resultados de facetitud. En cambio, nos conformamos con un objetivo más modesto, al mostrar que –junto con las restricciones del modelo– las desigualdades de estas familias no se implican entre ellas.

El resto de este capítulo está organizado de la siguiente manera. La Sección 3.1 presenta tres teoremas provenientes de consideraciones de simetría, y las Secciones 3.2, 3.3 y 3.4 introducen algunas propiedades y diversas familias de desigualdades e igualdades válidas, así como una gran cantidad de cortes optimales, basados en consideraciones de flujo, contigüidad y no solapamiento, respectivamente.

3.1. Propiedades provenientes de consideraciones de simetría

La formulación DSL-BF presenta varias simetrías, por ejemplo, con respecto a la dirección del camino asignado a cada demanda, o el orden de asignación de canales en los arcos. Estas consideraciones conducen a los siguientes resultados técnicos.

Definición 3.1.1. *Definimos como* $I_s : \mathbb{R}^{|D||E|\bar{s}} \to \mathbb{R}^{|D||E|\bar{s}}$ *a la transformación tal que si* $\bar{u} = I_s(u')$ *entonces* $\bar{u}_{des} = u'_{de,\bar{s}-s+1}$ *para todo arco* $e \in E$*, demanda* $d \in D$ *y* slot $s \in S$*.*

Es decir, dado un vector u, la función I_s devuelve otro vector que es igual a u pero invirtiendo los valores de los *slots* en cada arco para cada demanda. En la solución u^* toda demanda d tiene asignado al menos un *lightpath* compuesto por un canal de al menos $v(d)$ *slots* a lo largo de un camino que conecta $s(d)$ con $t(d)$ sin solapamiento entre demandas. El vector $\bar{u} = I_s(u^*)$ mantiene asignados los mismos arcos para cada demanda, y la misma cantidad de *slots* contiguos, sólo que en orden inverso, por tanto $\bar{u}$ es también una solución para $RSA(G, D, \bar{s})$. Este argumento prueba el siguiente resultado.

Lema 3.1.1. *Si* $u^* \in \mathbb{R}^{|D||E|\bar{s}}$ *es una solución factible para una instancia del* $RSA(G, D, \bar{s})$*, entonces* $I_s(u^*)$ *también es solución factible para dicha instancia.*

Los vectores $u^1, \ldots, u^k \in \mathbb{R}^d$ se dicen *afinmente independientes* si la única solución para el sistema $\sum_{i=1}^{k} \lambda_i u^i = 0$ y $\sum_{i=1}^{k} \lambda_i = 0$ tomando $\lambda \in \mathbb{R}^k$ es $\lambda_i = 0$ para todo $i = 1, \ldots, k$. Por lo tanto, dado que la función I_s invierte el orden de algunos elementos del vector recibido, podemos enunciar el siguiente resultado.

Lema 3.1.2. *Si los vectores* $u^1, \ldots, u^k$*, con* $k \in \mathbb{N}$*, son afinmente independientes, entonces sus transformaciones* $I_s(u^1), \ldots, I_s(u^k)$ *también son afinmente independientes.*

Demostración. El mencionado sistema compuesto por las ecuaciones $\sum_{i=1}^{k} \lambda_i u^i = 0$ y $\sum_{i=1}^{k} \lambda_i = 0$ se puede representar mediante la igualdad $\lambda_1 + \cdots + \lambda_k = 0$ junto con las $\bar{s}$ ecuaciones

$$\begin{cases} \lambda_1 u^i_{de1} + \cdots + \lambda_k u^k_{de1} = 0 \\ \vdots \\ \lambda_1 u^i_{de\bar{s}} + \cdots + \lambda_k u^k_{de\bar{s}} = 0, \end{cases}$$

correspondientes a cada $d \in D$ y $e \in E$. El sistema para los vectores $I_s(u^1), \ldots, I_s(u^k)$ es idéntico pero invirtiendo el orden de aparición de estas igualdades. Por lo tanto las soluciones de ambos sistemas son las mismas. □

Teorema 3.1.1. *Si $a \cdot u \leq b$ es una desigualdad válida (resp., un corte optimal) para $RSA(G, D, \bar{s})$, entonces la desigualdad denominada* simetría-por-slots*,*

$$\sum_{d \in D} \sum_{e \in E} \sum_{s \in S} a_{des}\ u_{de,\bar{s}-s+1} \leq b \tag{3.9}$$

es también válida (resp., un corte optimal) para $RSA(G, D, \bar{s})$. Más aún, si $a \cdot u \leq b$ induce faceta de $RSA(G, D, \bar{s})$, entonces (3.9) también induce una faceta de este politopo.

Demostración. Si u' es una solución factible para $RSA(G, D, \bar{s})$, por el Lema 3.1.1, su transformación $I_s(u')$ también es solución, por lo que si $a \cdot u \leq b$ es una desigualdad válida (resp. un corte optimal), como (3.9) es exactamente $a \cdot I_s(u) \leq b$, entonces (3.9) es también una desigualdad válida (resp. un corte optimal). Respecto a la proposición de facetitud, recordemos que una desigualdad válida induce una faceta de un politopo P si existen k puntos afinmente independientes que satisfacen la desigualdad por igualdad, con $\dim(P) = k$, y por lo tanto $\dim(\mathcal{F}) = k - 1$, siendo $\mathcal{F}$ el conjunto de puntos del politopo que satisfacen la desigualdad por igualdad. Asumiendo que la dimensión de $RSA(G, D, \bar{s})$ es k, si $a \cdot u \leq b$ define faceta, deben existir k vectores afinmente independientes que cumplen $a \cdot u^i = b$, con $i = 1, \ldots, k$, entonces como la transformación I_s mantiene la afinidad del conjunto de vectores recibido, *i.e.*, por el Lema 3.1.2, sus transformaciones $I_s(u^1), \ldots, I_s(u^k)$ también son afinmente independientes y cumplen $a \cdot I_s(u^i) = b$ con $i = 1, \ldots, k$, y por tanto la desigualdad (3.9) también induce una faceta del politopo. □

Cualquier solución al problema se puede espejar para obtener otra solución. Por esta razón, cuando una desigualdad tiene en cuenta el orden de los *slots*, es posible generar una nueva simétrica utilizando el orden opuesto. Es decir, cuando establecemos una familia de desigualdades que incluye de alguna manera el menor *slot* utilizable, podemos obtener una desigualdad simétrica considerando el mayor *slot* disponible. A modo de ejemplo, si para algún arco $e \in E$, una demanda $d \in D$ usa un *slot* $s < v(d)$, entonces el *slot* $s + 1$ también debe ser usado por d. Esto se puede forzar con las desigualdades

$$u_{de,s+1} \geq u_{des} \qquad \forall e \in E,\ \forall d \in D,\ \forall s \in \{1, \ldots, v(d) - 1\}. \tag{3.10}$$

Dado que estas desigualdades consideran el conjunto de *slots* de menor índice, es decir, de 1 a $v(d)$ para cada demanda d, es posible generar una familia simétrica considerando los *slots* más altos, a saber,

$$u_{de,s-1} \geq u_{des} \qquad \forall e \in E,\ \forall d \in D,\ \forall s \in \{\bar{s} - v(d) + 2, \ldots, \bar{s}\}. \tag{3.11}$$

Otra simetría presentada por el problema y el modelo bajo estudio surge del hecho de estar utilizando caminos. Como para cada demanda se debe conservar el flujo en cada nodo intermedio de su ruta, es decir, los *slots* utilizados por cada demanda deben ser los mismos tanto en el arco entrante como en el saliente de cada nodo distinto de su origen y destino, siempre que tengamos una desigualdad relativa a los conjuntos de arcos entrantes y salientes de dicho nodo, es posible reflejarla invirtiendo dichos conjuntos de arcos, generando una nueva desigualdad. Este hecho motiva el siguiente teorema.

Teorema 3.1.2. *Sea $j \in V$ tal que $j \notin \{s(d), t(d)\}$ para cada $d \in D$. Sea $a \cdot u \leq b$ una desigualdad válida (resp., un corte optimal) para $RSA(G, D, \bar{s})$ de modo tal que para cada $d \in D$ y cada $s \in S$ existen $\alpha_{ds} \in \mathbb{R}$ y $\beta_{ds} \in \mathbb{R}$ con $a_{des} = \alpha_{ds}$ para cada $e \in \delta^+(j)$ y $a_{des} = \beta_{ds}$ para cada $e \in \delta^-(j)$. Luego, la desigualdad*

$$\sum_{s\in S}\sum_{d\in D}\Big(\sum_{e\in\delta^-(j)}\alpha_{ds}u_{des} + \sum_{e\in\delta^+(j)}\beta_{ds}u_{des} + \sum_{e\in E\setminus\delta(j)}a_{des}u_{des}\Big) \leq b \tag{3.12}$$

es también válida (resp., un corte optimal) para $RSA(G, D, \bar{s})$.

Demostración. Por las condiciones pedidas respecto de a_{des} para $e \in \delta^-(j)$ y $e \in \delta^+(j)$, la desigualdad válida (resp. corte optimal) $a \cdot u \leq b$ puede escribirse como

$$\sum_{s\in S}\sum_{d\in D}\Big(\beta_{ds}\sum_{e\in\delta^-(j)}u_{des} + \alpha_{ds}\sum_{e\in\delta^+(j)}u_{des} + \sum_{e\in E\setminus\delta(j)}a_{des}u_{des}\Big) \leq b, \tag{3.13}$$

y por las restricciones de flujo (3.2), todo punto $u^* \in RSA(G, D, \bar{s})$ cumple que

$$\sum_{e\in\delta^-(j)}u^*_{des} = \sum_{e\in\delta^+(j)}u^*_{des},$$

para todo $s \in S$, $d \in D$, sobre los vértices $j \in V \setminus \{s(d), t(d)\}$, por lo que

$$\beta_{ds}\sum_{e\in\delta^-(j)}u^*_{des} + \alpha_{ds}\sum_{e\in\delta^+(j)}u^*_{des} = \alpha_{ds}\sum_{e\in\delta^-(j)}u^*_{des} + \beta_{ds}\sum_{e\in\delta^+(j)}u^*_{des},$$

con lo cual (3.13) puede transformarse en (3.12). Por lo tanto, esta última desigualdad es también válida (resp. un corte optimal). □

Nuevamente a causa a la conservación del flujo, es posible encontrar una simetría entre los extremos de cada camino. De esta forma, toda afirmación hecha para una solución óptima sobre los arcos incidentes al origen de un camino es aplicable al conjunto de arcos incidentes al destino, y viceversa; lo cual motiva el siguiente resultado.

Teorema 3.1.3. *Fijemos una demanda $d \in D$ y definamos $\delta_d = \delta(t(d)) \cup \delta(s(d))$. Sea $a \cdot u \leq b$ un corte optimal para $RSA(G, D, \bar{s})$ tal que para cada* slot *$s \in S$ existen $\alpha_{ds} \in \mathbb{R}$ y $\beta_{ds} \in \mathbb{R}$ con $a_{des} = \alpha_{ds}$ para cada $e \in \delta^+(s(d))$, $a_{des} = \beta_{ds}$ para cada $e \in \delta^-(t(d))$, $a_{des} = 0$ para cada $e \in \delta^-(s(d)) \cup \delta^+(t(d))$, y $a_{d'es} = 0$ para cada $d' \neq d$ y $e \in \delta_d$. Entonces, la desigualdad*

$$\sum_{s\in S}\Big(\sum_{e\in\delta^-(t(d))}\alpha_{ds}u_{des} + \sum_{e\in\delta^+(s(d))}\beta_{ds}u_{des} + \sum_{d'\in D}\sum_{e\in E\setminus\delta_d}a_{d'es}u_{d'es}\Big) \leq b \tag{3.14}$$

es también un corte optimal para $RSA(G, D, \bar{s})$.

Demostración. Siguiendo el razonamiento de la demostración del Teorema 3.1.2, alcanza con probar que en toda solución óptima u^*, para toda demanda d y *slot* s se cumple la siguiente igualdad:

$$\sum_{e \in \delta^-(t(d))} u^*_{des} = \sum_{e \in \delta^+(s(d))} u^*_{des}. \tag{3.15}$$

Por las restricciones de flujo (3.2) podemos verificar que si $u^*_{des} = 1$ con $e \in \delta^+(s(d))$, entonces debe haber un camino $P \subseteq E$ con $e \in P$ tal que $u^*_{de's} = 1$ para todo $e' \in P$. A causa de las restricciones (3.3), dicho camino no puede contener el nodo $s(d)$, por lo que debe terminar indefectiblemente en $t(d)$. Asimismo, dado que ninguna solución óptima puede contener un ciclo (en cuyo caso no sería óptima) tenemos que $\sum_{e \in \delta^+(t(d))} u^*_{des} = 0$. Por lo tanto, si todos los caminos sobre el *slot* s que comienzan en $s(d)$ terminan en $t(d)$ y no puede existir ningún ciclo comenzando y terminando en $t(d)$, la igualdad (3.15) vale, y por lo tanto es posible transformar $a \cdot u \leq b$ en (3.14). □

3.2. Desigualdades y cortes optimales basados en el flujo

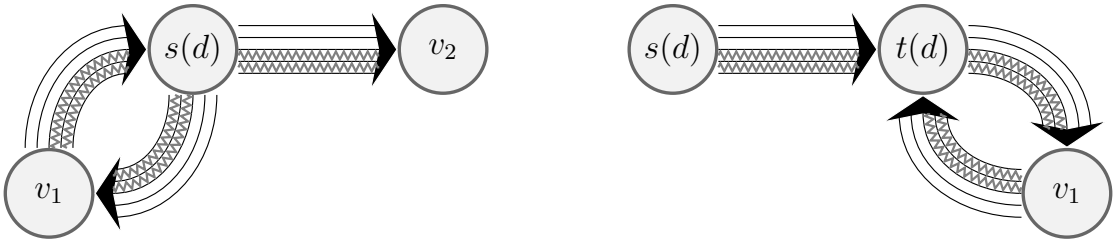
Un análisis sobre el modelo nos permite enumerar diferentes características no deseadas que puede tener una solución factible no óptima. Dichas anomalías son eliminadas por la función objetivo al minimizar los arcos y *slots* utilizados, por lo que no van a pertenecer a ninguna solución óptima. En estas secciones estudiamos uno a uno estos casos y proponemos diversas desigualdades válidas y cortes optimales con el fin de dejarlos fuera del conjunto solución. En la mayoría de las familias presentadas forzamos las demandas a ser satisfechas por igualdad, *i.e.*, la cantidad de *slots* contiguos que utiliza una demanda d es exactamente su volumen $v(d)$.

A pesar de que ayudar agregando cortes optimales ad-hoc y relegando a la función objetivo el filtrado de estas soluciones subóptimas sea una técnica ampliamente utilizada, también es posible agregar muchos de estos cortes directamente como restricciones iniciales al modelo. De esta forma, el modelo no admitiría soluciones con los elementos espurios mencionados en esta sección, aunque al costo de incluir un mayor número de restricciones y potencialmente una estructura muy complicada. En caso de agregar restricciones al modelo para evitar elementos espurios en las soluciones, entonces todos los cortes optimales del presente capítulo pasan a ser desigualdades (resp. igualdades) válidas.

Para ésta y las siguientes secciones, excepto cuando se aclare, siempre que asumimos una instancia para el RSA, llamamos $G = (V, E)$ al digrafo con un conjunto de vértices V y un conjunto de arcos E, denominamos D al conjunto de demandas y $\bar{s}$ a la capacidad de los arcos. Asimismo, cada vez que tengamos un vector particular $u' \in \{0,1\}^{|D||E|\bar{s}}$, escribimos como $u'_{de*} \in \{0,1\}^{\bar{s}}$ al sub-vector u' que contiene las entradas de u' asociadas con la demanda d y el arco e. Análogamente escribimos como $u'_{d**} \in \{0,1\}^{|E|\bar{s}}$ al sub-vector u' que contiene las entradas asociadas con la demanda d. Denotamos como $|C|$ al número de arcos del conjunto C para cualquier $C \subseteq E$, incluídos ciclos y caminos. Seguimos llamando $S = \{1, \ldots, \bar{s}\}$ al conjunto de *slots* disponibles. Nombramos como *camino simple* a un camino sin ciclos, y definimos como $P(G, i, j)$ al conjunto de caminos simples que conectan el nodo i con el nodo j en el grafo G, y como $E(G, d)$ al conjunto compuesto por los arcos e tales que existe un camino $P \in P(G, s(d), t(d))$ con $e \in P$. También utilizamos $V(G)$ y $E(G)$ para denotar el conjunto de vértices y arcos del grafo G, respectivamente.

3.2.1. Ciclos

Como efecto secundario de las restricciones (3.4), que obligan a tener al menos una ruta desde el origen al destino, para este modelo es imposible tener ciclos espurios que ingresen al origen de ninguna demanda en una solución factible. Llamamos a este tipo de ciclos espurios como ciclos de *tipo I*, de los cuales se muestra un ejemplo en la Figura 3.1a. Los ciclos de *tipo II* son permitidos por el modelo a pesar de que la función objetivo



(a) Ciclo de tipo I. **(b)** Ciclo de tipo II.

Fig. 3.1: Ciclos espurios.

los va a eliminar de cualquier solución óptima. Éstos son los ciclos espurios que salen del destino de alguna demanda. La Figura 3.1b muestra un ejemplo usando dos *slots*. Dado que no enunciamos restricciones análogas a (3.4) para el nodo de destino, los *slots* utilizados por el ciclo podrían ser cualquier subconjunto contiguo de al menos $v(d)$ *slots*, es decir, podrían ser los mismos o un conjunto diferente al utilizado por la ruta de la solución. Dicho conjunto de *slots* se verá restringido únicamente en los casos en que la intersección del ciclo y el camino que llega al destino no esté vacía (*i.e.*, cuando el ciclo y el camino comparten algún arco), como veremos más adelante.

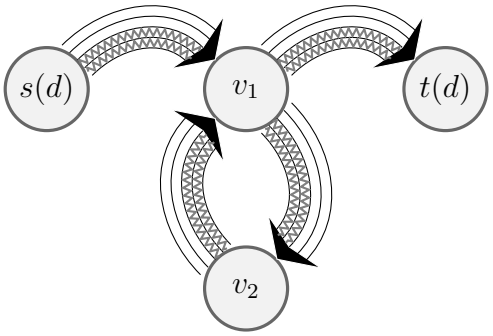
El tercer tipo de ciclos posibles permitidos por el modelo, los llamados ciclos de *tipo III*, son aquellos que están conectados a la ruta que conecta el origen con el destino de la demanda; es decir, la intersección de los nodos utilizados por el ciclo y los utilizados por la ruta que satisface la demanda no está vacía. Los dividimos en tres familias o sub-tipos.

- Tipo III-A: Ciclos que podrían ser parte de la ruta asignada a la demanda dado que usan los mismos *slots*. En este caso no podrían compartir arcos, sólo nodos debido a las restricciones de flujo, como puede verse en la Figura 3.2a.

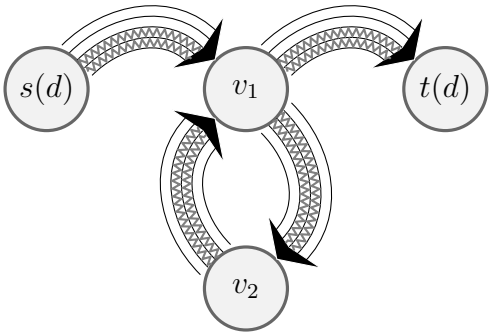
- Tipo III-B: Ciclos que no pueden pertenecer al *lightpath* que satisface la demanda porque el ciclo y el camino usan dos conjuntos diferentes de *slots*. Puede suceder que los *slots* utilizados por el ciclo se superpongan a los utilizados por la ruta. Como se indicó anteriormente, la ruta y el ciclo sólo pueden compartir nodos y no pueden compartir arcos. Un ejemplo de esta familia se representa en la Figura 3.2b.

- Tipo III-C: La última familia es aquella en la que la ruta y el ciclo utilizan conjuntos disjuntos de *slots*. En este caso el ciclo y la ruta pueden compartir nodos y arcos, como puede verse en los ejemplos de las Figuras 3.2c y 3.2d. Es posible que los *slots* utilizados por el ciclo y los utilizados por la ruta no formen un solo intervalo.

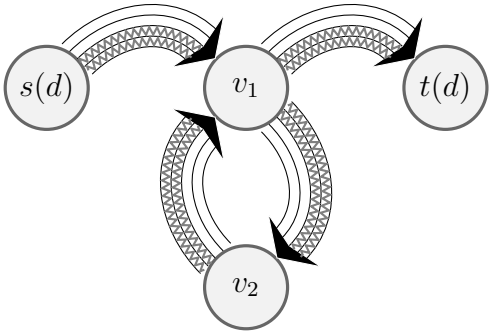
El cuarto y último tipo de ciclos espurios permitidos está compuesto por aquellos que no comparten ningún nodo con la ruta que satisface la demanda (ver Figura 3.3).



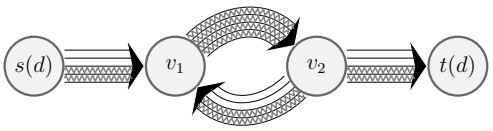
(a) Tipo III-A. El ciclo puede pertenecer al camino usado por el *lightpath* asignado a la demanda en una solución factible. Dicho *lightpath* no es óptimo, pero está compuesto por un camino y un conjunto de *slots* contiguos, por lo tanto es válido para las restricciones del modelo.



(b) Tipo III-B: El ciclo no puede pertenecer al camino utilizado por el *lightpath* asignado a la demanda en una solución factible. En este caso hay un solapamiento parcial de *slots*.



(c) Los ciclos espurios de tipo III-C no pueden pertenecer a la ruta asociada a un *lightpath* en una solución factible. El ciclo y el camino utilizan distintos conjuntos de *slots*.



(d) Ciclo espurio de tipo III-C compartiendo un arco con el camino.

Fig. 3.2: Ciclos espurios de tipo III.

3.2.2. Bifurcaciones

Otra característica no deseada que pueden presentar las soluciones subóptimas de este modelo es la que llamamos *bifurcación*. Tenemos una bifurcación cuando una demanda d se satisface mediante dos caminos diferentes utilizando al menos $v(d)$ *slots* en cada uno. Como hemos hecho con los ciclos, podemos dividir las bifurcaciones en muchos tipos según la posición de los arcos disjuntos.

Decimos que tenemos una bifurcación de *tipo I* cuando el primer y último nodo de la misma no son ni el origen ni el destino de la demanda, tal como se muestra en la Figura 3.4a.

Cuando el primer nodo de la bifurcación es el origen de la demanda, decimos que tenemos una bifrucación de *tipo II*. Puede verse un ejemplo en la Figura 3.4b. Cuando el último nodo de la bifurcación es el destino de la demanda, decimos que tenemos una bifurcación de *tipo III*. Un ejemplo de éstas es representado en la Figura 3.4c.

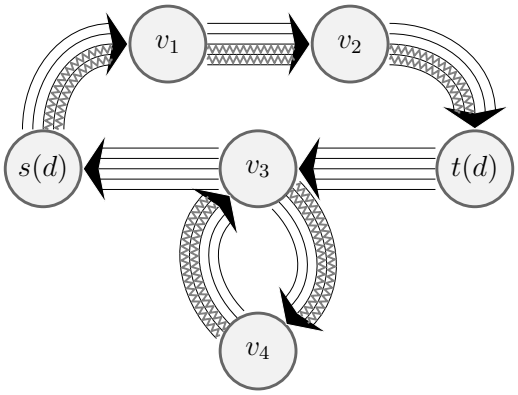
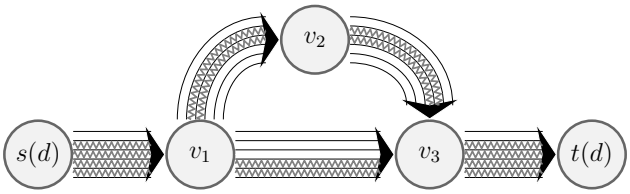
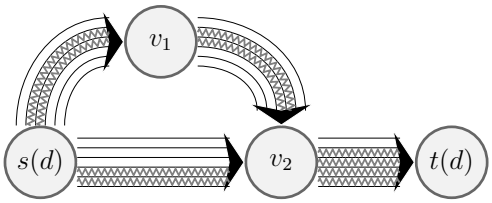


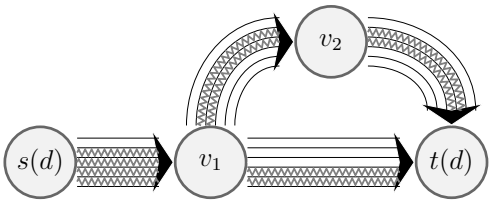
Fig. 3.3: Ciclo espurio de tipo IV que no podría pertenecer a la trayectoria de la demanda d en esta solución factible.



(a) Tipo I.



(b) Tipo II.

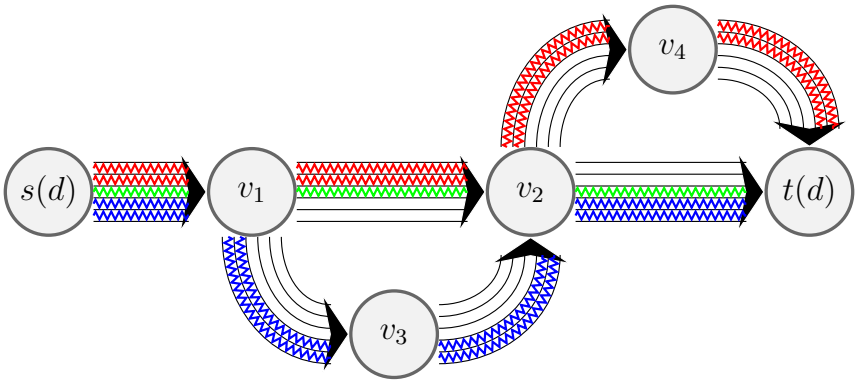


(c) Tipo III.

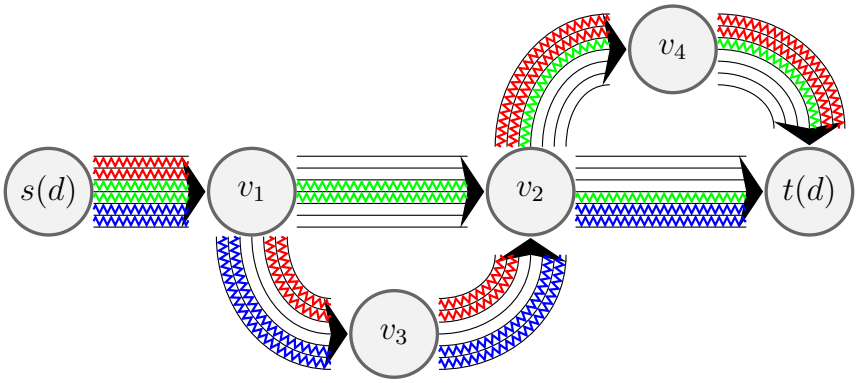
Fig. 3.4: Bifurcaciones espurias.

Como con los ciclos, a causa de la función objetivo, ninguna solución óptima del modelo DSL-BF va a contener una bifurcación de ningún tipo en las rutas utilizadas para satisfacer sus demandas. Sin embargo, en las soluciones factibles, podemos tener bifurcaciones de uno, dos o hasta tres tipos al mismo tiempo. Podemos incluso encontrarnos con un camino corriendo en paralelo a varios otros para alcanzar el destino, como puede verse en la Figura 3.5a, o varios caminos compartiendo algunos arcos pero no *slots*, o varios *slots* conformando todos un intervalo contiguo sobre un camino en una serie de arcos, pero estando separados en otros. Un ejemplo de este último caso puede observarse en la Figura 3.5b.

En ésta y las siguientes secciones proponemos diversas familias de desigualdades e igualdades válidas así como cortes optimales para $RSA(G, D, \bar{s})$. En particular, en esta sección, presentamos mayoritariamente cortes optimales con el fin de eliminar algunos o todos los ciclos y bifurcaciones descritos arriba. Debido a los teoremas enunciados al comienzo del capítulo, cada vez que tengamos una familia de desigualdades relacionada con los arcos salientes de un nodo particular, podemos formular la familia simétrica considerando los nodos entrantes de dicho arco, y viceversa. Asimismo, cuando definamos una familia de desigualdades que incluya de algún modo los *slots* más bajos, podemos enunciar



(a) El *lightpath* cuyos *slots* están coloreados de verde y que por si mismo no puede satisfacer de demanda, corre en paralelo al azul y rojo desde el origen al destino de la demanda.



(b) Los *slots* verdes son parte de un *lightpath* de 6 *slots* entre $s(d)$ y v_1, ellos solos conforman otro *lightpath* entre v_1 y v_2 y luego se separan para pertenecer a dos *lightpaths* distintos entre v_2 y $t(d)$. Por su lado, los *lightpaths* rojo y azul comparten sólo tres arcos y, en dos de éstos, hay *slots* sin usar en medio de los usados por ellos.

Fig. 3.5: Posibles escenarios en soluciones factibles para una instancia con una demanda d de volumen $v(d) = 2$. Los colores son únicamente para diferenciar más claramente el comportamiento de los *lightpaths*.

una familia simétrica considerando los *slots* más altos.

3.2.3. Nada desde el destino

Podemos eliminar fácilmente los ciclos de tipo II (que utilizan $t(d)$) para cada demanda $d \in D$, con la versión análoga de las $2|D|$ restricciones (3.4) aplicadas al conjunto de arcos $\delta^+(t(d))$. A esta familia la denominamos *nothingFromDst*, y podemos ver que no están implicadas por las restricciones del modelo dado que, en caso contrario, no podríamos tener ciclos del tipo II en ningúna solución factible. Podemos enunciarlas de la siguiente forma,

$$\sum_{s \in S} \sum_{e \in \delta^+(t(d))} u_{des} = 0 \qquad \forall d \in D. \tag{3.16}$$

Teorema 3.2.1. *Las igualdades (3.16) no cortan ninguna solución óptima de* $RSA(G, D, \bar{s})$.

Demostración. Sea u^* una solución óptima, y asumamos que para alguna demanda $d \in D$ existe un arco $e \in \delta^+(t(d))$ tal que $\sum_{s \in S} u^*_{des} > 0$. Como u^* es óptima, la demanda d es satisfecha por un *lightpath* compuesto por un camino simple (*i.e.*, sin nodos repetidos) P conectando $s(d)$ con $t(d)$ con al menos $v(d)$ *slots* consecutivos encendidos (*i.e.*, *slots* tales que sus variables correspondientes valen 1). Dado que $t(d)$ es el destino de P y P es un camino simple, entonces $e \notin P$. Por lo tanto, es posible definir una nueva solución factible u' que coincida con u^* en todas las variables a lo largo del camino simple P, *i.e.*, $u'_{de's} = u^*_{de's}$ para todo $s \in S$ y todo $e' \in P$, $u'_{de's} = 0$ para cada $e' \notin P$ y cada $s \in S$, y $u'_{d'**} = u^*_{d'**}$ para cada $d' \neq d$. Como d no utiliza e en u', entonces el valor de la función objetivo obtenido por u' es estrictamente mejor que el valor objetivo de u^*, una contradicción. □

3.2.4. Una única vez cada *slot*

Los siguientes cortes optimales eliminan la primera y segunda familia de ciclos de tipo III (A y B), pero permiten multiples ciclos si éstos usan conjuntos disjuntos de *slots* (III-C), y permiten ciclos de tipo IV. Para toda demanda, estos cortes optimales establecen que cada *slot* puede usarse como máximo en uno sólo de los arcos salientes de cada nodo interno del camino, *i.e.*,

$$\sum_{e \in \delta^+(i)} u_{des} \leq 1 \qquad \forall d \in D, \forall s \in S, \forall i \in V \setminus \{t(d)\}. \tag{3.17}$$

Denominamos a esta familia como *oneSlotOnceFromV*, y llamamos *oneSlotOnceFromSrc* al subconjunto obtenido al tomar $i = s(d)$. Separamos este caso especial para permitir una mayor flexibilidad al unirlos con otros cortes en el algoritmo *branch-and-cut*.

Teorema 3.2.2. *Las desigualdades (3.17) son cortes optimales para el modelo DSL-BF.*

Demostración. Sea u^* una solución óptima tal que u^* viola al menos una de las desigualdades (3.17). Entonces debe existir al menos un nodo $i \in V$ tal que la suma de las variables asociadas a un *slot* s sobre el conjunto $\delta^+(i)$ es mayor a 1. Esto significa que existe una bifurcación, por lo tanto dos caminos, y a causa de las restricciones (3.4) del modelo y las restricciones de flujo (3.2), cada uno de sus arcos tiene $v(d)$ *slots* utilizados por d, y dado que comparten *slots*, ambos caminos deben ser disjuntos en arcos. Por lo tanto, podemos definir una nueva solución u' que utiliza sólo uno de dichos caminos, *i.e.*, fijando en cero todas las variables asociadas a los *slots* de los arcos pertenecientes al otro camino, con función objetivo estrictamente menor que u^*, una contradicción. □

Para mostrar que las desigualdades (3.17) no son implicadas por el modelo DSL-BF, es suficiente presentar un contraejemplo. Consideremos una instancia del RSA tal que para cada demanda d, el grafo tiene dos caminos P_1 y P_2 tales que conectan $s(d)$ con $t(d)$, siendo P_1 y P_2 disjuntos en arcos, y asumamos que es posible asignar *lightpaths* con caminos simples al resto de las demandas sin hacer uso de un canal de $v(d)$ *slots* a lo largo de P_1 ni P_2. Un vector u' que satisface la demanda d utilizando dicho canal a lo largo de ambos caminos y que satisface el resto de las demandas es una solución factible para el modelo. Tenemos al menos un nodo, *i.e.*, $s(d)$ tal que los caminos P_1 y P_2 utilizan diferentes arcos de salida, definamos $e_1 \neq e_2$ a dichos arcos. Para cada s es cierto que $u_{de_1s} + u_{de_2s} = 2$, por lo que esta solución viola al menos una de las desigualdades (3.17), entonces éstas no pueden ser consecuencia de las restricciones del modelo.

También podemos formular la desigualdad simétrica considerando los arcos entrantes de cada nodo; *i.e.*, tomando la suma sobre todo $e \in \delta^-(i), i \in V \setminus \{s(d)\}$. Cuando $i = t(d)$ las llamamos *oneSlotOnceToDst*, en caso contrario *oneSlotOnceToV*. Debido a los Teoremas 3.1.2 y 3.1.3 podemos afirmar lo siguiente.

Corolario 3.2.1. *La familia de desigualdades* oneSlotOnceToV*, es decir,*

$$\sum_{e \in \delta^-(i)} u_{des} \leq 1 \qquad \forall d \in D,\ \forall s \in S,\ \forall i \in V \setminus \{s(d)\}, \tag{3.18}$$

y el caso particular oneSlotOnceToDst*,* i.e.*, cuando $i = t(d)$, son cortes optimales para el modelo DSL-BF.*

3.2.5. A lo sumo su volumen

Un modo de eliminar bifurcaciones de cualquier tipo en soluciones enteras, removiendo soluciones factibles pero no óptimas, es asegurando que para cada demanda $d \in D$ y nodo $i \in V$, la suma de *slots* utilizados por d en los arcos del conjunto $\delta^+(i)$ no sea mayor que $v(d)$. Esto puede ser forzado con la familia de cortes optimales

$$\sum_{e \in \delta^+(i)} \sum_{s \in S} u_{des} \leq v(d) \qquad \forall d \in D,\ \forall i \in V, \tag{3.19}$$

que llamamos *exactlyVdFromV* con el objeto de resaltar la idea de que, junto con (3.4), éstas fuerzan a cada demanda a ser satisfecha por exactamente su volumen. No son desigualdades válidas (dado que el modelo permite más de $v(d)$ *slots* asignados a cada demanda d), sino que son cortes optimales, debido a que siempre existe una solución óptima con exactamente $v(d)$ *slots* asignados a d. Cuando $i = s(d)$, removemos sólo bifurcaciones del tipo II; nos referimos a esta sub-familia como *exactlyVdFromSrc*, es decir,

$$\sum_{e \in \delta^+(s(d))} \sum_{s \in S} u_{des} \leq v(d) \qquad \forall d \in D. \tag{3.20}$$

Teorema 3.2.3. *Las desigualdades (3.20) no remueven ninguna solución óptima de la cápsula convexa $RSA(G, D, \bar{s})$.*

Demostración. Sea u^* una solución óptima removida por al menos una de las desigualdades (3.20). A causa de su optimalidad, cada demanda d debe estar satisfecha por un *lightpath* compuesto por un canal de $v(d)$ *slots* y un camino simple y sin bifurcaciones, pero la violación de la desigualdad implica que la cantidad de *slots* utilizada por una demanda d' en $\delta^+(s(d'))$ debe ser al menos $v(d') + 1$, una contradicción. □

Aplicando el mismo argumento en cada nodo del camino utilizado por la demanda llegamos al siguiente resultado.

Corolario 3.2.2. *Las desigualdades (3.19) son cortes optimales para la formulación DSL-BF.*

Una solución en la cual cada demanda es satisfecha mediante un *lightpath*, y al menos una de ellas, digamos d', utiliza al menos $v(d') + 1$ *slots* a lo largo de su camino es una

solución factible para el modelo pero no cumple las desigualdades (3.19) ni (3.20); por lo tanto, éstas no son implicadas por las restricciones del modelo.

Debido al Teorema 3.14, podemos también pedir que a lo sumo $v(d)$ *slots* alcancen el destino de cada demanda $d \in D$, tomando $e \in \delta^-(t(d))$ en (3.20). Podemos incluso decir lo mismo sobre los arcos entrantes a cada nodo en el grafo, *i.e.*, lo análogo a lo dicho por (3.19). Denominamos dichas familias *exactlyVdToDst* y *exactlyVdToV*, respectivamente.

Corolario 3.2.3. *La familia de desigualdades* exactlyVdToV*, a saber,*

$$\sum_{e \in \delta^-(i)} \sum_{s \in S} u_{des} \leq v(d) \qquad \forall d \in D,\ \forall i \in V, \tag{3.21}$$

y su caso particular cuando $i = t(d)$, i.e., exactlyVdToDst*, son cortes optimales para el modelo DSL-BF y no son implicadas por sus restricciones.*

3.2.6. Sin bifurcaciones

Otra forma de evitar bifurcaciones de cualquier tipo es agregando la siguiente familia de cortes optimales. Esta familia establece que si un *slot* s en un arco $e \in \delta^+(i)$ es usado por d, entonces ningún otro arco saliente de i puede tener *slots* usados por d, a saber,

$$\sum_{e' \in \delta^+(i) \setminus \{e\}} \sum_{s' \in S} u_{de's'} \leq v(d)(1 - u_{des}) \qquad \forall d \in D,\ \forall i \in V,\ \forall e \in \delta^+(i),\ s \in S. \tag{3.22}$$

Denominamos a esta familia de cortes optimales como *notBranchFromV*, y llamamos *notBranchFromSrc* al subconjunto obtenido al tomar $i = s(d)$. Estas desigualdades también fuerzan a d a usar a lo sumo $v(d)$ *slots*. Podemos relajarlas un poco agrandando la *big M*, *i.e.*, reemplazando $v(d)$ por $\bar{s}$, por ejemplo.

Dado que el modelo permite, por ejemplo, soluciones enteras que satisfagan una demanda utilizando dos caminos, mientras que las desigualdades (3.22) no, podemos concluir que éstas no están implicadas por las restricciones del modelo.

Teorema 3.2.4. *Las desigualdades (3.22) son cortes optimales para el modelo* DSL-BF*.*

Demostración. Sea u^* una solución óptima. Si u^* viola al menos una de las desigualdades (3.22), entonces debe existir una demanda $d \in D$ y un nodo $i \in V$ tales que d utiliza al menos dos arcos de $\delta^+(i)$. A causa de las restricciones de contigüidad e integralidad, ambos arcos deben tener al menos $v(d)$ *slots* usados por d, y debido a las limitaciones de flujo, cada uno debe pertenecer al ciclo o bien a la ruta que conecta $s(d)$ con $t(d)$. Dado que una demanda en cualquier solución óptima debe ser satisfecha por un *lightpath* compuesto por un camino simple y un canal con exactamente $v(d)$ *slots*, entonces u^* no puede ser óptima. □

Corolario 3.2.4. *Las desigualdades* notBranchFromSrc *son cortes optimales para el modelo DSL-BF.*

Asumamos que las restricciones (3.2)-(3.7) valen, entonces podemos mostrar que las desigualdades *notBranchFromSrc* no implican *exactlyVdFromSrc* presentando un contraejemplo. Consideremos una instancia del RSA con $\bar{s} > 1$, y una única demanda d con volumen igual a 1. Asumamos que la instancia es factible, entonces debe existir un camino P que una $s(d)$ con $t(d)$. Sea u' un vector en $\{0,1\}^{|D| \times |E| \times \bar{s}}$ tal que $u'_{1e1} = u'_{1e2} = 1$ para

cada arco $e \in P$ y $u'_{des} = 0$ en caso contrario. Este vector es una solución factible para la instancia dada y satisface las desigualdades *notBranchFromV* perio viola las desigualdades *exactlyVdFromSrc*.

Ahora presentaremos un contraejemplo para mostrar lo contrario, es decir, que si las restricciones (3.2)-(3.7) valen, entonces las desigualdades *exactlyVdFromSrc* no implican *notBranchFromSrc*. Consideremos una instancia del RSA con G tal que existen dos nodos a y b, y dos caminos P_1 y P_2, disjuntos en arcos, conectando a con b. Asumamos $\bar{s} = 5$ y D compuesto por una sola demanda $d = (a, b, 4)$. Sea $u' \in \mathbb{R}^{|D||E|\bar{s}}$ un vector tal que $u'_{de*} = (0, \frac{1}{2}, \frac{1}{2}, 1, \frac{1}{2})$ para cada $e \in P_1$; $u'_{de*} = (0, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}, \frac{1}{4})$ para $e \in P_2$, y $u'_{des} = 0$ para el resto de los arcos y *slots*. El vector u' satisface las restricciones de flujo, dado que forma dos caminos disjuntos entre $s(d)$ y $t(d)$, trivialmente satisface las restricciones de no solapamiento, y podemos fácilmente ver que no viola las restricciones de contigüidad. Asimismo, esta solución fraccionaria no viola las desigualdades *exactlyVdFromSrc* porque la suma de cada variable perteneciente a los arcos que salen de $s(d)$ es exactamente $v(d)$, sin embargo, u' viola la desigualdad *notBranchFromSrc* tomando e como el primer arco perteneciente a P_1 y $s = 4$.

Podemos también concluir que si las restricciones (3.2)-(3.7) valen, entonces las desigualdades (3.22) no implican (3.19) ni viceversa.

Debido a los Teoremas 3.1.2 y 3.1.3, podemos considerar los arcos entrantes, *i.e.*, $\delta^-(i)$ en cada nodo $i \in V \setminus s(d)$, obteniendo la familia denominada *notBranchToV*. Llamamos *notBranchToDst* a la sub-familia obtenida cuando $i = t(d)$.

Corolario 3.2.5. *Las desigualdades* notBranchToV*, a saber,*

$$\sum_{e' \in \delta^-(i) \setminus \{e\}} \sum_{s' \in S} u_{de's'} \leq v(d)(1 - u_{des}) \qquad \forall d \in D,\ \forall i \in V,\ \forall e \in \delta^-(i),\ \forall s \in S, \tag{3.23}$$

y su caso particular obtenido cuando $i = t(d)$, son cortes optimales para el modelo DSL-BF y no son implicadas por las restricciones del modelo.

3.2.7. Cada *slot* es utilizado en la misma cantidad de arcos

Con las desigualdades *oneSlotOnceFromSrc* tenemos un modo de saber si un *slot* s particular es utilizado por una demanda d. Como $\sum_{e \in \delta^+(s(d))} u_{des}$ es exactamente el lado izquierdo de las desigualdades (3.17) tomando $i = s(d)$ para cada s y d, *i.e.*, *oneSlotOnceFromSrc*, agregando éstas junto con las desigualdades

$$\sum_{e \in E} u_{des'} \leq \sum_{e \in E} u_{des} + |E|\Big(1 - \sum_{e \in \delta^+(s(d))} u_{des}\Big) \qquad \forall d \in D,\ \forall s, s' \in S,\ s' \neq s, \tag{3.24}$$

podemos eliminar diversas bifurcaciones y ciclos. Estas desigualdades denominadas *eqAmountOfAsForEachUsedS* junto con *oneSlotOnceFromSrc*, *i.e.*, pidiendo a su vez que estas últimas sean válidas (notar que si no se agregan estas últimas, no podemos afirmar lo siguiente), dicen que si el *slot* s es usado por una demanda d en algún arco de $\delta^+(s(d))$, entonces para algún otro *slot* s' la cantidad de arcos en los cuales s' es usado por d debe ser a lo sumo la cantidad de arcos en los cuales d usa s. Estas $\bar{s}^2|D|$ desigualdades eliminan los ciclos de tipo III-B y III-C y aquellos de tipo IV que usan un conjunto de *slots* diferente del usado por el camino. Notar que aún estarían permitidos aquellos ciclos de tipo III-A y los de tipo IV con el mismo conjunto de *slots* que la ruta.

Si asumimos que las restricciones (3.2)-(3.7) valen, entonces, es posible demostrar que las desigualdades (3.17) y (3.19) no implican (3.24). Consideremos una instancia del RSA. Sea $d' \in D$ una demanda particular y sea P un camino simple de l arcos que conecta $s(d')$ con $t(d')$. Para simplificar la demostración, asumamos que la capacidad de los arcos $\bar{s}$ es mayor que $2v(d')$, y sean Ch_1 y Ch_2 dos rangos distintos de exactamente $v(d')$ *slots* contiguos cada uno. Asumamos que existe un ciclo C disjunto en arcos con P y tal que $|C| \neq l$. Sea $u' \in \{0,1\}^{|D||E|\bar{s}}$ un vector que satisface todas las demandas $d \neq d'$ sin utilizar ningún *slot* del canal Ch_1 en el camino P y ningún *slot* del canal Ch_2 en el ciclo C. Fijemos $u'_{d'es} = 1$ para cada *slot* $s \in Ch_1$ y cada arco $e \in P$ de modo tal que u' satisfaga d', pero también fijemos $u'_{d'es} = 1$ para todo *slot* $s \in Ch_2$ en los arcos $e \in C$. De este modo u' satisface todas las restricciones del modelo, las desigualdades (3.17) y (3.19) pero, dado que cualquier *slot* utilizado en el camino es usado en diferente cantidad de arcos que en el ciclo, este vector no satisface (3.24).

Cualquier solución factible con una bifurcación que comparte arco tiene que usar $k\ v(d)$ *slots* diferentes para satisfacer una demanda $d \in D$, con $k \in \mathbb{N}$. Si uno de los caminos de la bifurcación es más largo que el otro, lo cual es permitido por el modelo, tenemos todos sus *slots* utilizados en más arcos que aquellos pertenecientes al camino más corto, violando por tanto varias de las desigualdades (3.24), con lo cual éstas no pueden ser implicadas por las restricciones del modelo.

Teorema 3.2.5. *Las desigualdades (3.24) son cortes optimales para el modelo dado por (3.2)-(3.7) y (3.17).*

Antes de dar una demostración del teorema vamos a enunciar una definición que será utilizada más adelante. Consideremos una instancia del RSA.

Definición 3.2.1. *Llamamos* lightpath minimal *de una demanda d a un par (P, C) donde C es un canal de $v(d)$* slots *y P es un camino simple entre $s(d)$ y $t(d)$ tal que no existe otra solución en la que d es satisfecha utilizando un camino más corto que P y usando los mismos* lightpaths *para el resto de las demandas.*

De este modo, una solución factible tal que existe una demanda satisfecha con un *lightpath* no minimal no es óptima.

Demostración. Sea u^* una solución óptima que satisface toda desigualdad (3.17). Asumamos que u^* viola al menos una de las desigualdades (3.24) para la demanda d y el *slot* s. Como el vector u^* satisface las desigualdades (3.17), entonces $\sum_{e \in \delta^+(s(d))} u_{des} \in \{0,1\}$. Si esta suma es 0, el número de arcos en los cuales s' es utilizado debe ser mayor que la cantidad total de arcos del grafo para violar (3.24), por lo tanto $\sum_{e \in \delta^+(s(d))} u_{des} = 1$. Esto implica que el *slot* s es usado por d y que existe un $s' \neq s$ usado por d en más arcos que s en u^*. Dado que u^* es una solución óptima, d debe ser satisfecha por un *lightpath* minimal compuesto por un camino P y un canal de $v(d)$ *slots*, y, por lo tanto, s y s' deben ser utilizados solamente en P, es decir, en la misma cantidad de arcos, una contradicción. □

Sumando sobre todo otro camino

Podemos combinar la suma de las desigualdades (3.24) sobre todos los *slots* con la idea principal de las desigualdades (3.19), *i.e.*, forzando a las demandas a usar a lo sumo sus volúmenes, generando así otra familia de desigualdades válidas, la cual, otra vez unida con

oneSlotOnceFromSrc (es decir, requiriendo también el cumplimiento de éstas), establecen que para cada demanda d si un *slot* s es usado por d entonces el número de *slots* usado por la demanda en todo el grafo debe ser $v(d)$ veces la cantidad de arcos en los cuales s es usado. Las desigualdades obtenidas así son denominadas *eqAmountOfAsForTheSumOfSs* y pueden expresarse como

$$\frac{1}{v(d)}\Big(\sum_{e\in E}\sum_{s'\in S} u_{des'}\Big) \leq \sum_{e\in E} u_{des} + |E|\Big(1 - \sum_{e\in\delta^+(s(d))} u_{des}\Big) \qquad \forall d \in D,\, \forall s \in S. \tag{3.25}$$

Para demostrar que las desigualdades (3.25) no son implicadas por las restricciones del modelo es suficiente ver que cualquier solución u' tal que alguna demanda d es satisfecha utilizando un camino con una longitud l sobre un canal C_1, y un ciclo espurio de l' arcos, utilizando éste un canal C_2, de forma tal que $l' \neq l$ y $C_1 \neq C_2$, satisface las restricciones del modelo pero no todas las desigualdades (3.25).

Teorema 3.2.6. *Las desigualdades (3.25) son cortes optimales para el modelo dado por (3.2)-(3.7) y (3.17).*

Demostración. Sea u^* una solución óptima y asumamos que ésta viola al menos una de las desigualdades (3.25). Dado que (3.17) son válidas y $u^* \in \{0,1\}^{|D||E|\bar{s}}$, por lo tanto $\sum_{e\in\delta^+(s(d))} u^*_{des} = \alpha$ debe ser 0 o 1 para cada $s \in S$ y $d \in D$. Si $\alpha = 0$, entonces debe existir una demanda $d \in D$ y un *slot* $s \in S$ tales que

$$\sum_{e\in E}\sum_{s'\in S} u^*_{des'} > v(d)|E|,$$

con lo que u^* no es óptima porque usa más *slots* que $v(d)|E|$. Si $\alpha = 1$, entonces deben existir una demanda $d \in D$ y un *slot* $s \in S$ tales que

$$\sum_{e\in E}\sum_{s'\in S} u^*_{des'} > v(d) \sum_{e\in E} u^*_{des}, \tag{3.26}$$

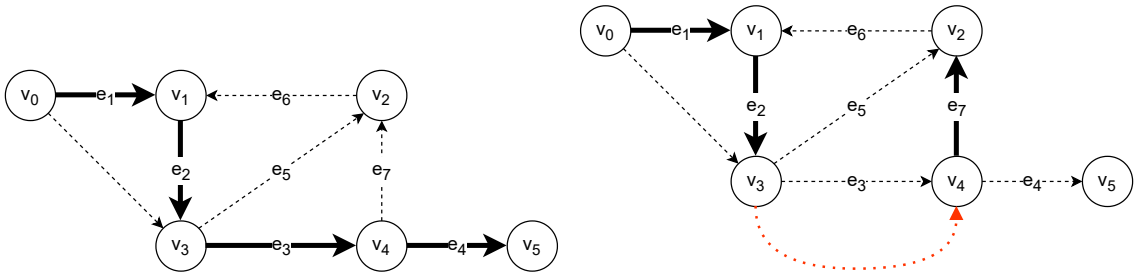
pero $\sum_{e\in E} u^*_{des} \geq 1$, porque el *slot* s es usado por d en $\delta^+(s(d))$. Dado que u^* es óptima, d debe ser satisfecha por un *lightpath* minimal compuesto por un canal C y un camino P que conecta $s(d)$ con $t(d)$. Dado que el *slot* s debe pertenecer a C, este *slot* debe ser utilizado todo a lo largo del camino P, por lo que el lado derecho de (3.26) es al menos $v(d)|P|$. Pero la cantidad de *slots* del *lightpath* compuesto por P y C es suficiente para satisfacer d, y éstos son exactamente $v(d)|P|$ *slots*, por lo tanto (3.26) no puede ser satisfecha por ninguna solución óptima, entonces u^* no es óptima, una contradicción. □

Asumamos que las restricciones (3.2)-(3.7) y las desigualdades (3.17) y (3.19) valen. Entonces, podemos probar que las desigualdades (3.25) no son implicadas por (3.24). Sea u' una solución de una instancia del RSA con una única demanda d de volumen $v(d) = 1$ tal que es satisfecha utilizando un camino de dos arcos, que conecta $s(d)$ con $t(d)$ y usando el *slot* 1, y un ciclo espurio de dos arcos ij y ji, utilizando el *slot* 2. El vector u' es una solución factible que satisface las desigualdades (3.17) y (3.19) porque para cada vértice $a \in V$, d usa a lo sumo un arco de $\delta^+(a)$ con a lo sumo $v(d)$ *slots*. Como cada *slot* es utilizado en exactamente dos arcos, u' también satisface (3.24). Sin embargo, u' no satisface (3.25) porque utiliza 4 *slots* en todo el grafo, lo cual es mayor que $v(d)$ veces el número utilizado por cada *slot*, *i.e.*, 2.

Es posible mejorar las desigualdades (3.24) y (3.25) precomputando el camino más largo, sin ciclos de ningún tipo, entre el origen y el destino de cada demanda y reemplazando $|E|$ en el lado derecho de las desigualdades por el número de arcos de tal camino máximo. Esta pequeña modificación tendría efecto si el *slot* s no fuera usado por la demanda, forzando a s' a no usar más arcos que el camino máximo, *i.e.*, apagando las variables de cualquier ciclo espurio. Como el problema de camino máximo es $\mathcal{NP}$-hard, habría un *trade-off* entre el costo de resolver dicho problema y la mejora en los tiempos proporcionada por las desigualdades así obtenidas.

3.2.8. Máximo *matching* de caminos en un grafo

Dado un subconjunto de arcos $E' \subseteq E$, llamamos $P_{E'}$ al conjunto de todos los caminos sin ciclos en E'. Denominamos como *matching de caminos* a cada conjunto de arcos de los caminos de $P_{E'}$ disjuntos en vértices, y definimos como $M_{E'}$, al cardinal del *matching* de caminos maximal, *i.e.*, el conjunto de arcos de los caminos disjuntos en vértices con el mayor número de arcos. Entonces cada demanda en una solución óptima puede utilizar a lo sumo $M_{E'}$ arcos de E'. Dos ejemplos de esto son mostrados en la Figura 3.6. En dichos ejemplos, las líneas negras representan los arcos que pertenecen a E'. Las líneas negras continuas representan los usados por la demanda d, mientras que las líneas punteadas representan los arcos no utilizados por d (aunque posiblemente utilizados por otras demandas). En el ejemplo de la Figura 3.6a, los arcos utilizados por la demanda d en E' forman un camino, mientras que en el ejemplo de la Figura 3.6b el conjunto E' captura dos partes del camino utilizado por la demanda. En este caso la demanda debe también utilizar arcos no pertenecientes a E' –representado con la línea punteada roja– conectando v_3 con v_4, debido a las restricciones de flujo.



(a) La demanda usa sólo un camino que pertenece al subgrafo.

(b) La demanda usa dos caminos disjuntos en vértices y que pertenecen al subgrafo.

Fig. 3.6: Dos ejemplos de caminos simples utilizados por una demanda y que pertenecen a un subconjunto de arcos.

Estas observaciones motivan la familia de desigualdades mostrada en (3.27). Éstas pueden ser convertidas en cortes optimales si también forzamos a cada demanda a ser satisfecha por exactamente su volumen, como se muestra en (3.28). Un caso particular de esta familia es obtenido tomando $E' = \delta^+(i)$ para cada $i \in V$, *i.e.*, (3.19).

$$\sum_{e \in E'} u_{des} \leq M_{E'} \qquad \forall d \in D, \forall s \in S, \forall E' \subseteq E, \tag{3.27}$$

$$\sum_{s \in S} \sum_{e \in E'} u_{des} \leq v(d) M_{E'} \qquad \forall d \in D, \forall E' \subseteq E. \tag{3.28}$$

Dado que el modelo permite soluciones factibles con ciclos espurios, y ningún ciclo entero satisface (3.27), entonces las restricciones del modelo no implican a estas últimas. Asimismo, cualquier solución factible con un ciclo entero espurio viola (3.28) tomando E' como el conjunto de arcos del ciclo, por ejemplo; por lo tanto, las restricciones del modelo no implican a estas desigualdades tampoco.

Teorema 3.2.7. *Las desigualdades (3.27) son cortes optimales para el modelo DSL-BF.*

Demostración. Sea u^* una solución óptima que viola al menos una de las desigualdades de esta familia. Entonces, deben existir una demanda $d \in D$, un *slot* $s \in S$ y un conjunto de arcos $E' \subseteq E$ tales que $\sum_{e \in E'} u^*_{des} > M_{E'}$. Esto es, la demanda d utiliza el *slot* s en un conjunto de arcos $Y \subseteq E'$ tal que $|Y| > M_{E'}$. Sin embargo, dado que u^* es óptima, sus caminos no pueden contener ciclos ni bifurcaciones, con lo que Y debe ser un conjunto de rutas, *i.e.*, $Y \subseteq P_{E'}$, entonces $|Y| \leq M_{E'}$. Por lo tanto, u^* no puede ser óptima. □

Teorema 3.2.8. *Las desigualdades (3.28) son cortes optimales para el modelo DSL-BF.*

Demostración. Sea u^* una solución óptima para una instancia del RSA y asumamos que u^* viola alguna de las desigualdades (3.28). Entonces, deben existir $d \in D$ y $E' \subseteq E$ tales que $\sum_{s \in S} \sum_{e \in E'} u^*_{des} > v(d)\ M_{E'}$. Sea $Y \subseteq E'$ el conjunto de arcos usados por d en E'. Dado que u^* es óptima, d debe ser satisfecha por un *lightpath* minimal con un camino P y un canal C, entonces $Y \subseteq P$ no puede contener ciclos ni bifurcaciones, *i.e.*, $Y \subseteq P_{E'}$, con lo que $|Y| \leq M_{E'}$. Es también cierto que el canal usado por d debe ser C a lo largo de todo el camino P, en particular en Y, y dado que d no puede usar ningún arco $e \notin P$ ni más de $v(d)$ slots (por ser minimal), entonces $\sum_{s \in S} \sum_{e \in E'} u^*_{des} = v(d)|Y|$, por lo tanto u^* no puede ser óptima. □

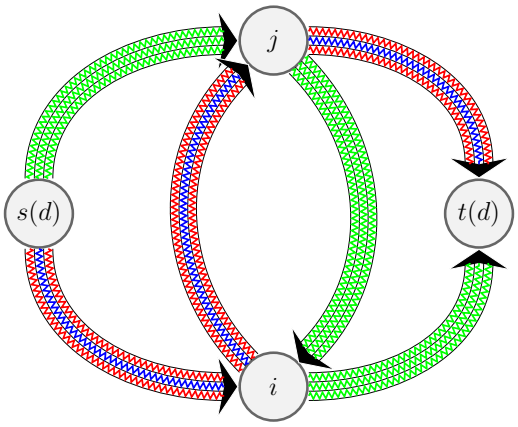


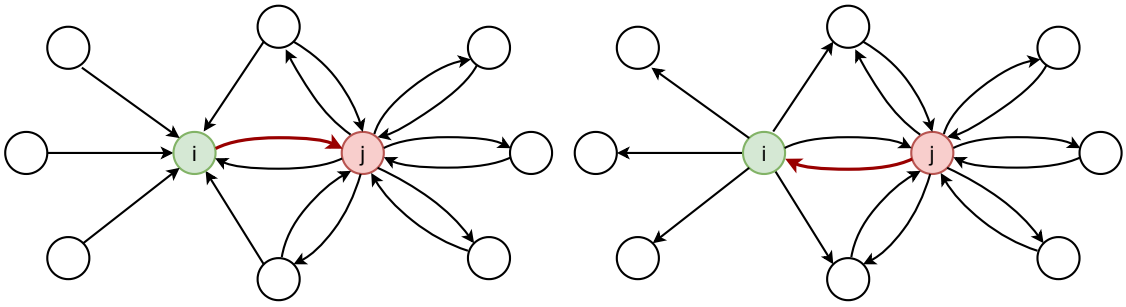
Fig. 3.7: Una solución fraccionaria en la cual una demanda d es satisfecha usando dos caminos que contienen los arcos ij y ji. Los colores verde, rojo y azul significan, respectivamente, que la demanda d usa $\frac{1}{3}, \frac{1}{2}$ o el *slot* completo.

Para demostrar que el modelo dado por las restricciones (3.2)-(3.7) y las desigualdades (3.28) no implica a la familia (3.27) basta con presentar un contraejemplo. Asumamos una instancia del RSA con $\bar{s} = 3$, una única demanda d de volumen $v(d) = 3$ y una solución fraccionaria u' en la cual d es satisfecha por dos *lightpaths* utilizando los caminos P_1 y P_2 como se muestra en la Figura 3.7, de modo tal que P_1 usa el arco ij y P_2 el arco ji,

con $u'_{de*} = (\frac{1}{2}, 1, \frac{1}{2})$ para cada e en P_1 y $u'_{de*} = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ para cada $e \in P_2$. Esta solución fraccionaria satisface todas las restricciones del modelo excepto las de integralidad, y podemos ver que u' satisface cada una de las desigualdades (3.28) pero u' viola (3.27) dado que, tomando $E' = \{(s(d), i), (i, j), (j, i), (i, t(d))\}$, entonces $M_{E'} = 2$ pero $\sum_{e \in E'} u'_{des} = 4$ para $s = 1$.

Double Brooms

Dado un arco $ij \in E$, definimos al conjunto de arcos $\delta^-(i) \cup \delta(j)$ como *double broom entrante* (InDBroom) asociado a ij como muestra la Figura 3.8a. De igual modo, definimos a $\delta^+(i) \cup \delta(j)$ como *double broom saliente* (OutDBroom) asociado con $ji \in E$; ver Figura 3.8b. Notar que el arco ij debe existir en orden a tener un InDBroom. Análogamente con ji para el OutDBroom. Estos arcos son llamados arcos *maestros* en ambos casos, y son resaltados en los grafos mostrados en la Figura 3.8.



(a) Un InDBroom está compuesto por todos los arcos incidentes a j y todos los arcos entrantes a i.

(b) Un OutDBroom está compuesto por todos los arcos incidentes a j y todos los arcos salientes de i.

Fig. 3.8: Los tipos posibles de double brooms.

Podemos ver que si DB es un InDBroom o un OutDBroom en un grafo G, entonces $M_{DB} = 3$, *i.e.*, la máxima cantidad de arcos de DB utilizada por una demanda en una solución óptima debe ser menor o igual a 3. Los casos particulares de (3.28) en los cuales E' es un InDBroom o un OutDBroom son llamados *incomingDBrooms* y *outgoingDBrooms*, respectivamente.

3.2.9. Un camino en un subgrafo

Sea E' un subconjunto de arcos de E. La cantidad de arcos de un conjunto de caminos dentro de E' debe ser menor que el número de nodos de E'. Las desigualdades (3.29) y (3.30) son dos familias extensas de cortes optimales para $RSA(G, D, \bar{s})$ que expresan este hecho. Dado que $M(P_{E'})$ es un conjunto de caminos disjuntos por vértices, el número total de arcos que pertenecen a ellos, *i.e.*, $M_{E'}$, debe ser menor que $|V(E')|$, con lo que (3.29) y (3.30) están dominadas por las desigualdades (3.27) y (3.28), respectivamente. El objetivo buscado al presentar estas familias es enunciar luego diferentes subgrupos, los cuales, debido a su menor cardinalidad y algunas particularidades que presentan, resultan

más fáciles de separar.

$$\sum_{e \in E'} u_{des} \leq (|V(E')| - 1) \qquad \forall d \in D, \forall s \in S, \forall E' \subseteq E, \tag{3.29}$$

$$\sum_{s \in S} \sum_{e \in E'} u_{des} \leq v(d)(|V(E')| - 1) \qquad \forall d \in D, \forall E' \subseteq E. \tag{3.30}$$

Por medio de dos contraejemplos podemos mostrar que las restricciones (3.2)-(3.7) y (3.30) no implican la familia (3.19), ni viceversa. Dada una instancia del RSA, cualquier solución factible que satisface una demanda d utilizando sólo dos caminos que comparten al menos un arco ij, satisface las restricciones del modelo y no viola las desigualdades (3.30), pero viola al menos una de las desigualdades (3.19) en el nodo i dado que, a causa de las restricciones de flujo, ambos caminos utilizan distintos rangos de *slots*, por lo que debe ocurrir que $\delta^+(i) > v(d)$. Teniendo una solución factible (*i.e.*, con todas sus variables tomando valor entero) que satisface una demanda d con un camino y un ciclo aislado, ambos usando exactamente $v(d)$ *slots*, hemos probado la inversa. Es decir, esta solución viola una de las desigualdades (3.30) al tomar el ciclo como E', mientras que satisface las restricciones del modelo y cada una de las desigualdades (3.19).

En nuestros experimentos consideramos las subfamilias de (3.29) y (3.30) generadas tomando E' como

- un ciclo no dirigido en G (*i.e.*, descartando la orientación de los arcos), obteniendo las subfamilias *cyclesBySlot* y *cycles*, respectivamente,
- el conjunto de todas las aristas inducidas por los vértices de un ciclo en G, obteniendo las subfamilias de desigualdades *extendedCyclesBySlot* y *extendedCycles*, respectivamente, y
- el conjunto $\{ij, ji\}$, cuando ambos arcos existen entre los nodos i y j, obteniendo las subfamilias *simplestCyclesBySlot* y *simplestCycles*, respectivamente.

Finalmente, para dos arcos $ij, ji \in E$ (cuando tal estructura existe), consideramos la desigualdad

$$\sum_{s' \in S} u_{d,ji,s'} \leq v(d)(1 - u_{d,ij,s}) \qquad \forall ij, ji \in E, \forall d \in D, \forall s \in S. \tag{3.31}$$

Esta desigualdad, que denominamos *simplestCyclesImplication*, establece que si la demanda d utiliza el *slot* s en el arco ij, entonces d no puede utilizar ningún *slot* en el arco ji, y no puede usar más que $v(d)$ *slots* en ningún arco, lo cual claramente vale en cualquier solución óptima.

Para ver que las desigualdades (3.31) no son implicadas por las restricciones del modelo dado por (3.2)-(3.7) junto con (3.28), consideremos una instancia del RSA con una única demanda d de volumen $v(d) = 3$ y asumamos una solución factible u' en la cual d es satisfecha por dos caminos P_1 y P_2, como se muestra en la Figura 3.7, de modo tal que P_1 usa el arco ij, y P_2 el arco ji, con $u'_{de*} = (\frac{1}{2}, 1, \frac{1}{2})$ para cada e en P_1 y $u'_{de*} = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ para cada $e \in P_2$. Esta solución fraccionaria u' satisface todas las restricciones del modelo excepto las de integralidad, y también satisface cada desigualdad (3.28), sin embargo, u' viola (3.31) dado que tomando el *slot* $s = 2$ tenemos $u'_{d,ij,s} = 1$, y la suma sobre cada *slot* en ji es 1, lo cual es mayor que $v(d)(1 - u'_{d,ij,s}) = 0$.

Las desigualdades (3.31) sólo están definidas para arcos ij cuyo reverso ji también pertenece al grafo, y eliminan dos tipos de soluciones factibles: por un lado las soluciones en las cuales una demanda d utiliza algún ciclo que incluye ambos arcos ij y ji, y por otro las soluciones en las cuales d utiliza más de $v(d)$ *slots* en un arco aunque no use el reverso. Motivo por el cual, cualquier solución donde una demanda d utiliza más de $v(d)$ *slots* solamente en arcos tales que sus reversos no pertenecen al grafo viola (3.19) pero no (3.31). Por lo tanto, las restricciones del modelo (3.2)-(3.7) junto con las desigualdades (3.31) no implican (3.19). Asimismo, las restricciones del modelo (3.2)-(3.7) junto con (3.31) no implican (3.28) ni (3.30).

3.3. Desigualdades de contigüidad

En esta sección presentamos desigualdades e igualdades válidas relacionadas con los requerimientos de contigüidad, a saber, que cada demanda debe utilizar *slots* consecutivos.

3.3.1. *Slots* principales

Estas familias están basadas en el Teorema 2.3.1 donde, dada una demanda d, la restricción de contigüidad es obtenida principalmente agrupando los *slots* que están distanciados entre sí por el volumen de dicha demanda d. La adaptación del Teorema 2.3.1, que nosotros denominamos como teorema *de unos consecutivos*, establece que si las igualdades (3.32) y las desigualdades (3.33) son satisfechas por un vector binario $u \in \{0,1\}^{|D||E|\bar{s}}$, para una demanda $d \in D$ y un arco $e \in E$, entonces d utiliza exactamente $v(d)$ *slots* contiguos en e.

$$\sum_{\substack{s\in S:\\ s \,\equiv\, i \;(v(d))}} u_{des} \;=\; 1 \qquad\qquad \forall i \in \{1,\ldots,v(d)\} \tag{3.32}$$

$$\sum_{\substack{s\in\{1,\ldots,i\}:\\ s \,\equiv\, i \;(v(d))}} u_{des} \;\geq\; \sum_{\substack{s\in\{1,\ldots,i-1\}:\\ s+1 \,\equiv\, i \;(v(d))}} u_{des} \qquad\qquad \forall i \in \{1,\ldots \bar{s}+1-v(d)\}. \tag{3.33}$$

La primer familia que proponemos es una generalización de (3.33), denominada *contiguityIneqs*, y está definida por

$$\sum_{\substack{s\in\{1,\ldots,i\}:\\ s \,\equiv\, i \;(v(d))}} u_{des} \;\geq\; \sum_{\substack{s\in\{1,\ldots,i-1\}:\\ s+1 \,\equiv\, i \;(v(d))}} u_{des} \qquad\qquad \forall d \in D,\, \forall e \in E,\, \forall i \in S. \tag{3.34}$$

Teorema 3.3.1. *Las desigualdades (3.34) son cortes optimales para el modelo DSL-BF.*

Demostración. Sea u^* una solución óptima para una instancia del RSA tal que u^* viola al menos una de las desigualdades (3.34). Entonces, deben existir una demanda $d \in D$, un arco $e \in E$ y un *slot* $i \in S$ tales que el lado izquierdo de la desigualdad es menor que el lado derecho. Si d no utiliza e en u^*, la desigualdad se cumple, entonces d debe usar e, y, dado que es óptima, debe haber $v(d)$ *slots* contiguos usados por d en ese arco. Dado que u^* debe también satisfacer integralidad, entonces $u_{des} \in \{0,1\}$ para cada $s \in S$, y dado que los *slots* en el lado izquierdo son tomados distanciados por $v(d)$ *slots*, sólo uno de ellos puede estar encendido. Entonces el lado derecho debe usar al menos dos *slots* a causa de la integralidad, pero los *slots* en este lado también están a una distancia de $v(d)$ *slots* cada uno, por lo que sólo uno puede ser usado. □

Podemos también enunciar una familia simétrica denominada *symmContiguityIneqs*, la cual es válida para cada arco, a saber,

$$\sum_{\substack{s\in\{\bar{s}-i+1,\dots,\bar{s}\}:\\ s\ \equiv\ \bar{s}-i+1\ (v(d))}} u_{des} \ \geq \sum_{\substack{s\in\{\bar{s}-i+2,\dots,\bar{s}\}:\\ s-1\ \equiv\ \bar{s}-i+1\ (v(d))}} u_{des} \qquad \forall d\in D,\ \forall e\in E,\ \forall i\in S, \tag{3.35}$$

y debido al Teorema 3.1.1 podemos establecer el siguiente corolario.

Corolario 3.3.1. *Las desigualdades (3.35) son cortes optimales para el modelo DSL-BF.*

Por medio de dos contraejemplos podemos ver que, si las restricciones del modelo DSL-BF valen, las desigualdades (3.34) no implican (3.35) ni viceversa. Consideremos una instancia del RSA con $\bar{s}=5$, y un grafo con un único arco e conectando el único par de nodos i y j, los cuales son el origen y el destino de la única demanda d. Si $v(d)=2$, el vector $u'_{de*}=(\frac{1}{4},\frac{1}{4},\frac{1}{2},\frac{1}{2},\frac{1}{2})$ satisface las desigualdades (3.34) junto con las restricciones del modelo DSL-BF, pero no (3.35). Definiendo un vector u'' invirtiendo u'_{de*} tenemos un contraejemplo para la afirmación opuesta.

Si para cada demanda d agrupamos los *slots* que están distanciados entre sí por su volumen, entonces dicha demanda d en una solución óptima debe usar la misma cantidad de *slots* de cada uno de estos grupos, dado que cada demanda d debe ser satisfecha mediante exactamente $v(d)$ *slots*. Las igualdades (3.32), además de eso, dicen que la cantidad de *slots* de cada grupo debe ser exactamente 1 para cada arco utilizado. Por lo tanto, podemos definir las siguientes igualdades, que llamamos *contiguityEqs*,

$$\sum_{\substack{s\in S:\\ s\ \equiv\ i\ (v(d))}} u_{des} \ = \sum_{\substack{s\in S:\\ s+1\ \equiv\ i\ (v(d))}} u_{des} \qquad \forall d\in D,\ \forall e\in E,\ \forall i\in\{1,\dots,v(d)\}. \tag{3.36}$$

Teorema 3.3.2. *Las igualdades (3.36) son válidas para cada solución óptima para el modelo DSL-BF.*

Demostración. Supongamos que tenemos una solución óptima u^* tal que viola al menos una de las igualdades (3.36). Esto implica que existen un arco $e\in E$, una demanda $d\in D$ con $v(d)>1$, y un índice $i'=2,\dots,v(d)$, tales que

$$\sum_{\substack{s\in S:\\ s\ \equiv\ i'\ (v(d))}} u^*_{des} \ \neq \sum_{\substack{s\in S:\\ s\ \equiv\ i'-1\ (v(d))}} u^*_{des}. \tag{3.37}$$

Si el arco e no es utilizado, la igualdad se cumple trivialmente. Por lo tanto d usa el arco e. Sean $L=\{s : s\in S,\ s\equiv i'(v(d))\}$ y $R=\{s : s\in S,\ s\equiv i'-1(v(d))\}$ los conjuntos de los *slots* asociados a las variables del lado izquierdo y derecho de (3.37), respectivamente. Dado que cada par de *slots* en L se encuentra a una distancia mayor o igual a $v(d)$, entonces la demanda d puede usar a lo sumo uno de ellos en e para que u^* sea óptima. Lo mismo ocurre con el conjunto R. Supongamos $u^*_{des'}=1$, con $s'\in L$, entonces, dado que d debe usar exactamente $v(d)$ *slots* contiguos en e, d debe también usar el *slot* $s'+1$ ó $s'-1\in R$. Si $u^*_{des'}=u^*_{de,s'-1}=1$, entonces la igualdad se cumple, por lo tanto asumamos $u^*_{des'}=u^*_{de,s'+1}=1$. Pero, dado que $s'-1<s'<s'+1<s'+v(d)-1$, con la diferencia $(s'+v(d)-1)-(s'-1)=v(d)$, y ambos *slots*, *i.e.*, $s'+v(d)-1$ y $s'-1$, perteneciendo a R, entonces d debe usar uno de ellos, por lo que el lado izquierdo y el derecho de (3.37) valen 1. Dada la simetría podemos seguir el mismo razonamiento al asumir $u^*_{des'}=1$ para algún $s'\in R$. Por lo tanto, no existe una tal solución óptima u^* y las ecuaciones (3.36) son igualdades optimales. □

Ya hemos demostrado en el Teorema 1.4.1 que encontrar el conjunto de arcos que una demanda dada utiliza en alguna solución factible es $\mathcal{NP}$-hard. Encontrar los arcos que una demanda utiliza en todas las soluciones factibles pareciera también ser $\mathcal{NP}$-hard para el caso general. Sin embargo podemos intentar obtener algunos subconjuntos útiles. Siendo que $P(G, s, t)$ nos brinda todos los caminos simples que conectan s con t, podemos definir el siguiente concepto.

Definición 3.3.1. *Llamamos* corte (s,t)-minimal *a un subconjunto $C \subseteq E$ tal que $|C \cap P| = 1$ para cada $P \in P(G, s, t)$.*

Proposición 3.3.1. *Para cada demanda $d \in D$ y cada corte $(s(d), t(d))$-minimal S_c, el* lightpath *asociado a d en cualquier solución óptima contiene exactamente un arco del conjunto S_c.*

Demostración. Una solución óptima u^* debe satisfacer cada demanda utilizando un camino sin ciclos ni bifurcaciones, caso contrario u^* no sería óptima, y, por definición, cada camino entre el origen y el destino de la demanda d tiene exactamente un arco que pertenece a S_c, en particular los caminos usados en el *lightpath* que satisface d. □

Teorema 3.3.3. *Sea SC_d un conjunto de cortes $(s(d), t(d))$-minimales para una demanda $d \in D$ en un digrafo G, entonces las desigualdades*

$$\sum_{e \in C} \sum_{\substack{s \in S: \\ s \equiv i \ (v(d))}} u_{des} \ = \ 1 \qquad \forall d \in D, \ \forall i \in \{1, \ldots, v(d)\}, \ \forall C \in SC_d, \tag{3.38}$$

son válidas para cada solución óptima de $RSA(G, D, \bar{s})$.

Demostración. Consideremos una instancia del RSA, una demanda d y un corte $C \in SC_d$. Dada una solución óptima u^*, debido a la Proposición 3.3.1 existe un único arco $e' \in C$ tal que e' pertenece al camino del *lightpath* que satisface la demanda d en u^*. Luego $\sum_{s \in S} u^*_{de's} = v(d)$. Más aún, u^* satisface (3.32) para e'. Por otro lado, $\sum_{s \in S} u^*_{des} = 0$ para cada $e \neq e'$ en C por definición, entonces u^* satisface la igualdad (3.38). □

Corolario 3.3.2. *Las familias (3.34), (3.36), y (3.38) son suficientes para asegurar que en cualquier solución que satisface el modelo DSL-BF, cada demanda $d \in D$ usa* 0 *ó exactamente $v(d)$* slots *consecutivos en cada arco $e \in E$.*

Los casos especiales de (3.38) dados por $C = \delta^+(s(d))$ y $C = \delta^-(t(d))$ son denominados *ppalSlotsFromSrc* y *ppalSlotsToDst*, respectivamente. Notar que cuando usamos los cortes (3.36) no tenemos necesidad de establecer la familia *ppalSlotsFromSrc* para cada valor de j sino que alcanza con decirlo para uno solo de ellos. También separamos la subfamilia obtenida al tomar un único subgrupo de *slots*, por ejemplo fijando $i = 1$. Denominamos a cada una de estas familias como *onePpalSlotsFromSrc*.

3.3.2. *Slots* centrales

Tomando cada demanda d que cumpla $2v(d) > \bar{s}$, podemos obtener otra familia de igualdades válidas. En estos casos, los $2v(d) - \bar{s}$ *slots* centrales de algún arco en $C \in SC_d$ deben ser usados por d, a saber,

$$\sum_{e \in C} \sum_{s \in Y} u_{des} = |Y| \qquad Y = \{\bar{s} - v(d) + 1, \ldots, v(d)\}, \ \forall C \in SC_d. \tag{3.39}$$

Denominamos esta familia como *centralSlots*, y llamamos *centralSlotsFromSrc* al caso especial dado por $C = \delta^+(s(d))$. Para los experimentos, implementamos únicamente esta última familia.

Si nos restringimos a las demandas d tales que $2v(d) > \bar{s}$, por medio de un contraejemplo podemos demostrar que las igualdades (3.38) no son implicadas por el modelo dado por (3.2)-(3.7) y las igualdades (3.39). Consideremos una instancia del RSA con una demanda d con $\frac{\bar{s}}{2} < v(d) < \bar{s}$. Sea u' una solución factible tal que d es satisfecha utilizando $\bar{s}$ *slots*. Luego podemos ver que (3.39) son satisfechas por u' para cada $C \in SC_d$, pero u' viola (3.38) por ejemplo tomando $C = \delta^+(s(d))$ y $i = 1$ dado que $u_{de1} + u_{de,1+v(d)} = 2 > 1$.

Teorema 3.3.4. *Las igualdades (3.39) son cortes optimales para el modelo DSL-BF.*

Demostración. Consideremos una instancia del RSA y asumamos una solución óptima u^* tal que u^* viola algunas desigualdades (3.39). Deben existir una demanda $d \in D$ tal que $2v(d) > \bar{s}$, y un conjunto no vacío $C \in SC_d$, con $\sum_{e \in C} \sum_{s \in Y} u^*_{des} \neq |Y|$, caso contrario las igualdades (3.39) se cumplirían trivialmente. Dado que u^* es óptima, d debe ser satisfecha por un *lightpath* compuesto por un camino simple P y un canal Ch con $v(d)$ *slots* consecutivos. Por definición de C, debe existir un único arco $e \in C$ tal que $e \in P$, por lo que d debe usar $v(d)$ *slots* en ese arco y ningún otro *slot* en el resto de los arcos pertenecientes a C, *i.e.*, $\sum_{e' \in C \setminus \{e\}} u^*_{de's} = 0$. A causa de las restricciones de integralidad y dado que Y pertenece a cada intervalo posible de $v(d)$ *slots* consecutivos en $[1, \ldots, \bar{s}]$, entonces d debe usar cada *slot* de Y en el arco e, por lo tanto $\sum_{s \in Y} u^*_{des} = |Y|$, una contradicción. □

Podemos disgregar las igualdades (3.39) por *slot* obteniendo el conjunto de igualdades

$$\sum_{e \in C} u_{des} = 1 \qquad Y = \{\bar{s} - v(d) + 1, \ldots, v(d)\},\ \forall s \in Y,\ C \in SC_d, \tag{3.40}$$

denominadas *centralSlotsBySlots*, las cuales son también válidas para cada demanda d tal que $2v(d) > \bar{s}$. Análogamente, llamamos *centralSlotsFromSrcBySlot* al grupo de igualdades resultante de tomar $C = \delta^+(s(d))$. Para los experimentos, implementamos únicamente este último grupo de igualdades.

3.3.3. La posición fuerza el uso de algunos *slots*

Si una demanda d usa un *slot* $s < v(d)$, entonces los *slots* desde s a $v(d)$ deben ser también usados por d. Este hecho puede ser expresado por un conjunto de las desigualdades (3.35) tomando $i \in \{1, \ldots, v(d)\}$, o mediante las familias de desigualdades

$$\sum_{s'=s+1}^{v(d)} u_{des'} \geq (v(d) - s)\ u_{des} \qquad \begin{array}{r} \forall e \in E,\ \forall d \in D, \\ \forall s \in \{1, \ldots, v(d) - 1\} \end{array} \tag{3.41}$$

$$\sum_{s'=\bar{s}-v(d)+1}^{s-1} u_{des'} \geq (s - 1 - \bar{s} + v(d))\ u_{des} \qquad \begin{array}{r} \forall e \in E,\ \forall d \in D, \\ \forall s \in \{\bar{s} - v(d) + 2, \ldots, \bar{s}\} \end{array} \tag{3.42}$$

que llamamos *lowPositionForces* y *highPositionForces*, respectivamente.

Teorema 3.3.5. *Las desigualdades (3.41) y (3.42) son implicadas por (3.34) y (3.35), respectivamente, junto con las restricciones (3.2)-(3.7).*

Demostración. Sea u' una solución fraccionaria para una instancia del modelo dado por las restricciones (3.2)-(3.7) y las desigualdades (3.35), sea d una demanda y e un arco. Las desigualdades (3.35) obligan a la variable u'_{des} relacionada con cada s mayor que $\bar{s} - v(d)$ a ser mayor o igual a $u'_{de,s+1}$, y por lo tanto u'_{des} debe ser mayor o igual que $u'_{des'}$ para cada $s' > s$. Sumando sobre cada una de estas desigualdades obtenemos (3.42). La demostración de la proposición simétrica es análoga. □

Con esto tenemos que las desigualdades (3.41) y (3.42) son cortes optimales para el modelo DSL-BF. Sin embargo, estas dos familias de desigualdades valen para cualquier solución entera, por lo tanto podemos enunciar el siguiente resultado.

Teorema 3.3.6. *Las desigualdades (3.41) y (3.42) son válidas para el modelo DSL-BF.*

Demostración. Sea u' una solución factible y supongamos que viola al menos una de las desigualdades (3.41). Por lo tanto tiene que existir un arco $e \in E$, una demanda $d \in D$ y un *slot* $s < v(d)$ tal que

$$\sum_{s'=s+1}^{v(d)} u'_{des'} > (v(d) - s)\, u'_{des}.$$

Pero esto implica que el lado derecho es mayor a cero, y por integralidad $u'_{des} = 1$, *i.e.*, el *slot* s es utilizado por la demanda d en dicho arco. Asimismo, por contigüidad, $u'_{de,v(d)} = 1$ debido a que pertenece a todos los intervalos de al menos $v(d)$ *slots* consecutivos que incluyen a s, y por lo tanto $u'_{des'} = 1$ para todo $s' \in \{s+1, \ldots, v(d)\}$. Por otro lado tenemos que la cantidad de términos del lado izquierdo, es exactamente $v(d) - s$, por lo que para violar la desigualdad debe cumplirse que $u'_{des'} = 0$ para algún $s' \in \{s+1, \ldots, v(d)\}$. Absurdo. De forma análoga es posible demostrar la validez de las desigualdades (3.42). □

3.3.4. Los *slots* lejanos no se usan

Si un *slot* s es usado por una demanda d y d usa exactamente $v(d)$ *slots*, entonces d debería usar únicamente *slots* ubicados a una distancia de a lo sumo $v(d)$ *slots* de s. Si definimos $S'_{sd} = \{1, \ldots, s - v(d)\} \cup \{s + v(d), \ldots, \bar{s}\}$, podemos forzar este hecho con las desigualdades

$$\sum_{s' \in S'_{sd}} u_{des'} \leq M(1 - u_{des}) \qquad \forall e \in E,\, \forall d \in D,\, \forall s \in S,\, M = \text{mín}\{|S'_{sd}|, v(d)\}, \tag{3.43}$$

que denominamos *farSlotsOff*, y las cuales están representadas en la Figura 3.9.

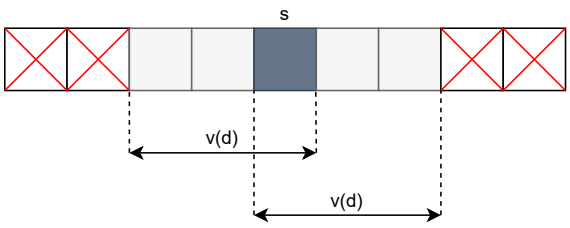


Fig. 3.9: Los cortes optimales denominados *farSlotsOff* dicen que si un *slot* s es utilizado por una demanda d, ésta no puede usar ningún *slot* a una distancia mayor que su volumen $v(d)$ de s.

Por medio de un contraejemplo podemos demostrar que los cortes optimales (3.43) no son implicados por las restricciones del modelo (3.2)-(3.7) junto con (3.19), (3.36), y (3.38) tomando $C = \delta^+(s(d))$. Consideremos una instancia del RSA con $\bar{s} = 12$ y una única demanda d con $v(d) = 2$. Entonces M debe ser igual a $v(d)$. Sea $u' \in \mathbb{R}^{|D||E|\bar{s}}$ un vector tal que $u'_{de*} = (\frac{1}{4}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}, 0, \frac{1}{8}, \frac{1}{2}, \frac{1}{4}, 0, \frac{1}{8}, \frac{1}{8}, \frac{1}{8})$ para cada e a lo largo de un camino simple entre $s(d)$ y $t(d)$, y $u'_{des} = 0$ para los arcos y *slots* restantes. El vector u' trivialmente satisface las restricciones de flujo y no solapamiento y podemos ver que no viola las restricciones de contigüidad. Asimismo, esta solución fraccionaria no viola las desigualdades (3.19) ni las (3.20), porque la suma de cada variable perteneciente a los arcos que salen de $s(d)$ es exactamente $v(d)$ y para los otros vértices, esta suma es menor o igual a $v(d)$. Esta solución también satisface las igualdades (3.36) y las (3.38) con $C = \delta^+(s(d))$, dado que las sumas de los dos conjuntos posibles de *slots* son ambas iguales a 1. Sin embargo u' viola las desigualdades (3.43) en cada arco del camino tomando $s = 7$, *i.e.*, el único *slot* con valor $\frac{1}{2}$.

Teorema 3.3.7. *Las igualdades (3.43) son cortes optimales para el modelo DSL-BF.*

Demostración. Sea u^* una solución óptima para una instancia del RSA tal que existe un *slot* $s \in S$, un arco $e \in E$, y una demanda $d \in D$ que violan al menos una de las desigualdades (3.43). Sin pérdida de generalidad podemos asumir $v(d) \leq s \leq \bar{s} - v(d) + 1$ y extender los siguientes resultados a los casos borde. Si $u^*_{des} = 0$, como $M \geq |S'_{sd}|$ tendríamos $\sum_{s' \in S'_{sd}} u^*_{des'} > |S'_{sd}|$, lo cual no puede ocurrir dado que $u^*_{des'} \in \{0, 1\}$ para cada $s' \in S$ debido a las restricciones de integralidad. Consecuentemente, u^*_{des} tiene que ser igual a 1. Pero esto, sumado a las restricciones de contigüidad, implica que otros $v(d) - 1$ *slots* deben ser usados alrededor de s, *i.e.*,

$$\sum_{s'=s-v(d)+1}^{s+v(d)} u^*_{des'} > v(d),$$

y como $S'_{sd} = S - \{s - v(d) + 1, \ldots, s + v(d)\}$, por lo tanto $\sum_{s' \in S'_{sd}} u^*_{des'} = 0$ para ser óptima. □

3.3.5. Restricciones de contigüidad simétricas

También consideramos las desigualdades obtenidas al aplicar el Teorema 3.1.1 sobre las restricciones del modelo (3.7), las cuales denominamos *symmetricalBFContiguity*, a saber,

$$\sum_{s'=1}^{v(d)} u_{des'} \geq v(d)\ u_{de1} \qquad \forall d \in D, \forall e \in E \tag{3.44}$$

$$\sum_{s'=s}^{f} u_{des'} \geq v(d)(u_{des} - u_{de,s-1}) \qquad \begin{array}{r} \forall d \in D, \forall e \in E, \forall s \in \{2, \ldots, \bar{s}\}, \\ f = \text{mín}\{\bar{s}, s + v(d) - 1\}. \end{array} \tag{3.45}$$

Debido al Teorema 3.1.1 y a la validez de las restricciones (3.7), es posible enunciar el siguiente resultado.

Teorema 3.3.8. *Las familias de desigualdades (3.44) y (3.45) son válidas para el modelo DSL-BF.*

Podemos mostrar que (3.44) y (3.45) no son implicadas por las restricciones del modelo, en particular por (3.7), presentando un contraejemplo. Consideremos una instancia del RSA con una única demanda d de volumen $v(d) = 2$ y tal que el arco $e = s(d)t(d)$ pertenece al grafo. Sea u' un vector tal que todos sus elementos son cero, excepto para $u'_{de*} = (0, 1, \frac{1}{2}, \frac{1}{2})$. El vector u' satisface todas las restricciones, en particular (3.7), pero no todas las desigualdades (3.45). Tomando $s = 1$ tenemos que $f = 3$ y por lo tanto $u'_{de2} + u'_{de3} = \frac{3}{2} < 2 = 2(u'_{de2} - u'_{de1})$. Generando otro vector u'' invirtiendo u', *i.e.*, $u''_{de*} = (\frac{1}{2}, \frac{1}{2}, 1, 0)$, podemos demostrar la no implicación opuesta.

3.3.6. Contigüidad del modelo ASCC

La variación del modelo DSL-BF propuesta, *i.e.*, la formulación DSL-ASCC, reemplaza las restricciones (3.6) y (3.7), las cuales aseguran la contigüidad de los *slots* y la satisfactibilidad de las demandas, por las siguientes restricciones alternativas

$$v(d)\ u_{des} \leq \sum_{s' \in S} u_{des'} \qquad \forall d \in D,\ \forall e \in E,\ \forall s \in S$$

$$u_{des_1} + u_{des_2} \leq u_{de(s_1+1)} + 1 \qquad \forall d \in D,\ \forall e \in E,\ \forall s_1, s_2 \in S,\ s_2 > s_1. \tag{3.46}$$

En particular, podemos utilizar (3.46) como cortes optimales para el modelo DSL-BF. Denominamos a éstos como *contiguityASCC*, y con un contraejemplo podemos demostrar que no son implicados por las restricciones del modelo (3.2)-(3.7), las desigualdades de flujo (3.19) y las desigualdades *symmetricalBFContiguity*. Asumamos una instancia del RSA con una única demanda d tal que $v(d) = 5$ y con un arco e conectando $s(d)$ con $t(d)$. Sea u' un vector tal que $u'_{de*} = (0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1, \frac{3}{4}, 1, \frac{3}{4}, \frac{1}{2}, 0, 0)$, y $u'_{de's} = 0$ para cada otro arco y *slot*. El vector u' trivialmente satisface las restricciones de flujo y las de no solapamiento. También satisface las desigualdades (3.20) dado que todos sus elementos suman exactamente 5. Es sencillo verificar que satisface las restricciones de contigüidad y sus simétricas (3.44)-(3.45), pero $u'_{de5} - u'_{de6} + u'_{de7} > 1$, violando al menos una de las desigualdades (3.46).

3.4. Desigualdades basadas en el no solapamiento

En esta sección presentamos varias desigualdades e igualdades válidas y cortes optimales basados principalmente de la restricción de no solapamiento, es decir, que dos demandas no pueden usar el mismo *slot* en el mismo arco.

3.4.1. No disponibles debido a la contigüidad

Las restricciones de contigüidad y no solapamiento implican que si una demanda d utiliza dos *slots* s_1 y s_3 con $s_1 < s_3 - 1$, entonces ninguna otra demanda puede usar ningún *slot* $s_2 \in \{s_1 + 1, \ldots, s_3 - 1\}$. Las siguientes desigualdades válidas capturan este hecho:

$$u_{d_2es_1} + u_{d_1es_2} + u_{d_2es_3} \leq 2 \qquad \forall e \in E,\ \forall d_1 \neq d_2 \in D,\ \forall s_1 < s_2 < s_3 \in S. \tag{3.47}$$

Esta familia de cortes es denominada *nonOver* y está representada en la Figura 3.10. Notar que no necesitamos pedir que s_2 sea igual a los otros dos *slots* dado que esos casos son contemplados directamente por las restricciones (3.5). Supongamos que tenemos una

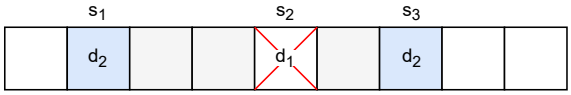


Fig. 3.10: Desigualdades de no solapamiento con dos demandas en un arco.

instancia del RSA tal que el conjunto D está compuesto por dos demandas d_1 y d_2 con volúmenes 1 y 3, respectivamente, y sus nodos de origen y destino forman el único par de nodos del grafo. Por simplicidad asumamos que hay un solo arco e que permite satisfacer a ambas demandas, y fijemos $\bar{s} = 6$. Sea u' un vector tal que $u'_{d_1es} = 1$ para $s = 4$, y 0 para cualquier otro *slot*, mientras que $u'_{d_2e*} = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0, 1, \frac{1}{2})$. Podemos verificar que u' es una solución fraccionaria del modelo. Trivialmente satisface las restricciones de flujo y no solapamiento para las dos demandas y podemos ver que cumple con cada restricción de contigüidad. Sin embargo u' no satisface ninguna desigualdad (3.47) para d_1 y d_2 fijando $s_2 = 4$ y $s_3 = 5$. Por lo tanto, las desigualdades (3.47) no son implicadas por las restricciones del modelo DSL-BF.

Considerando la suma de las demandas

Podemos reforzar las desigualdades anteriores utilizando las restricciones (3.5) considerando la suma de las demandas que utilizan el *slot* s en lugar de tomar una única demanda. Denominamos estas desigualdades como *nonOverBySum*, a saber,

$$u_{des_1} + \sum_{d' \in D \setminus \{d\}} u_{d'es_2} + u_{des_3} \leq 2 \qquad \forall d \in D, \forall s_1, s_2, s_3 \in S,\ s_1 < s_2 < s_3. \tag{3.48}$$

Teorema 3.4.1. *Las desigualdades (3.48) son cortes optimales para el modelo DSL-BF.*

Demostración. Supongamos que existe una solución óptima u^* que viola alguna desigualdad (3.48), *i.e.*, existen $d \in D$, $e \in E$, y $1 \leq s_1 < s_2 < s_3 \leq \bar{s}$ tales que $u^*_{des_1} + \sum_{d' \in D \setminus \{d\}} u^*_{d'es_2} + u^*_{des_3} > 2$. La solución u^* debe satisfacer las restricciones de integralidad para cada variable binaria, y por las restricciones de no solapamiento $\sum_{d' \in D \setminus \{d\}} u^*_{d'es_2} \in \{0, 1\}$, por lo tanto debe existir una demanda particular $d' \neq d$ tal que $u^*_{des_1} + u^*_{d'es_2} + u^*_{des_3} > 2$ y las tres variables deben ser iguales a 1. Dado que u^* es óptima, cada demanda debe usar exactamente $v(d)$ *slots* consecutivos en e, por lo que si $u^*_{des_3} = 1$ entonces debe existir un *slot* $s' \leq \bar{s} + 1$ tal que (3.7) es satisfecha por igualdad, *i.e.*, $u^*_{des'} = 1$, $u^*_{de,s'+1} = 0$, y $\sum_{s \in A} u^*_{des} = v(d)$ con $A = \{s - v(d) + 1, s\}$, y como $|A| = v(d)$, por lo tanto $u^*_{des} = 1$ para cada $s \in A$. Además, dado que u^* es óptima, d no puede usar ningún *slot* no perteneciente a A en e, por lo tanto, $s_1, s_3 \in A$ y también $s_2 \in A$ porque este *slot* está entre s_1 y s_3, por lo que s_2 debe ser usado también por d. Pero, como s_2 ya es usado por d', estaríamos violando las restricciones de no solapamiento (3.5). □

El siguiente resultado es una consecuencia de que las desigualdades (3.47) sean dominadas por las desigualdades (3.48).

Teorema 3.4.2. *Las desigualdades (3.47) son satisfechas por cada punto de la relajación lineal del modelo DSL-BF que satisface las desigualdades (3.48).*

Dado que (3.48) son cortes optimales que no cortan todas las soluciones óptimas, y cada punto de la relajación lineal que satisface (3.48) también satisface (3.47), entonces el siguiente resultado también es válido.

Corolario 3.4.1. *Las desigualdades (3.47) son cortes optimales para el modelo DSL-BF.*

Asimismo, dado que existe un vector u' en la relajación lineal del modelo que viola alguna desigualdad (3.47) y dado que u' tiene que violar también alguna desigualdad (3.48), tenemos el siguiente resultado.

Corolario 3.4.2. *Las desigualdades (3.48) no son implicadas por las restricciones del modelo.*

Más aún, sea una instancia con dos demandas d y d' tal que $\bar{s} \geq 2\ v(d)+1$ y $v(d') = 1$. Supongamos que existen dos caminos p y p' que comparten un arco e tal que p y p' son utilizados en los *lightpaths* que satisfacen d y d' respectivamente. La solución que asigna el slot $s_2 = v(d) + 1$ a lo largo del camino p' a la demanda d' y todos los *slots* de S excepto s_2 en los arcos pertenecientes al camino p los asigna a d satisface todas las restricciones del modelo DSL-BF pero viola las desigualdades (3.47) y (3.48) en e.

Tomando dos *slots* consecutivos

Como el número de desigualdades en las familias anteriores puede ser muy grande para separar, en nuestros experimentos también tomamos los casos particulares con $s_3 = s_2+1$, permitiendo a s_1 y s_2 estar en los rangos $[1, \bar{s} - 2]$ y $[s_1 + 1, \bar{s} - 1]$, respectivamente. Denominamos a estas subfamilias como *nonOverFixNextLow* y *nonOverFixNextHi* cuando reducen (3.47) y como *nonOverFixNextLowBySum* y *nonOverFixNextHiBySum* cuando son un subconjunto de (3.48), respectivamente.

3.4.2. K demandas que superan la capacidad del arco

Sea $D' \subseteq D$ un conjunto minimal de demandas tal que la suma de sus volúmenes es mayor que la capacidad de slots, *i.e.*, un conjunto D' tal que $\sum_{d\in D'} v(d) > \bar{s}$ y para cada $d' \in D'$, $\sum_{d\in D'\setminus\{d'\}} v(d) \leq \bar{s}$. Por lo tanto, para cada arco $e \in E$, al menos una demanda de D' no puede usar e. Asumiendo que cada demanda $d \in D'$ usa exactamente $v(d)$ *slots*, este hecho puede ser capturado por los cortes optimales

$$\sum_{s'\in S} u_{des'} \leq v(d) \sum_{d'\in D'\setminus\{d\}} \Big(v(d') - \sum_{s'\in S} u_{d'es'}\Big) \qquad \forall d \in D',\ \forall e \in E. \tag{3.49}$$

Si asumimos las restricciones (3.2)-(3.7) y la familia (3.28), entonces las desigualdades (3.48) no implican (3.49). Para demostrar esto vamos a construir un contraejemplo. Asumamos una instancia del RSA con $\bar{s} = 2$ y dos demandas d_1 y d_2 tales que comparten origen y destino, con $v(d_1) = 2$ y $v(d_2) = 1$. Sean P_1 y P_2 dos caminos simples disjuntos (*i.e.*, sin ciclos) que conectan el origen con el destino. Sea $u' \in \mathbb{R}^{2\times|E|\times 2}$ tal que $u'_{d_1e*} = (\frac{1}{2}, \frac{1}{2})$ para cada $e \in P_1 \cup P_2$, $u'_{d_2e*} = (\frac{1}{2}, \frac{1}{2})$ para cada $e \in P_1$ y $u'_{des} = 0$ para los otros valores de d, e, y s. Ésta claramente no es una solución factible sino una solución fraccionaria, no obstante trivialmente satisface las restricciones de flujo, las restricciones de no solapamiento y las de contigüidad. Podemos también ver que ambas demandas son satisfechas por exactamente sus volúmenes, d_2 utilizando P_1 y d_1 a través de ambos caminos, satisfaciendo las restricciones (3.4). Podemos también probar que esta solución fraccionaria satisface todas las desigualdades de la familia (3.28). El vector u' trivialmente satisface las desigualdades (3.48) (no hay ninguna). Pero, dado que la suma de los volúmenes de ambas demandas

excede la capacidad $\bar{s}$, u' viola las desigualdades (3.49) para cada e todo a lo largo del camino P_1, tomando d_1 como d.

La proposición opuesta es también verdadera, es decir, si las restricciones (3.2)-(3.7) y la familia (3.28) valen, entonces las desigualdades (3.49) no implican (3.48). Para demostrar esto vamos a presentar otro contraejemplo. Asumamos nuevamente una instancia para el RSA con $\bar{s} = 6$, dos demandas d_1 y d_2 con $v(d_1) = 2$ y $v(d_2) = 1$, y sea u' una solución fraccionaria para esa instancia tal que los caminos simples utilizados para satisfacer ambas demandas compartan al menos un arco e'. Si $u_{d_1e*} = (\frac{1}{8}, \frac{1}{8}, 0, 1, \frac{1}{2}, \frac{1}{4})$ y $u_{d_2e*} = (0, 0, 1, 0, 0, 0)$ para cada arco e perteneciente a sus respectivos caminos, en particular para e', entonces u' satisface todas las restricciones del modelo, la familia (3.28) y (3.49) pero viola al menos una desigualdad de (3.48), porque en el arco e' tenemos $u'_{d_1e'2} + u'_{d_2e'3} + u'_{d_1e'4} = \frac{17}{8} > 2$.

Teorema 3.4.3. *Las desigualdades (3.49) son cortes optimales para el modelo DSL-BF.*

Demostración. Sea u^* una solución óptima para una instancia del RSA, y asumamos que u^* viola al menos una de las desigualdades (3.49). Entonces deben existir un conjunto $D' \subseteq D$, una demanda $d \in D'$, y un arco $e \in E$ tales que u^* no cumple la desigualdad (3.49) asociada. Dado que u^* es óptima, entonces cada demanda $d' \in D$ debe usar exactamente 0 ó $v(d')$ *slots* en e, y, como $v(d') \in \mathbb{N}$, la suma del lado derecho de la desigualdad debe ser $k \times v(d)$ con $k \in \mathbb{N}_0$. Pero, dado que el lado izquierdo debe ser igual a 0 ó $v(d)$ a causa de la optimalidad de u^*, el único modo de violar la desigualdad es con el lado derecho igualado a 0, *i.e.*, cada demanda $d' \in D'$, incluída d, utilizando exactamente $v(d')$ *slots*, lo cual no es posible por definición de D'. □

Considerando la suma

Podemos considerar una familia alternativa de cortes optimales, denominada *KDemandsDoNotExceedSBySum*, forzando a la suma sobre todos los *slots* usados por las demandas en D' a ser menor o igual que la suma de sus volúmenes restándole el menor, a saber

$$\sum_{s \in S} \sum_{d \in D'} u_{des} \leq \sum_{d \in D'} v(d) - \min_{d \in D'} v(d) \qquad \forall e \in E. \tag{3.50}$$

Si asumimos las restricciones (3.2)-(3.7), entonces las desigualdades (3.49) no implican (3.50). Para demostrar dicha afirmación consideremos una instancia del RSA en un grafo G. Sean d_1 y d_2 dos demandas que comparten volumen, origen y destino, asumamos que existen dos caminos disjuntos P_1 y P_2 en G conectando $s(d_1)$ con $t(d_1)$, y asumamos que ninguna otra demanda necesita utilizar ningún arco $e' \in P_1 \cup P_2$. Sea $\bar{s} = 3$ y sea 2 el volumen requerido por cada demanda, de modo tal que ambas demandas no entren juntas en ningún arco. Construyamos una solución fraccionaria u' como sigue. Definimos $u'_{d_1e*} = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4})$ y $u'_{d_2e*} = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ para cada $e \in P_1$, y $u'_{d_1e*} = (\frac{1}{4}, \frac{1}{2}, \frac{1}{2})$ y $u'_{d_2e*} = (\frac{1}{4}, \frac{1}{4}, 0)$ para cada $e \in P_2$, y asignemos *lightpaths* para satisfacer todas las otras demandas sin utilizar P_1 ni P_2. Podemos ver que u' satisface las restricciones del modelo y (3.49) pero viola (3.50).

Teorema 3.4.4. *Las desigualdades (3.50) son cortes optimales para el modelo DSL-BF.*

Demostración. Asumamos que existe una solución óptima u^* para una instancia del RSA tal que para un arco $e \in E$ y un conjunto minimal $D' \in D$ al menos una desigualdad de la familia (3.50) es violada. Dado que u^* es óptima, cada $d' \in D'$ debe usar 0 ó $v(d')$ *slots* contiguos en e, pero si una demanda d' no utiliza e, la suma de los *slots* usados por las otras será menor o igual al lado derecho de la desigualdad, porque $v(d') \geq \text{mín}_{d \in D'} v(d)$. Por lo tanto todas las demandas en D' usan e, pero debido a la restricción de no solapamiento, cada *slot* en e puede ser utilizado por una única demanda, y por definición de D' necesitaríamos más de $\bar{s}$ *slots*. □

En nuestros experimentos computacionales, hemos separado el caso $|D'| = 2$ tanto en (3.49) como en (3.50), los cuales mostraron un mejor rendimiento, obteniendo las familias *twoDemandsDoNotExceedS* y *twoDemandsDoNotExceedSBySum*, respectivamente.

3.4.3. No cabe a causa de la posición

Las siguientes familias de desigualdades están motivadas por el hecho de que la posición del *slot* utilizado por una demanda a veces obliga a otra demanda a no poder usar un conjunto particular de *slots* porque su volumen no encaja allí sin una superposición. Esta idea puede expresarse de muchas formas y generalizarse para una estructura grande que genera varias familias de desigualdades.

La razón de la definición de una estructura es no sólo establecer el comportamiento de algunas demandas en un arco en particular, sino también poder extrapolar este comportamiento a un conjunto mayor de arcos. Específicamente, si una demanda tiene prohibido usar un rango de *slots* en un arco, queremos decir que tampoco puede usar estos *slots* en algunos otros. Tal estructura está compuesta por un arco e' y todos los arcos que cumplen que cualquier ruta simple que use primero un arco no perteneciente a la estructura y luego un arco perteneciente a la estructura (quizás el mismo e'), tiene que pasar por e'.

La Figura 3.11 representa un ejemplo donde sería util tener dicha estructura caracterizada. La demanda coloreada en rojo utiliza dos *slots* del arco v_1v_2 dejando un solo *slot* libre, por lo que podemos afirmar que, dado que no hay otro modo de acceder a los arcos v_2v_3, v_2v_5, y v_2v_8, ninguna otra demanda que requiera dos o más *slots* puede utilizar dichos arcos. De hecho, tanto la demanda verde como la amarilla de la figura usan caminos más largos a causa de esta prohibición.

La estructura particular en este ejemplo podría ser el conjunto de arcos v_1v_2, v_2v_3, v_2v_5, y v_2v_8, y puede ser formalmente definida como sigue.

Definición 3.4.1. *Sean $e' \in E$ y $E' \subset E \setminus \{e'\}$. El conjunto de arcos $PP = E' \cup \{e'\}$ es un* camino privado entrante *si y sólo si $e' \in P$ para cada camino $P = [e_1, e_2, ..., e_k] \subseteq E$ que contiene dos arcos e_i y e_j, $1 \leq i < j \leq k$, con $e_i \notin PP$ y $e_j \in PP$.*

Análogamente, podemos definir una estructura similar cuando los caminos van en el sentido contrario, es decir, desde adentro hacia afuera del camino privado.

Definición 3.4.2. *Sean $e' \in E$ y $E' \subset E \setminus \{e'\}$. El conjunto de arcos $PP = E' \cup \{e'\}$ es un* camino privado saliente *si y sólo si $e' \in P$ para cada camino $P = [e_1, e_2, ..., e_k] \subseteq E$ que contiene dos arcos e_i y e_j, $1 \leq i < j \leq k$, con $e_i \in PP$ y $e_j \notin PP$.*

En ambos casos de caminos privados denominamos al arco e' como el *arco maestro*. La Figura 3.12 ejemplifica estas dos estructuras. Cuando cada camino que conecta un arco no perteneciente a PP con un arco del interior de PP debe ir a través del arco maestro,

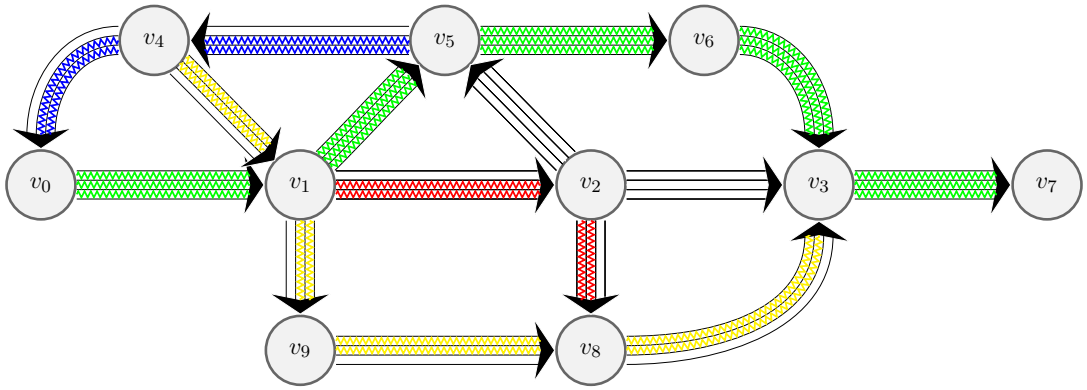
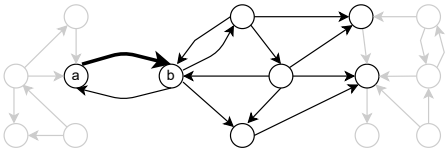
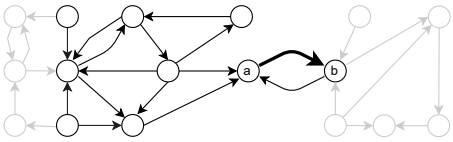


Fig. 3.11: Si una demanda roja utiliza el arco v_1v_2, entonces ninguna otra demanda que requiera dos o más *slots*, por ejemplo la verde o la amarilla, puede usar ningún *slot* en el conjunto de arcos v_1v_2, v_2v_3, v_2v_5, y v_2v_8.

tenemos un camino privado entrante, como se muestra en el ejemplo de la Figura 3.12a. Análogamente, si cada camino que conecta un arco del interior de PP con un arco fuera de PP debe usar el arco maestro, entonces tenemos un camino privado saliente, como es mostrado en la Figura 3.12b.



(a) Los arcos negros forman un camino privado entrante PP, porque cada camino que conecta un arco fuera de PP con un arco dentro de PP debe usar el arco maestro ab.



(b) Los arcos negros forman un camino privado saliente PP, dado que cada camino que conecta un arco $e \in PP$ con un arco $e' \notin PP$ debe usar el arco maestro ab.

Fig. 3.12: Ejemplos de caminos privados.

Definición 3.4.3. *Dado un camino privado entrante PP con ij como su arco maestro, denominamos como $D(PP)$ al conjunto compuesto por cada demanda $d \in D$ tal que*

- $s(d) \neq j$, *y*
- *si $s(d) \in V(PP) \setminus \{i\}$ entonces $t(d) \in V(PP) \setminus \{i\}$ y no existe un camino con todos sus arcos dentro de $PP \setminus \{ij\}$ conectando $s(d)$ con $t(d)$.*

Definimos lo análogo cuando PP es un camino privado saliente.

Esto es, para cada camino P que satisface una demanda $d \in D(PP)$, si P usa algún arco $e \in PP$, debe también utilizar el arco maestro de PP, *i.e.*, el arco ij.

Lema 3.4.1. *Dado un camino privado PP con arco maestro ij, y una demanda $d \in D(PP)$, si d no puede usar un* slot *s en ij, no puede usar s en ningún arco $e \in PP$ sin utilizar también un ciclo espurio.*

Demostración. Sin pérdida de generalidad, asumamos que PP es un camino privado entrante. Supongamos que existe tal demanda $d \in D(PP)$. Si $s(d) \notin V(PP)$, por definición de PP no existe un camino $P \subseteq E \setminus \{e'\}$ que use primero un arco $e_1 \notin PP$ y luego un arco $e_2 \in PP$, y si $s(d) \in V(PP)$, por definición de $D(PP)$, entonces $t(d) \in V(PP)$ y no existe un camino $P \subseteq D(PP) \setminus \{e'\}$ que conecte $s(d)$ con $t(d)$. Por lo tanto, en ambos casos, si d usa un arco perteneciente a PP dentro de un camino P, e' debe también pertenecer a P y, a causa de las restricciones de continuidad, si d usa el *slot* s en un arco perteneciente a P, d debe usar este *slot* todo a lo largo del camino P, en particular en el arco maestro e'. □

Un único límite

En esta sección nos estamos limitando a soluciones sin ciclos espurios, por lo que las desigualdades presentadas son cortes optimales. Dado un camino privado PP con su arco maestro e' y una demanda $d_1 \in D(PP)$, si un *slot* $s \leq v(d_1)$ es usado en e' por otra demanda d_2, entonces d_1 no puede usar ninguno de los primeros s *slots* en ningún arco de PP sin utilizar también un ciclo espurio. Más aún, d_1 no puede usar ninguno de los primeros $v(d_2)$ *slots*. Podemos expresar este hecho individualmente para cada uno de esos primeros *slots* y demandas como se muestra en la Figura 3.13, a saber,

$$u_{d_1 e s_1} + u_{d_2 e' s} \leq 1 \qquad \begin{array}{r} \forall e \in PP,\ \forall d_2 \in D,\ \forall d_1 \in D(PP),\ d_1 \neq d_2, \\ \forall s \in \{2, \ldots, v(d_1)\},\ \forall s_1 \leq \text{máx}\{s, v(d_2)\}, \end{array} \tag{3.51}$$

o podemos sumar sobre cada demanda en $D \setminus \{d_1\}$ en lugar de tomar cada demanda particular d_2; esto es,

$$u_{d_1 e s_1} + \sum_{d' \neq d_1} u_{d' e' s} \leq 1 \qquad \begin{array}{r} \forall e \in PP,\ \forall d_1 \in D(PP),\ \forall s \in \{2, \ldots, v(d_1)\}, \\ \forall s_1 \leq \text{máx}\{s, \text{mín}_{d' \in D \setminus \{d_1\}}(v(d'))\}. \end{array} \tag{3.52}$$

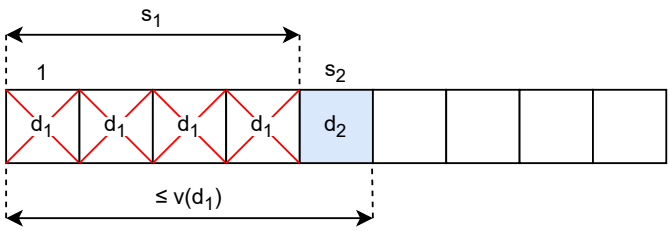


Fig. 3.13: El *slot* usado por la demanda d_2 es menor o igual a $v(d_1)$, por lo que d_1 no puede usar ningún *slot* menor que éste.

Teorema 3.4.5. *Las desigualdades (3.52) son cortes optimales para el modelo DSL-BF.*

Demostración. Supongamos que u^* es una solución óptima para una instancia del RSA que viola al menos una desigualdad (3.52). Es decir, existen un par de *slots* $s \neq s_1 \in S$, un camino privado PP con arco maestro e', un arco $e \in PP$, y una demanda $d_1 \in D(PP)$ tales que el lado izquierdo de la desigualdad es mayor a 1. A causa de las restricciones de integralidad y las de no solapamiento, a lo sumo una demanda puede usar el *slot* s en el arco maestro e'. Sea d_2 una demanda que cumpla esto, entonces ambos términos del

lado izquierdo deben ser iguales a 1. Dado que s_1 es menor o igual a $v(d_1)$, a causa de las restricciones de integralidad, contiguidad y no solapamiento, d_1 no puede usar el *slot* s_1 en el arco maestro. Como u^* es óptima, no contiene ciclos, por lo que debido al Lema 3.4.1, no puede usar s_1 en ningún arco perteneciente a PP. □

Si la solución tomada en la demostración anterior no fuera óptima, podemos ver que el resultado vale igual siempre y cuando la demanda no utilice ciclos espurios dentro de PP, lo cual no puede ocurrir si éste está compuesto de sólo un arco o un camino simple.

Corolario 3.4.3. *Si el camino privado PP está compuesto por un solo arco, las desigualdades (3.52) son válidas para el modelo DSL-BF.*

Dado que las desigualdades (3.52) dominan a las desigualdades (3.51), podemos enunciar el siguiente resultado.

Teorema 3.4.6. *Las desigualdades (3.51) son satisfechas por cada punto de la relajación lineal del modelo DSL-BF que satisface las desigualdades (3.52).*

Corolario 3.4.4. *Las desigualdades (3.51) son cortes optimales para el modelo DSL-BF en el caso general, y desigualdades válidas cuando PP está compuesto por un solo arco.*

Alternativamente, podemos expresar este hecho considerando todos estos *slots* juntos como en las desigualdades

$$\sum_{s'=1}^{s_1} u_{d_1 e s'} \leq s_2(1 - u_{d_2,e',s}) \qquad \begin{array}{c} s \in \{1,\ldots,v(d_1)\},\ \forall e \in PP, \\ \forall d_2 \in D,\ d_1 \in D(PP),\ d_1 \neq d_2, \end{array} \tag{3.53}$$

donde $s_1 = \max\{s, v(d_2)\}$ y $s_2 = \min\{v(d_1), s_1\}$. Aquí asumimos que d_1 usa exactamente $v(d_1)$ *slots* por lo que (3.53) son cortes optimales, y no están implicadas por las desigualdades (3.51), las cuales permiten a cada demanda utilizar más slots que sus volúmenes.

Podemos combinar las dos variantes considerando todos los *slots* juntos, sumando sobre cada demanda en $D\backslash\{d_1\}$ y relajando la cota superior de la suma tomando $\bar{s_1} = \max\{s, \min_{d' \in D\backslash\{d_1\}}(v(d'))\}$, a saber

$$\sum_{s'=1}^{\bar{s_1}} u_{d_1 e s'} \leq s_2(1 - \sum_{d' \neq d_1} u_{d',e',s}) \qquad \forall e \in PP,\ \forall d_1 \in D(PP),\ \forall s \in \{1,\ldots,v(d_1)\}. \tag{3.54}$$

Teorema 3.4.7. *Las desigualdades (3.54) son cortes optimales para el modelo DSL-BF y no están implicados por las restricciones (3.2)-(3.6).*

Demostración. Dada una instancia del RSA, supongamos que u^* es una solución óptima que viola al menos una de las desigualdades (3.54). Esto significa que existe un camino privado PP con arco maestro e', una demanda $d_1 \in D(PP)$, un arco $e \in PP$ y un *slot* $s \leq v(d_1)$ tales que el lado izquierdo de la desigualdad es estrictamente mayor que el lado derecho. Debido a las restricciones de integralidad y las de no solapamiento la suma $\alpha = \sum_{d' \neq d_1} u^*_{d',e',s} \in \{0,1\}$ pero como $s_2 \leq v(d_1)$ por definición y $\sum_{s' \in S} u^*_{d_1,e,s'} \in \{0, v(d_1)\}$ por la optimalidad de u^*, entonces $\alpha = 1$ para violar la desigualdad. Esto implica que, debido a la integralidad y el no solapamiento, existe una demanda $d_2 \neq d_1$ que usa el *slot* s en el arco maestro e', y como $s \leq v(d_1)$ a causa de la contiguidad, del no solapamiento

y de la integralidad, d_1 no puede usar ningún *slot* menor o igual a s en e', y debido al Lema 3.4.1, tampoco en ningún arco $e \in PP$.

Para demostrar la segunda parte presentamos un contraejemplo fraccionario que satisface las restricciones del modelo mencionadas, *i.e.*, (3.2)-(3.6), pero que viola las desigualdades (3.54). Dada una instancia del RSA con $\bar{s} = 4$, dos demandas $d_1 \neq d_2$ con $v(d_1) = v(d_2) = 2$, y un grafo tal que es posible asignarles a las demandas d_1 y d_2 dos caminos simples P_1 y P_2, respectivamente, que comparten un arco e'. Sea u' una solución fraccionaria tal que $u'_{d_1 e*} = (1, \frac{1}{2}, \frac{1}{4}, \frac{1}{4})$ para cada arco $e \in P_1$ y $u'_{d_2 e*} = (0, \frac{1}{2}, \frac{3}{4}, \frac{3}{4})$ para cada arco $e \in P_2$. Esta solución satisface las restricciones del modelo, pero como $\bar{s_1} = s_2 = 2$, al tomar $s = 2$ y $PP = \{e'\}$ obtenemos $\frac{3}{2} > 2(1 - \frac{1}{2}) = 1$ violando una de las desigualdades (3.54). □

Dado que (3.54) implica (3.53), entonces podemos enunciar el siguiente corolario.

Corolario 3.4.5. *Las desigualdades (3.53) son cortes optimales para el modelo DSL-BF.*

Dos conjuntos de demandas

Para $s \in S$, definimos $D_s^{\geq} = \{d \in D(PP) : v(d) \geq s\}$ como el conjunto de demandas con volúmenes mayores o iguales a s, y definimos $D^{<} = D \backslash D^{\geq}$ al conjunto de las demandas restantes. Podemos entonces considerar una variación de las desigualdades (3.51) sumando en s_1 las demandas de $D_{s_2}^{\geq}$ y sumando en s_2 las demandas de $D_{s_2}^{<}$, obteniendo por lo tanto la desigualdad

$$\sum_{d' \in D_{s_2}^{\geq}} u_{d' e s_1} + \sum_{d' \in D_{s_2}^{<}} u_{d' e' s_2} \leq 1 \qquad \begin{array}{r} \forall e \in PP,\ s_1 \in \{1, \ldots, s_2 - 1\}, \\ s_2 \in \{2, \ldots, \text{máx}_{d \in D(PP)}\, v(d)\}. \end{array} \tag{3.55}$$

Es decir, si alguna demanda perteneciente a $D_{s_2}^{<}$ usa un *slot* s_2 en el *arco maestro* de PP, entonces ninguna demanda en $D_{s_2}^{\geq}$ puede utilizar un *slot* menor que s_2 en ningún arco de PP. Esto es válido tomando cada *slot* s_2 menor o igual al máximo volumen de las demandas de $D(PP)$, en cuyo caso, $D_{s_2}^{\geq}$ estará compuesto únicamente por aquellas con volumen mayor.

Teorema 3.4.8. *Las desigualdades (3.55) son desigualdades válidas para el modelo DSL-BF.*

Demostración. Supongamos que dada una instancia del RSA es posible encontrar una solución factible u' que viole al menos una desigualdad (3.55) al tomar un camino privado PP con un arco maestro e', un *slot* $s_2 \in \{2, \ldots, \text{máx}_{d \in D(PP)}\, v(d)\}$ y un *slot* $s_1 < s_2$. Debido a las restricciones de no solapameinto y a las restricciones de integralidad, cada suma del lado izquierdo de la desigualdad debe ser igual a 0 ó a 1, por lo que para violar la igualdad ambos deben ser iguales a 1, luego existe una demanda $d_2 \in D_{s_2}^{<}$ que usa el *slot* s_2 en el arco maestro e', y otra demanda $d_1 \in D_{s_2}^{\geq} \subseteq D(PP)$ que usa un *slot* $s_1 < s_2 \leq v(d_1)$ en algún arco $e \in PP$. Pero entonces u' violaría también alguna de las desigualdades válidas (3.51). □

Podemos también usar estos dos conjuntos de demandas en (3.54), a saber

$$\sum_{d' \in D_{s_2}^{\geq}} \sum_{s'=1}^{s_2} u_{d' e s'} \leq s_2 \left(1 - \sum_{d' \in D_{s_2}^{<}} u_{d' e' s_2}\right) \qquad \begin{array}{r} \forall e \in PP,\ s_1 \in \{1, \ldots, s_2 - 1\}, \\ \forall s_2 \in \{2, \ldots, \text{máx}_{d \in D(PP)}\, v(d)\}. \end{array} \tag{3.56}$$

Teorema 3.4.9. *Las desigualdades (3.56) son cortes optimales para el modelo DSL-BF.*

Demostración. Sea u^* una solución óptima para una instancia del RSA que viola al menos una de las desigualdades (3.56). Entonces, existe un camino privado PP con un arco maestro e', un *slot* $s_2 \in \{2, \dots, \max_{d \in D(PP)} v(d)\}$ y un *slot* $s_1 < s_2$ tales que el lado derecho de la desigualdad es estrictamente menor que el lado izquierdo. A causa de las restricciones de integralidad y de no solapamiento del modelo, la suma $\beta = \sum_{d' \in D^{<}_{s_2}} u_{d'e's_2}$ debe ser igual a 0 ó a 1. De igual modo, $\sum_{d' \in D^{\geq}_{s_2}} u_{d'es'}$ debe ser igual a 0 ó a 1 para cada *slot* s', con lo que el lado izquierdo debe ser menor o igual a s_2. Entonces $\beta = 1$. Debido a las restricciones de integralidad y las de no solapamiento, esto implica que existe una única demanda $d_2 \in D^{<}_{s_2}$ tal que $u_{d'e's_2} = 1$, por lo que ninguna otra demanda con $v(d') \geq s_2$ puede usar ningún *slot* $s_1 \leq s_2$ en el arco maestro e'; y, por el Lema 3.4.1, tampoco en cualquier arco $e \in PP$. Por lo tanto, el lado izquierdo de la desigualdad debe ser igual a 0. □

Debido al Teorema 3.1.1 podemos formular las desigualdades simétricas a (3.51)-(3.56), las cuales son también válidas para el modelo DSL-BF.

Tres demandas

Sean $d_1, d_3 \in D$, $d_2 \in D(PP)$ y $s_1, s_2, s_3 \in S$ tres demandas y tres *slots* diferentes tales que $s_3 - s_1 \leq v(d_2)$ y $v(d_1) \leq s_1 < s_2 < s_3 \leq \bar{s} + 1 - v(d_3)$. Entonces podemos definir las desigualdades

$$u_{d_1,e',s_1} + u_{d_2es_2} + u_{d_3,e',s_3} \leq 2 \qquad \forall e \in PP, \tag{3.57}$$

las cuales, junto con las restricciones de integralidad, fuerzan a que al menos una de las tres demandas d_i no use el *slot* s_i en dicho arco, porque imposibilitaría asignarle suficiente cantidad de slots a la demanda d_2. La Figura 3.14 representa un ejemplo de esta situación. Una variante de estas desigualdades puede ser obtenida al reemplazar d_1 y d_3 por la suma

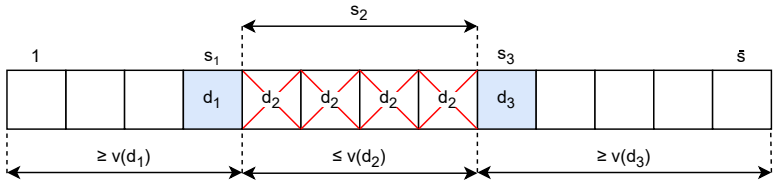


Fig. 3.14: El *slot* usado por la demanda d_2 es menor o igual a $v(d_1)$, con lo que d_1 no puede usar ningún *slot* menor que ese.

de todas las demandas en $D \backslash \{d_2\}$. Aquí estamos forzados a relajar los límites impuestos a los *slots*. Es decir, los tres *slots* deben ser tales que $\min_{d' \in D \backslash d_2} v(d') \leq s_1 < s_2 < s_3 \leq \bar{s} + 1 - \min_{d' \in D \backslash d_2} v(d')$, y $s_3 - s_1 \leq v(d_2)$. En este marco, podemos enunciar la siguiente familia de desigualdades,

$$\sum_{d' \neq d_2} u_{d',e',s_1} + u_{d_2es_2} + \sum_{d' \neq d_2} u_{d',e',s_3} \leq 2 \qquad \forall e \in PP, \forall d_2 \in D(PP). \tag{3.58}$$

Teorema 3.4.10. *Las desigualdades (3.58) son cortes optimales para el modelo DSL-BF.*

Demostración. Dada una instancia del RSA, asumamos que es posible hallar una solución óptima u^* que viole alguna de las desigualdades (3.58). Esto implica que para algún camino privado PP con arco maestro e', *slots* $s_1 < s_2 < s_3$, demanda d_2 y arco e, el lado izquierdo de la desigualdad es mayor a 2. Por las restricciones de no solapamiento y las de integralidad cada término del lado izquierdo debe ser igual a 1 en orden a violar la desigualdad, y $u_{d_2,e's_1} = u_{d_2,e's_1} = 0$. Pero, como la cantidad de *slots* entre s_1 y s_3 es estrictamente menor que el volumen de d_2, entonces $e \neq e'$, y a causa del Lema 3.4.1, u_{d_2,es_2} para cada $e \in PP$. □

Dado que las desigualdades (3.58) dominan a las desigualdades (3.57), es posible enunciar el siguiente resultado.

Teorema 3.4.11. *Las desigualdades (3.57) son satisfechas por cada punto de la relajación lineal del modelo DSL-BF que satisface las desigualdades (3.58).*

Corolario 3.4.6. *Las desigualdades (3.57) son cortes optimales para el modelo DSL-BF.*

Es posible expresar la familia de desigualdades anterior de una forma diferente tomando $M = s_3 - s_1 - 1$ en las desigualdades

$$\sum_{s'=s_1+1}^{s_3-1} u_{d_2es'} \leq M(2 - \sum_{d' \neq d_2} u_{d',e',s_1} - \sum_{d' \neq d_2} u_{d',e',s_3}) \qquad \begin{array}{r} \forall e \in PP, \\ 2 \leq s_3 - s_1 \leq v(d_2). \end{array} \tag{3.59}$$

Corolario 3.4.7. *Las desigualdades (3.59) son válidas para el modelo DSL-BF.*

Subfamilias implementadas

Dado que cada arco en las instancias reales que utilizamos tiene su reverso y los grafos son 2-conexos, éstas tienen muy pocos caminos privados. Por este motivo, las últimas familias presentadas fueron implementadas únicamente tomando cada arco del grafo como un camino privado (*i.e.*, $PP = \{e\}$ para cada $e \in E$, con e como su arco maestro).

Las desigualdades generadas de esta forma son denominadas *posFitLow2DBySlots* y *posFitLow2D- BySlotsOneVsAll* cuando simplifican (3.51) y (3.52), respectivamente. El corte optimal (3.53) para este tipo de camino privado simple es llamado *positionalFittingLower*, cuando $s_1 = s2 = s$, y *positionalFittingLowerTight*, cuando $s_1 = \text{máx}\{s, v(d_2)\}$ y $s_2 = \text{mín}\{v(d_1), s_1\}$. La familia (3.54) resulta en la denominada *positionalFittingLowerOneVsAll*.

Por último, las desigualdades (3.57)-(3.59) generan las familias denominadas como *posFit3DBySlots*, *posFit3DBySum*, *posFit3DByImplication* y *posFitLow2DBySlots2SetsOfDs*, respectivamente.

Reemplazando *Low* por *High* obtenemos el nombre de la familia de desigualdades simétrica para cada una de ellas.

Dado que los resultados de los experimentos mostraron que ningún corte era agregado al utilizar *posFitLow2DBySlots2SetsOfDs*, no hemos implementado la versión reducida de las desigualdades (3.56).

3.4.4. Slots centrales entre límites

Podemos combinar las desigualdades (3.39) con la idea principal de las últimas familias presentadas. Para cada $e \in E$ y $d \in D$, con $v(d) > 1$, si otra demanda usa un *slot*

$s_2 \in [v(d)+1, 2v(d)]$ y d usa un *slot* $s_1 < s_2$, entonces d debe usar los *slots* centrales entre 1 y $s_2 - 1$. Este hecho es capturado por la Figura 3.15 y las desigualdades

$$\sum_{s' \in Y} u_{des'} \geq |Y| \Big(\sum_{d' \in D \setminus \{d\}} u_{d'es_2} + u_{des_1} - 1 \Big) \qquad \forall d \in D,\, e \in E, \tag{3.60}$$

donde $s_2 \in \{v(d)+1, \ldots, 2v(d)\}$, $s_1 < s_2$, e $Y = \{s_2 - v(d), \ldots, v(d)\}$. Esta familia es llamada *centralSsBetweenLowBounds* y su simétrica *centralSsBetweenHighBounds*.

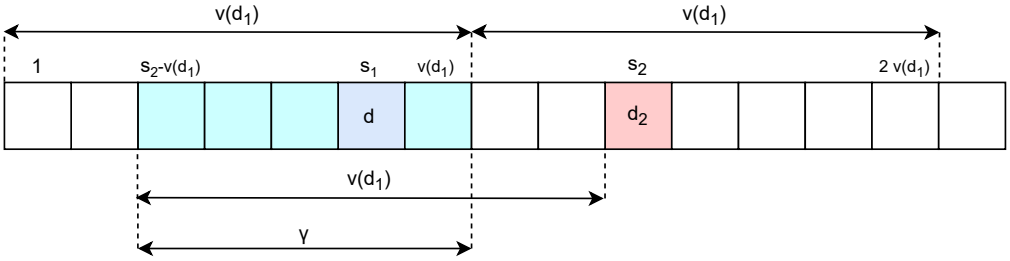


Fig. 3.15: Si la demanda d usa el *slot* s_1 y otra demanda usa s_2, entonces d debe usar todos los *slots* pertenecientes al intervalo Y.

Teorema 3.4.12. *Las desigualdades (3.60) son válidas para el modelo DSL-BF.*

Demostración. Supongamos que tenemos una instancia del RSA y asumamos que existe una solución factible u' que viola al menos una de las desigualdades (3.60). Formalmente, existen $d \in D, e \in E, s_1 < s_2 \in S$ con $s_2 \in \{v(d)+1, \ldots, 2v(d)\}$ tales que el lado derecho de la desigualdad es estrictamente mayor que el lado izquierdo. Por las restricciones de integralidad y de no solapamiento tenemos que el lado izquierdo es menor o igual a $|Y|$, y ambos $\sum_{d' \in D \setminus \{d\}} u_{d'es_2}$ y u_{des_1} deben ser iguales a 1, y por lo tanto $u_{des_2} = 0$. Para lograr que el lado izquierdo sea estrictamente menor que $|Y|$, d no debería utilizar al menos un *slot* $s \in Y$, a causa de la integralidad. Pero dado que d usa $s_1 \in Y$, debe usar $v(d)$ *slots* a la izquierda o a la derecha de s por la contigüidad, pero no hay suficientes *slots* libres ni a la derecha ni a la izquierda. □

Finalmente, también es posible agregar una tercera demanda utilizando otro *slot* como cota inferior, obteniendo con ello

$$\sum_{s' \in Y} u_{des'} \geq |Y| \Big(\sum_{d' \in D \setminus \{d\}} u_{d'es_1} + \sum_{d' \in D \setminus \{d\}} u_{d'es_3} + u_{des_2} - 2 \Big) \quad \forall d \in D,\, e \in E, \tag{3.61}$$

donde $s_1 \in \{1, \ldots, \bar{s} - 2v(d)\}$, $s_3 \in \{s_1 + v(d) + 1, \ldots, s_1 + 2v(d)\}$, $s_1 < s_2 < s_3$, y $Y = \{s_3 - v(d), \ldots, s_1 + v(d)\}$. Estas desigualdades son denominadas *centralSsBetweenDs* y la Figura 3.16 ejemplifica una situación en la que aplicarían.

Asumiendo que existe una solución factible que viola al menos una de las desigualdades (3.60) arribamos a un resultado similar al obtenido al demostrar la validez de las desigualdades (3.60). La única diferencia es que aquí todo está desplazado s_1 *slots* hacia la derecha. Por estos motivos, podemos enunciar el siguiente teorema.

Teorema 3.4.13. *Las desigualdades (3.60) son válidas para el modelo DSL-BF.*

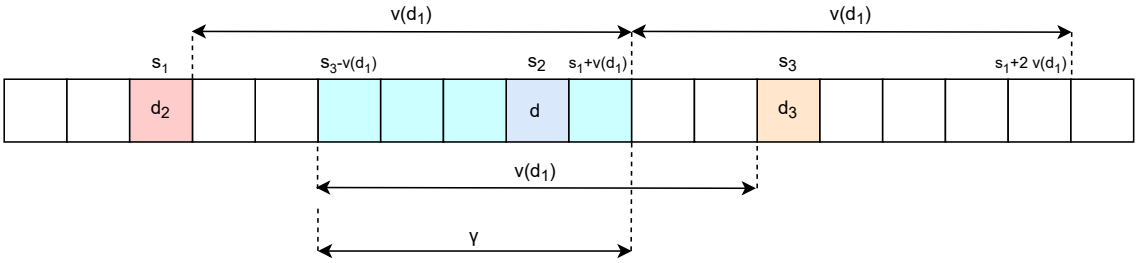


Fig. 3.16: Si la demanda d utiliza el slto s_2 y otras demandas usan s_1 y s_3, entonces d debe usar todos los *slots* dentro del intervalo Y.

3.5. Conclusiones

En el presente capítulo hemos explorado una gran variedad de familias de desigualdades e igualdades válidas, y cortes optimales para la formulación DSL-BF. Dado que incluso caracterizar la dimensión del politopo $RSA(G, D, \bar{s})$ no es una tarea sencilla, no parece directo encontrar resultados de facetitud para estas familias. En cambio, hemos mostrado que –junto con las restricciones del modelo– las desigualdades provenientes de algunos pares de estas familias no se implican mutuamente. Como trabajo a futuro, luego de caracterizar la dimensión del politopo, resta determinar si estas familias definen o no facetas de $RSA(G, D, \bar{s})$. Mientras tanto avanzamos con la experimentación con el fin de determinar empíricamente su efectividad. Para ello, en los siguientes capítulos implementamos un algoritmo *branch-and-cut* utilizando estas desigualdades como planos de corte, pero dado que existen más de 60 familias, proponemos varias estrategias de filtrado y selección con el fin de agregar sólo subconjuntos de ellas que mejoren el rendimiento del algoritmo.

4

Un algoritmo branch-and-cut para DSL-BF

Hemos implementado un procedimiento branch-and-cut para la versión estática del RSA con el fin de evaluar la contribución de las familias de igualdades y desigualdades válidas y los cortes optimales presentados en el Capítulo 3 dentro de un entorno de planos de corte. En el presente capítulo detallamos dicho algoritmo junto con los resultados de los diversos experimentos computacionales realizados con el fin de medir la eficiencia de agregar cada uno de estos cortes. Para ser justos, las comparaciones de las distintas variantes se realizan, además de entre ellas, con los algoritmos genéricos de tipo *branch-and-bound* y *branch-and-cut* implementados por Cplex. La primera de estas últimas mostrará si agregar los cortes genera mejoras con respecto a un *branch-and-bound* simple, mientras que la otra mostrará si los cortes dedicados propuestos son mejores que los cortes genéricos implementados por Cplex.

4.1. Procedimientos de separación

En esta sección exhibimos y estudiamos los procedimientos implementados para separar las familias presentadas en el capítulo anterior. Las complejidades temporales de estos procedimientos son casi todas polinomiales, siendo su peor caso $\mathcal{O}((|D|^k|E|\bar{s}^r)$ con $k, r \leq 3$. Los procedimientos exponenciales para familias que involucran estructuras complicadas sobre el grafo G generalmente calculan previamente un conjunto grande de dichas estructuras y toman un subconjunto aleatorio cada vez.

Definimos un parámetro ϵ para cada procedimiento de separación de tal manera que una desigualdad violada se agrega como corte si el valor absoluto de la diferencia entre el lado izquierdo y el lado derecho es al menos ϵ. La intuición detrás de esta decisión es que una violación más profunda de las desigualdades dejaría más soluciones fraccionarias fuera de la nueva relajación lineal. Esto también se hace a menudo para no agregar demasiados cortes y, en su lugar, usar sólo los más prometedores, y para evitar problemas numéricos. Cuanto mayor sea el valor de este parámetro se agregarán menos cortes. Este parámetro se calibra para cada familia y se fija en su mejor valor para los experimentos finales.

En esta sección, entonces, presentamos los algoritmos de separación para cada familia

de desigualdades e igualdades válidas y cortes optimales. Siempre que exista la familia simétrica, ya sea por simetría de *slots* o por la dirección de los caminos, su procedimiento de separación es análogo. Excepto cuando se menciona, asumimos una instancia de $RSA(G, D, \bar{s})$ (i.e, la cápsula convexa de soluciones factibles de la formulación DSL-BF (3.2)-(3.8)), siendo $G = (V, E)$ el grafo dirigido, D el conjunto de demandas y $\bar{s}$ la capacidad máxima de los enlaces. También asumimos que $u^* \in \mathbb{R}^{|D||E|\bar{s}}$ es una solución fraccionaria para la relajación lineal. Eso significa que u^* satisface todas las restricciones del modelo DSL-BF, excepto las restricciones de integralidad. Para una solución fraccionaria u^* dada, un *slot* s se considera utilizado por una demanda d en un arco e si la variable asociada es mayor a 0, *i.e.*, $u^*_{des} > 0$. Para evitar errores numéricos, en los experimentos utilizamos una tolerancia para las comparaciones, por lo que en estos casos pedimos que u^*_{des} sea mayor que 1×10^{-11}.

4.1.1. Nada desde el nodo destino

El algoritmo desarrollado para separar las igualdades violadas de la familia *nothingFromDst*, a saber,

$$\sum_{s \in S} \sum_{e \in \delta^+(t(d))} u_{des} = 0 \qquad \forall d \in D,$$

es el siguiente. Dada una solución fraccionaria u^*, el procedimiento verifica para cada demanda $d \in D$ si la suma de las variables u^*_{des} para cada *slot* $s \in S$ y arco $e \in \delta^+(t(d))$ es mayor que ϵ. Esto es realizado en $\mathcal{O}(|D||E|\bar{s})$, siendo mucho menos el tiempo en el caso promedio para las instancias utilizadas, debido a que muy pocos arcos salen de cada vértice, *i.e.*, para cada demanda generalmente alcanza con revisar a lo sumo cuatro arcos.

4.1.2. Cada *slot* se usa una única vez desde un vértice

Para separar las desigualdades violadas de la familia *oneSlotOnceFromV*, es decir,

$$\sum_{e \in \delta^+(i)} u_{des} \leq 1 \qquad \forall d \in D, \forall s \in S, \forall i \in V \setminus \{t(d)\},$$

se implementó el siguiente procedimiento. Cuando es llamado con una solución fraccionaria u^*, el mismo itera sobre cada demanda $d \in D$ y cada vértice $i \in V$ filtrando aquellos *slots* $s \in S$ para los cuales $\sum_{e \in \delta^+(i)} u^*_{des} > 1 + \epsilon$. Dado que un arco sale una única vez de un vértice, este algoritmo es ejecutado en $\mathcal{O}(|D||E|\bar{s})$. La versión reducida que considera al origen de cada demanda d en lugar de todos los vértices, *i.e.*, la familia *oneSlotOnceFromSrc*, tiene una complejidad de $\mathcal{O}(\sum_{d \in D} |\delta^+(s(d))|\bar{s})$. Si definimos Δ^+ como la mayor cantidad de arcos que salen de un nodo del grafo, *i.e.*, el mayor grado saliente de los nodos, entonces la complejidad puede expresarse como $\mathcal{O}(|D|\Delta^+\bar{s})$.

4.1.3. A lo sumo su volumen

Cuando la familia de desigualdades *exactlyVdFromV* es agregada, a saber,

$$\sum_{e \in \delta^+(i)} \sum_{s \in S} u_{des} \leq v(d) \qquad \forall d \in D, \forall i \in V,$$

cada demanda puede utilizar a lo sumo su volumen en cada arco. Por lo tanto, el procedimiento que intenta separar desigualdades violadas para esta familia itera sobre el conjunto de vértices y el conjunto de demandas filtrando cada $d \in D$ y vértice $i \in V$ que hacen que $\sum_{e \in \delta^+(i)} \sum_{s \in S} u^*_{des}$ sea mayor que $v(d) + \epsilon$. Una vez más, dado que $\sum_{i \in V} |\delta^+(i)| = |E|$, este proceso es ejecutado en $\mathcal{O}(|D||E|\bar{s})$, mientras que el algoritmo para la versión reducida de esta familia, *i.e.*, *exactlyVdFromSrc*, tiene una complejidad de $\mathcal{O}(|D|\Delta\bar{s})$ en promedio.

4.1.4. Sin bifurcaciones

La familia de desigualdades *notBranchFromV* fuerza dos condiciones: no solamente prohibe a cada demanda d utilizar más de un arco en $\delta^+(v)$ para cada vértice $v \in V$, sino que además limita el número de *slots* usados por dicha demanda en cada subconjunto de $|\delta^+(v)| - 1$ arcos en $\delta^+(v)$. El procedimiento que separa desigualdades violadas para esta familia, a saber,

$$\sum_{e' \in \delta^+(i) \setminus \{e\}} \sum_{s' \in S} u_{de's'} \leq v(d)(1 - u_{des}) \qquad \forall d \in D,\, \forall i \in V,\, \forall e \in \delta^+(i),\, s \in S,$$

puede observarse en el Algoritmo 1. Usamos $\{key : \text{valueFor}(key) \text{ for } key \in list\}$ para definir un diccionario por comprensión, donde cada clave key se obtiene de iterar la lista $list$, y cada valor es el resultado de alguna función sobre la clave. Para cada solución fraccionaria u^*, el procedimiento itera sobre cada demanda $d \in D$ y cada vértice $v \in V$ buscando los arcos $e \in \delta^+(v)$ y *slots* $s \in S$ tales que

$$\sum_{s' \in S} \sum_{e' \in \delta^+(v) \setminus \{e\}} u^*_{de's'} - v(d)(1 - u^*_{des}) > \epsilon.$$

Debido a que $\sum_{i \in V} |\delta^+(i)| = |E|$, podemos precomputar $\sum_{s \in S} u^*_{des}$ para cada demanda $d \in D$ y arco $e \in E$, y también $\sum_{s \in S} \sum_{e \in \delta^+(v)} u^*_{des}$ para cada vértice $v \in V$, ambos en $\mathcal{O}(|D||E|\bar{s})$. Con lo cual, el procedimiento puede ser implementado en $\mathcal{O}(|D||E|\bar{s})$, mientras que el algoritmo para separar las desigualdades violadas de la subfamilia *notBranchFromSrc* es realizado en $\mathcal{O}(|E|\bar{s})$ en promedio.

4.1.5. Cada *slot* se usa en la misma cantidad de arcos

Dado que la familia *eqAmountOfAsForEachUsedS*, definida como

$$\sum_{e \in E} u_{des'} \leq \sum_{e \in E} u_{des} + |E|\Big(1 - \sum_{e \in \delta^+(s(d))} u_{des}\Big) \qquad \forall d \in D,\, \forall s, s' \in S,\, s' \neq s,$$

requiere que las desigualdades *oneSlotOnceFromSrc* también sean satiefechas, el algoritmo de separación propuesto comienza verificando si la desigualdad correspondiente de dicha familia está violada, *i.e.*, si $\beta_{s(d)} = \sum_{e \in \delta^+(s(d))} u^*_{des} > 1 + \epsilon$, en cuyo caso agrega la desigualdad violada. Luego de eso, el algoritmo calcula la suma de las variables asociadas con arcos en los cuales la demanda d usa el *slot* s, *i.e.*, $\beta_s = \sum_{e \in E} u^*_{des}$, y computa $\alpha = |E|(1 - \beta_{s(d)})$, luego itera sobre todo otro *slot* $s' \in S$ con $s' \neq s$, verificando si $\beta_{s'} - \beta_s - \alpha > \epsilon$, siendo $\beta_{s'} = \sum_{e \in E} u^*_{des'}$. El procedimiento total tiene entonces una complejidad de $\mathcal{O}(\bar{s}\ (\sum_{d \in D} \delta^+(s(d)) + |E| + \bar{s}|E|))$, la cual es menor o igual a $\mathcal{O}(\bar{s}|E|(|D| + \bar{s}))$, siendo que $\sum_{d \in D} \delta^+(s(d)) \leq |D||E|$. El pseudocódigo puede observarse en el Algoritmo 2.

Algorithm 1 Procedimiento que, dada una solución u^* y un ϵ, separa las desigualdades de la familia *notBranchFromV* que son violadas por al menos ϵ.

```
1: procedure NOTBRANCHFROMV(Float ε, Solution u*)
2:     for d ∈ D do
3:         sum ← {e : Σ_{s∈S} u*_{des} for e ∈ E}
4:         for v ∈ V do
5:             sum_v ← Σ_{e∈δ+(v)} sum[e]
6:             for e ∈ δ+(v) do
7:                 sum ← sum_v − u*_{des}
8:                 if sum − vd(1 − u*_{des}) > ε then
9:                     addViolatedInequality(d, e, s)
10:                end if
11:            end for
12:        end for
13:    end for
14: end procedure
```

La variación de esta familia que considera la suma sobre todo el rango de *slots*, la *eqAmountOfAsForTheSumOfSs*, *i.e.*,

$$\frac{1}{v(d)}\Big(\sum_{e\in E}\sum_{s'\in S} u_{des'}\Big) \leq \sum_{e\in E} u_{des} + |E|\Big(1 - \sum_{e\in\delta^+(s(d))} u_{des}\Big) \qquad \forall d \in D, \forall s \in S,$$

también requiere que la familia de desigualdades *oneSlotOnceFromSrc* se cumpla, por lo tanto el procedimiento que intenta separar las desigualdades violadas de esta familia es similar al Algoritmo 2. La única diferencia es que, en lugar de iterar sobre cada *slot* $s' \neq s$, calcula la suma β_d como $\sum_{s'\in S} \beta_{s'}$ y verifica si $\frac{(\beta_d)}{v(d)} - \beta_s - \alpha > \epsilon$. La complejidad computacional es la misma.

4.1.6. *Double Brooms*

Las dos familas *incomingDBrooms* y *outgoingDBrooms*, *i.e.*, siendo DB un InDBroom o un OutDBroom, respectivamente, en las desigualdades

$$\sum_{s\in S}\sum_{e\in DB} u_{des} \leq v(d)M_{DB} \qquad \forall d \in D,$$

requieren algunas estructuras para hallar desigualdades violadas; por lo tanto, primero presentamos los algoritmos utilizados para obtener dichas estructuras y luego los procedimientos de separación.

Dado un arco $ab \in E$, un InDBroom está compuesto por todos los arcos $ij \in E$ tales que $j = a$, $j = b$ ó $i = b$, mientras que un OutDBroom se compone por los arcos $ij \in E$ tales que $i = a$, $j = b$ ó $i = b$. Por lo tanto el algoritmo itera para cada arco $ab \in E$ sobre todos los arcos $ij \in E$. Dado que estos algoritmos son ejecutados al comienzo del *branch-and-cut*, su complejidad no afecta el rendimiento de este último. Sin embargo, teniendo dos diccionarios tales que para un nodo dado uno devuelva todos los arcos que llegan y otro todos los que salen del mismo, podemos resolver estos algoritmos más rápidamente

Algorithm 2 Procedimiento que, dada una solución u^* y un ϵ, separa las desigualdades violadas de la familia *oneSlotOnceFromSrc* y aquellas de la familia *eqAmountOfAsForEachUsedS* también violadas pero éstas por al menos ϵ.

```
1: procedure EQAMOUNTOFASFOREACHUSEDS(Float ε, Solution u*)
2:     for d ∈ D do
3:         vd ← d.volume()
4:         if vd = 1 then
5:             continue
6:         end if
7:         sd ← d.source()
8:         for s ∈ S do
9:             s_used ← Σ_{e∈δ+(sd)} u*_des
10:            if s_used > 1 then
11:                addViolatedInequality(d, s)
12:            end if
13:            sum_s ← Σ_{e∈E} u*_des
14:            α ← |E|(1 − s_used)
15:            for s' ∈ S; s' ≠ s do
16:                if Σ_{e∈E} u*_des' − sum_s − α > ε then
17:                    addViolatedInequality(d, s, s')
18:                end if
19:            end for
20:        end for
21:    end for
22: end procedure
```

–al menos en promedio– que el peor tiempo consumido por el algoritmo propuesto, el cual requiere $\mathcal{O}(|E|^2)$.

Una vez que tenemos dichas estructuras, los procedimientos de separación son muy simples. Para cada Double Broom DB y cada demanda $d \in D$ verifican si $\sum_{s\in S}\sum_{e\in DB} u^*_{des} - 3v(d)$ es mayor que ϵ. Como queda claro a partir de sus nombres, la familia *incomingDBrooms* considera el conjunto de los InDBroom's y la otra el de los OutDBroom's.

La complejidad de cada uno de estos algoritmos de separación en una implementación muy sencilla es $\mathcal{O}(|D||E|\bar{s}\ \text{máx}_{DB}(|DB|))$, sin embargo se puede mejorar si precomputamos $\sum_{s\in S}\sum_{e\in E} u^*_{des}$ para cada demanda $d \in D$, lo cual toma $\mathcal{O}(|D||E|\bar{s})$. Con ello, lo único que necesitamos es iterar para cada demanda sobre cada arco $e \in DB$ para cada DB, mejorando la complejidad en el peor caso a $\mathcal{O}(|D||E|\bar{s} + |D|(\sum_{ab\in E}(\delta^-(a) + \delta(b) - 1))) \leq \mathcal{O}(|D|(|E|\bar{s} + \text{máx}_{DB}(|DB|)))$, lo cual, dado que $\text{máx}_{DB}(|DB|) \leq |E|$, nos da una complejidad de $\mathcal{O}(|D||E|\bar{s})$.

4.1.7. Ciclos

Una idea para reducir la complejidad de los algoritmos de separación de desigualdades violadas de las familias *cyclesBySlot*, *cycles*, *extendedCyclesBySlot* y *extendedCycles*,

obtenidas al tomar distintos conjuntos E' en las desigualdades

$$\sum_{e \in E'} u_{des} \leq (|V(E')| - 1) \qquad \forall d \in D, \forall s \in S,$$

$$\sum_{s \in S} \sum_{e \in E'} u_{des} \leq v(d)(|V(E')| - 1) \qquad \forall d \in D,$$

es tener precomputado un conjunto de ciclos. Debido a que el número de ciclos simples (*i.e.*, ciclos sin nodos repetidos) en un grafo es exponencial, siendo mayor o igual a 2^{n-1} si el grafo es conexo y no dirigido, con $n = |E| - |V| + 1$ [44], y por ejemplo 125.664 en un digrafo completo con sólo nueve vértices [64], hemos generado solamente un subconjunto de ellos. Cada ciclo simple de un grafo no dirigido $G = (V, E)$ puede obtenerse mediante una operación XOR de un subconjunto particular de ciclos simples de G, aquellos conocidos con el nombre de *ciclos fundamentales* o, a veces también, *ciclos principales* [6, 13]. Un conjunto de ciclos fundamentales está compuesto, por ejemplo, por los ciclos generados al agregar cada arista $e \in E \setminus T$ a un árbol generador $T \in E$, *i.e.*, $T + \{e\}$. Dado que para cada árbol generador $T \subseteq E$, el conjunto de aristas de la resta $E \setminus T$ es diferente, entonces hay al menos tantos conjuntos de ciclos fundamentales como árboles generadores. En un grafo no dirigido, por ejemplo, el número de árboles generadores diferentes puede llegar a ser $|V|^{|V|-2}$ [133].

Sin embargo, para nuestro propósito de ser capaces de generar todos los ciclos simples del grafo, sólo necesitamos uno de estos conjuntos, como se menciona en [13]. Para conseguir uno de ellos para cada instancia, desarrollamos un algoritmo de la forma más general posible, aunque en algunos pasos hemos aprovechado las particularidades de las topologías utilizadas. Como el cálculo previo del conjunto de ciclos fundamentales se realiza al recibir los datos de la instancia, nuestro foco estuvo en minimizar la memoria utilizada más que en mejorar los tiempos de ejecución. Por lo tanto, en vez de matrices, a veces preferimos usar conjuntos, *arrays* y listas. La idea es contar con estructuras robustas pero no demasiado pesadas que permitan realizar consultas en tiempos razonables dentro del procedimiento de separación.

Primero enunciamos un conjunto de definiciones necesarias para comprender el algoritmo. Sea D un digrafo y $U(D)$ el grafo no dirigido resultante de eliminar el sentido a cada arco de D. Dado que no trabajamos con un multigrafo, si ambos arcos ij y ji pertenecen al digrafo original D, en el grafo resultante tendremos únicamente uno de ellos. Llamamos *extender* un grafo no dirigido G –producto de extender D– al proceso inverso, es decir, agregar todos los arcos de D que son incidentes a los nodos de G. Denominamos como *ciclo dirigido* de un grafo $G = (V, E)$ a todo conjunto de arcos $C = \{ab, bc, \ldots, yz, za\} \subseteq E$, tal que para cada arco $ij \in C$ existe otro arco $jk \in C$, y si $ij \in C$ entonces $ji \notin C$. Es decir, no permitimos que dos arcos tales que uno sea el reverso del otro pertenezcan a un ciclo dirigido. Definimos como $e^{-1} \in E$ al arco reverso de e, y $C^{-1} \subseteq E$ al ciclo reverso de C, *i.e.*, $\{e^{-1} : e \in C\}$.

Generación de la matriz de ciclos fundamentales

Como primer paso, el algoritmo calcula un árbol generador T a partir del grafo no dirigido $U(D)$, siendo D el grafo dirigido recibido (*i.e.*, la topología). Se supone que este grafo es conexo y que cada arco tiene su reverso. Este hecho tiene sentido para las instancias reales con las que estamos trabajando, porque las fibras ópticas suelen tenderse en

ambas direcciones. Luego T se extiende con todos los arcos originales obteniendo un árbol generador extendido ET, como se muestra en la Figura 4.1. El algoritmo es prácticamente un BFS, por lo que su complejidad es $\mathcal{O}(|V| + |E|)$.

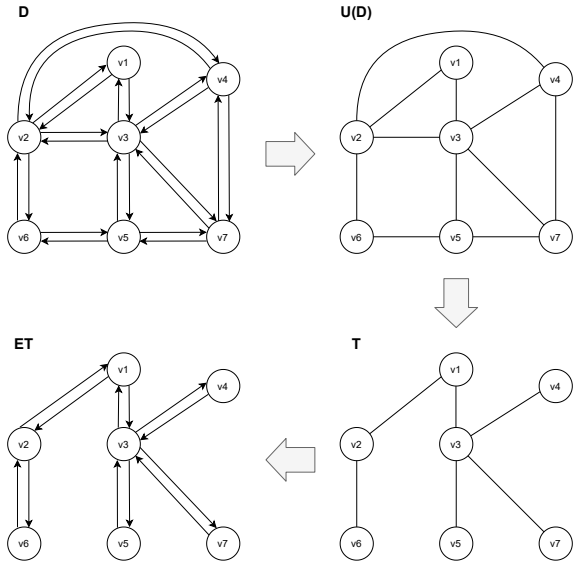


Fig. 4.1: Algoritmo de construcción del árbol generador. Recibe un digrafo conexo D con todos sus arcos teniendo reverso y retorna un árbol generador extendido ET.

Dado que ET es un árbol generador tal que todos sus arcos son bidireccionales, agregar cualquier arco $ij \in D \setminus ET$ produce un único ciclo (ver Figura 4.2). Agregar el arco ji genera el mismo ciclo en la dirección opuesta. Luego, el algoritmo intenta generar un ciclo para cada arco que no está en el árbol. Como también vamos a extender los ciclos, si genera un ciclo en un sentido no intenta generar el opuesto. Los arcos que usamos en este segundo paso se denominan *arcos principales*, abreviados *MArcs*, y los diferentes ciclos producidos por ellos son los ya mencionados ciclos fundamentales o ciclos principales, abreviados como *MCycles*. Dado que el árbol tiene exactamente $2(|V|-1)$ arcos, la cantidad de arcos principales debe ser $|MArcs| = |E| - 2(|V| - 1)$, pero como utilizamos sólo la mitad de ellos para obtener los ciclos, *i.e.*, $|MArcs| = 2|MCycles|$, entonces este paso es realizado en menos que $\mathcal{O}(|MCycles||ET|) = \mathcal{O}(|E|(|V| - 1) - 2(|V| - 1)^2) \leq \mathcal{O}(|E||V| - |V|^2)$.

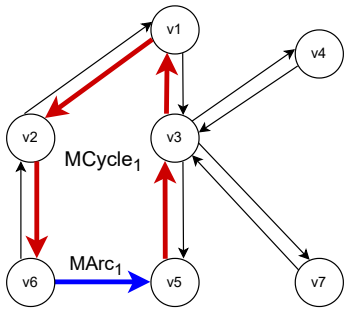


Fig. 4.2: Agregar un arco principal al árbol generador extendido produce un ciclo principal.

Usamos este tipo de árbol generador extendido porque el problema con la versión no extendida es que un arco adicional podría no generar un ciclo cuando se usan los arcos originales. Aquí estamos aprovechando el hecho de que las topologías reales tienen todos sus arcos con su reverso.

Una vez obtenidos los ciclos principales, el algoritmo rellena la matriz de vecinos. Definimos dos ciclos C_1 y C_2 como vecinos si C_1 o su reverso C_1^{-1} comparten al menos un arco con C_2 o su reverso C_2^{-1}, *i.e.*, $(C_1 \cup C_1^{-1}) \cap (C_2 \cup C_2^{-1}) \neq \emptyset$. Dado que cada arco tiene su reverso, alcanza con chequear para cada par de ciclos $C_1 \neq C_2$ si C_2 contiene algún arco de C_1 o C_1^{-1}. Con estos resultados se genera la matriz booleana simétrica M donde $M_{ij} = True$ si y sólo si los ciclos C_i y C_j son vecinos. Este paso toma $\mathcal{O}(|MCycles|^2)$.

Generación de todos los ciclos simples del grafo

Cada ciclo simple del grafo es una combinación de los ciclos principales generados previamente. La forma de combinar estos ciclos principales es aplicando XOR a sus arcos. Por lo tanto, implementamos una función que recibe una lista de ciclos principales y retorna el ciclo resultante de tomar sólo aquellos arcos que pertenecen a una cantidad impar de ciclos. Obviamente, no todas las combinaciones dan un ciclo conexo, pero cuando dan como resultado dos o más ciclos desconexos, la función falla. Como poda, cuando el conjunto de ciclos recibido está compuesto por sólo dos ciclos, verificamos que sean vecinos en la matriz antes de intentar la unión. En el peor caso debemos iterar sobre cada ciclo del conjunto Cs recibido, *i.e.*, $\mathcal{O}(|E| \times |Cs|)$. Hay muchas formas de mejorar esta complejidad. No es necesario revisar si cada arco principal pertenece a cada ciclo, por ejemplo, y también es posible generar una buena estructura que relacione arcos con ciclos principales en $\mathcal{O}(1)$ reduciendo drásticamente la cantidad de iteraciones.

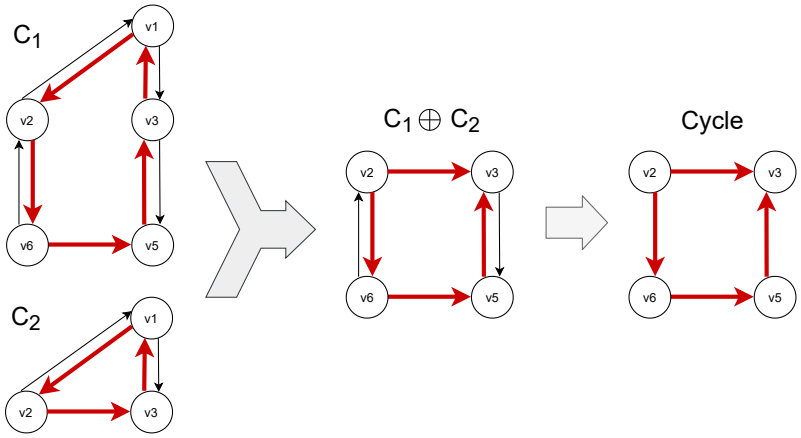


Fig. 4.3: Generación de un ciclo simple producto de la unión de dos ciclos principales extendidos.

Como usamos ciclos principales extendidos para obtener los nuevos ciclos, el siguiente paso es recuperar las versiones dirigidas de éstos. Esta parte del algoritmo se implementó permitiendo tener arcos sin reverso pensando en hacerlo más general. El conjunto de arcos recibidos es siempre un conjunto de ciclos, por lo que el algoritmo intenta reconstruir un ciclo simple en un sentido utilizando DFS y manteniendo un conjunto de los arcos visitados (sin sus reversos) y un conjunto de nodos visitados. Si hay al menos un nodo no visitado y todos los arcos tienen reverso, falla; si hay algún arco sin reverso, intenta

reconstruir el ciclo en la dirección opuesta. Si hay un nodo no visitado, vuelve a fallar. Si se visitaron todos los nodos y la cantidad de aristas del grafo no dirigido no es igual a la cantidad de nodos, falla. En cualquier otro caso, retorna el subgrafo generado. El algoritmo DFS devuelve toda la componente conexa, incluso si comparten sólo un nodo. Este paso es realizado en tiempo polinomial sobre cada conjunto de arcos $G' = (V', E')$, y como la cantidad de arcos es siempre $V' \leq E' \leq 2V'$, entonces es resuelto en $\mathcal{O}(|V'|)$. La Figura 4.3 muestra un ejemplo de estos últimos pasos generando un ciclo simple a partir de dos ciclos principales.

Dado que hay una gran cantidad de ciclos en un grafo, 16.064 en un digrafo completo con sólo ocho nodos [64], y como estamos trabajando con instancias con hasta 43 nodos, el algoritmo permite limitar la cantidad recuperada a un entero positivo k. En los experimentos usamos los primeros $k = 1.000$ ciclos. Una posible variación es tomar k ciclos al azar, pero creemos que no aportaría una mejora dado que los ciclos utilizados se recuperan uniendo los ciclos principales, los cuales están conformados por conjuntos de arcos muy diferentes, por lo que los ciclos resultantes elegidos así secuencialmente serían suficientemente diferentes entre sí. Además, los experimentos sugirieron que estos cortes no eran lo suficientemente buenos como para intentar mejorar los algoritmos.

Los procedimientos de separación

Una vez computados los conjuntos de ciclos y ciclos extendidos, se ejecutan los correspondientes procedimientos de separación con el fin de hallar desigualdades violadas para las dos variaciones de las familias *extendedCycles* y *cycles*.

El algoritmo que separa las desigualdades violadas de la familia *cyclesBySlots* toma un ciclo C, y revisa para cada demanda $d \in D$ y *slot* $s \in S$ si $\sum_{e \in C} u^*_{des} > |C| - 1 + \epsilon$, en cuyo caso halló una desigualdad violada. El algoritmo que separa las desigualdades de *cyclesBySum* agrupa todos los *slots* juntos buscando una demanda y un ciclo tales que la suma de las variables utilizadas sea mayor que $(|C| - 1) \times v(d) + \epsilon$.

La única diferencia con los procedimientos que separan las familias *extendedCycles* y *extendedCyclesBySlot* radica en los conjuntos sobre los cuales iteran.

4.1.8. Los ciclos más simples

Podemos separar las desigualdades violadas de las familias *simplestCyclesBySlot* y *simplestCycles*, obtenidas de tomar E' como el conjunto $\{ij, ji\}$, cuando ambos arcos existen entre los nodos i y j en las desigualdades

$$\sum_{e \in E'} u_{des} \leq 1 \qquad \forall d \in D, \forall s \in S,$$

$$\sum_{s \in S} \sum_{e \in E'} u_{des} \leq v(d) \qquad \forall d \in D,$$

respectivamente, y las desigualdades violadas de las familia *simplestCyclesImplication*, a saber,

$$\sum_{s' \in S} u_{d,ji,s'} \leq v(d)(1 - u_{d,ij,s}) \qquad \forall ij, ji \in E, \forall d \in D, \forall s \in S.$$

para dos arcos $ij, ji \in E$ (cuando tal estructura exista), en tiempo polinomial con una complejidad de $\mathcal{O}(|D||E|\bar{s})$.

Para las tres familias iteramos sobre cada arco $ij \in E$, tal que exista su reverso, *i.e.*, $ji \in E$. Para la primera familia buscamos una demanda $d \in D$ y un *slot* $s \in S$ tales que $u^*_{d,ij,s} + u^*_{d,ji,s} > 1 + \epsilon$. El segundo procedimiento, *i.e.*, el que separa desigualdades de la familia *simplestCycles*, agrupa todos los *slots* juntos buscando que la suma de las variables asociadas a los *slots* usados por d en los arcos ij y ji sea mayor que $v(d)+\epsilon$ para tener una desigualdad violada. El pseudocódigo para estos dos procedimientos puede ser observado en los Algoritmos 3 y 4, respectivamente.

El tercer y último algoritmo, *i.e.*, el procedimiento que separa desigualdades de la familia *simplestCyclesImplication*, es levemente diferente. Éste también itera sobre cada demanda d, pero, para minimizar el cómputo, primero revisa si la suma de las variables asociadas con la demanda d y los arcos $\{ij, ji\} \subseteq E$ es mayor a cero. Luego, para obtener una desigualdad violada, busca un *slot* $s \in S$ en el arco ij tal que $\sum_{s' \in S} u^*_{d,ji,s'} + v(d)\ u^*_{d,ij,s} > v(d) + \epsilon$. Por último intenta con el proceso opuesto, *i.e.*, sumando sobre ij y buscando *slots* s en ji. Este pseudocódigo es mostrado en el Algoritmo 5

Algorithm 3 Procedimiento que dada una solución u^*, una lista de arcos con reverso en G, y un ϵ separa las desigualdades de la familia *simplestCyclesBySlot* que son violadas por al menos ϵ.

```
1: procedure simplestCyclesBySlot(Arc[] arcs, Float ϵ, Solution u*)
2:     for d ∈ D do
3:         for ij ∈ arcs do
4:             for s ∈ S do
5:                 if u*_{d,ij,s} + u*_{d,ji,s} > 1 + ϵ then
6:                     addViolatedInequality(d, ij, s)
7:                 end if
8:             end for
9:         end for
10:    end for
11: end procedure
```

Algorithm 4 Procedimiento que, dada una solución u^*, una lista de arcos con reverso en G y un ϵ, separa las desigualdades de la familia *simplestCycles* que son violadas por al menos ϵ.

```
1: procedure simplestCycles(Arc[] arcs, Float ϵ, Solution u*)
2:     for d ∈ D do
3:         for ij ∈ arcs do
4:             if Σ_{s∈S} u*_{d,ij,s} + u*_{d,ji,s} > d.volume() +ϵ then
5:                 addViolatedInequality(d, ij, s)
6:             end if
7:         end for
8:     end for
9: end procedure
```

Algorithm 5 Procedimiento que dada una solución u^*, una lista de pares de arcos con reverso, y un ϵ separa las desigualdades de la familia *simplestCyclesImplication* que son violadas por al menos ϵ.

```
1: procedure SIMPLESTCYCLESIMPLICATION(Pair[] pairs, Float ε, Solution u*)
2:     for d ∈ D do
3:         vd ← d.volume()
4:         for (ij, ji) ∈ pairs do
5:             sum ← {
6:                 ij : Σ_{s∈S} u*_{d,ij,s},
7:                 ji : Σ_{s∈S} u*_{d,ji,s}
8:             }
9:             if sum[ij] + sum[ji] = 0 then
10:                continue
11:            end if
12:            for s ∈ S do
13:                if sum[ij] + vd  u*_{d,ji,s} > vd + ε then
14:                    addViolatedInequality(d, j, i, s)
15:                end if
16:                if sum[ji] + vd  u*_{d,ij,s} > vd + ε then
17:                    addViolatedInequality(d, i, j, s)
18:                end if
19:            end for
20:        end for
21:    end for
22: end procedure
```

4.1.9. Igualdades de contigüidad

La familia *contiguityEqs* asegura que agrupando los *slots* separados por $v(d)$ para una demanda $d \in D$ dada, dicha demanda d debe utilizar la misma cantidad de *slots* de cada uno de esos grupos en cualquier arco en una solución óptima. Esto se debe a que toda demanda d debe ser satisfecha usando exactamente $v(d)$ *slots*. Esta familia puede ser descrita de la siguiente manera,

$$\sum_{\substack{s \in S: \\ s \equiv i\ (v(d))}} u_{des} \;=\; \sum_{\substack{s \in S: \\ s+1 \equiv i\ (v(d))}} u_{des} \qquad \forall d \in D,\, \forall e \in E,\, \forall i \in \{1, \ldots, v(d)\}.$$

Con el fin de evitar una comparación exhaustiva entre todos los pares de grupos, lo cual es necesario cuando la tolerancia no es 0, el procedimiento de separación calcula para cada demanda $d \in D$ y cada arco $e \in E$, la suma de las variables asociadas con el grupo de *slots* $s \equiv i\ (v(d))$ y la suma de variables relacionadas con el grupo de *slots* $s \equiv i+1\ (v(d))$ para $i \in \{1, \ldots, v(d)-1\}$. Si la diferencia absoluta es mayor que ϵ, entonces se halló una igualdad violada. Notar que si $\epsilon > 0$, este procedimiento puede no hallar algunas igualdades violadas cuando, por ejemplo, la diferencia entre las sumas de las variables relacionadas con cada grupo consecutivo de *slots* es cercano pero menor que ϵ. En ese caso, la suma de la tolerancia podría ser mayor que ϵ entre algún *slot* s y otro $s+k$ con $k \in \{2, \ldots, v(d)\}$. La complejidad del algorimo, utilizando una estructura que almacene

cada suma, es $\mathcal{O}(|D||E|\bar{s})$, mientras que la búsqueda exhaustiva que no implementamos implica realizar $\binom{v(d)}{2}$ comparaciones para cada arco e y demanda d.

4.1.10. Desigualdades de contigüidad

Para separar las desigualdades de la familia *contiguityIneqs*, a saber,

$$\sum_{\substack{s\in\{1,\dots,i\}:\\ s\ \equiv\ i\ (v(d))}} u_{des} \ \geq \sum_{\substack{s\in\{1,\dots,i-1\}:\\ s+1\ \equiv\ i\ (v(d))}} u_{des} \qquad \forall d \in D,\, \forall e \in E,\, \forall i \in S,$$

el algoritmo itera sobre cada demanda $d \in D$ con volumen $v(d)$ mayor a 1, y sobre cada arco $e \in E$. Luego, para obtener una desigualdad violada, busca un $i \in \{2, \dots, \bar{s}\}$ tal que la diferencia entre la suma de u^*_{des} para $s \equiv i-1\ (v(d))$ y la suma para $s \equiv i\ (v(d))$ sea menor que ϵ. Este procedimiento, guardando la suma para cada i, tiene la misma complejidad que el que implementado para separar la familia *contiguityEqs*, dado que requiere comparar pares de grupos consecutivos.

4.1.11. *Slots* principales

La subfamilia *ppalSlotsFromSrc* establece que cuando se toman *slots* separados por $v(d)$ para cada demanda $d \in D$ en todos los arcos que salen del origen $s(d)$, cada grupo de las variables asociadas debe sumar 1, a saber,

$$\sum_{e\in\delta^+(s(d))} \sum_{\substack{s\in S:\\ s\ \equiv\ i\ (v(d))}} u_{des} \ = \ 1 \qquad \forall d \in D,\, \forall i \in \{1, \dots, v(d)\}.$$

Por lo tanto, el procedimiento de separación correspondiente busca una demanda d y un $i \in \{1, \dots, v(d)\}$ tales que el valor absoluto del lado izquierdo de la igualdad, con $s \equiv i\ (v(d))$, sea mayor que $1+\epsilon$. Es decir, si la suma es menor que $1-\epsilon$ ó mayor que $1+\epsilon$, hemos hallado una igualdad violada. El algoritmo de separación para la familia simétrica *ppalSlotsToDst* es análogo sobre $\delta^-(t(d))$. Teniendo acceso en $\mathcal{O}(1)$ a cada arco perteneciente a $\delta^-(v)$ y $\delta^+(v)$ para cada $v \in V$, estos procedimientos son ejecutados en $\mathcal{O}(\bar{s} \sum_{d\in D} |\delta^+(s(d))|)$ y $\mathcal{O}(\bar{s} \sum_{d\in D} |\delta^-(t(d))|$, respectivamente.

El algoritmo que busca desigualdades violadas pertenecientes a la subfamilia *onePpalSlotsFromSrc* toma únicamente un grupo, en nuestro caso el obtenido fijando $i = 1$, y es implementado en $\mathcal{O}(\sum_{d\in D} |\delta^+(s(d))|)$.

4.1.12. *Slots* centrales

Las familias de desigualdades *centralSlotsFromSrc* y *centralSlotsFromSrcBySlot*, a saber,

$$\sum_{e\in\delta^+(s(d))} \sum_{s\in Y} u_{des} = |Y| \qquad Y = \{\bar{s} - v(d) + 1, \dots, v(d)\},$$

$$\sum_{e\in\delta^+(s(d))} u_{des} = 1 \qquad Y = \{\bar{s} - v(d) + 1, \dots, v(d)\},\, \forall s \in Y,$$

que toman los *slots* centrales de un arco, sólo cobran sentido si existe al menos una demanda $d \in D$ tal que $v(d) > \bar{s}/2$, por lo que antes de ejecutar el *branch-and-bound*, precomputamos este conjunto de demandas.

El procedimiento de separación de *centralSlotsFromSrc* busca una demanda d en dicho conjunto tal que el valor absoluto de la suma de las variables correspondientes a los *slots* centrales de los arcos pertenecientes a $\delta^+(s(d))$ sea mayor que $v(d)+\epsilon$. Esto es, si $\left|\sum_{s\in Y} u^*_{des} - |Y|\right| > \epsilon$ para el conjunto $Y = \{\bar{s}-v(d),\ldots,v(d)\}$ y $e \in \delta^+(s(d))$, entonces hay una desigualdad violada. Este procedimiento es implementado en $\mathcal{O}(\sum_{d\in D'}(2v(d)-\bar{s})|\delta^+(s(d))|)$ siendo D' el conjunto de demandas con volúmenes mayores a $\bar{s}/2$, *i.e.*, que no entran dos veces en un arco.

El algoritmo que separa las desigualdades violadas de la familia *centralSlotsFromSrcBySlot* considera cada *slot* por separado buscando un *slot* s tal que $\left|\sum_{e\in\delta^+(s(d))} u_{des}\right| > 1+\epsilon$. La complejidad temporal es la misma que la del procedimiento anterior.

4.1.13. Forzado a ser utilizado debido a su posición

Las desigualdades *lowPositionForces* aseguran que si una demanda d utiliza un *slot* $s < v(d)$, entonces los *slots* desde s hasta $v(d)$ también deben ser usados por la demanda d, es decir,

$$\sum_{s'=s+1}^{v(d)} u_{des'} \geq (v(d)-s)\; u_{des} \qquad \forall e\in E,\, \forall d\in D,\, \forall s\in\{1,\ldots,v(d)-1\}.$$

Estas desigualdades tienen sentido únicamente sobre demandas con un volumen mayor a 1. Por lo tanto, el procedimiento de separación asociado itera únicamente sobre este subconjunto de demandas D'. Dicho algoritmo busca una demanda $d\in D'$, un arco $e\in E$, y un *slot* $s\in\{1,\ldots,v(d)-1\}$ tales que $u^*_{des}=1$ y la suma de las variables relacionadas con d y con los *slots* en el rango $\{s,\ldots,v(d)\}$ en ese arco sea menor que $v(d)-s-\epsilon$. Este proceso puede ser implementado en $\mathcal{O}(|E|\sum_{d\in D} v(d))$ con una estructura que guarde la suma acumulada en orden inverso desde el *slot* $v(d)$ hasta el primero, *i.e.*, almacenando $[u^*_{de,v(d)},\ u^*_{de,v(d)}+u^*_{de,v(d)-1},\ldots]$ para la solución fraccionaria actual u^*.

4.1.14. No se utilizan los *slots* más alejados

Para hallar desigualdades violadas de la familia *farSlotsOff*, es decir,

$$\sum_{s'\in S'_{sd}} u_{des'} \leq M(1-u_{des}) \qquad \forall e\in E,\, \forall d\in D,\, \forall s\in S,\, M=\min\{|S'_{sd}|, v(d)\},$$

con $S'_{sd} = S - \{\max\{s-v(d),1\}, \min\{s+v(d),\bar{s}\}\}$, buscamos una demanda $d\in D$, un arco $e\in E$, y un *slot* $s\in S$ tales que la suma de $u^*_{des'}$ para $s'\in S'_{sd}$ sea mayor que $\min\{|S'_{sd}|, v(d)\}(1-u^*_{des})$. Este procedimiento es implementado en $\mathcal{O}(|D||E|\bar{s}^2)$. Sin embargo, dado que para cada *slot* s estamos calculando la suma sobre las variables asociadas a *slots* no pertenecientes a la ventana deslizable W centrada en s, tomando dos estructuras que guarden las sumas acumuladas desde el *slot* 1 hasta el primero antes de los pertenecientes a W, y desde el primer *slot* después de W hasta $\bar{s}$, todo el algoritmo puede ser implementado en $\mathcal{O}(|D||E|\bar{s})$.

4.1.15. Restricciones de contigüidad simétricas

Las desigualdades *symmetricalBFContiguity* establecen que si una demanda d utiliza un *slot* s en el arco e pero no el *slot* $s-1$, entonces debe utilizar todos los *slots* en el rango

$\{s, \ldots, s + v(d) - 1\}$, a saber,

$$\sum_{s'=1}^{v(d)} u_{des'} \geq v(d)\ u_{de1} \qquad \forall d \in D,\ \forall e \in E$$

$$\sum_{s'=s}^{f} u_{des'} \geq v(d)(u_{des} - u_{de,s-1}) \qquad \forall d \in D,\ \forall e \in E,\ \forall s \in \{2, \ldots, \bar{s}\},\ f = \text{mín}\{\bar{s}, s + v(d) - 1\}.$$

Por lo tanto, el procedimiento de separación busca una demanda $d \in D$, un arco $e \in E$ y un *slot* $s \in \{1, \ldots, \bar{s}\}$ tales que la diferencia entre $v(d)$ $(u^*_{des} - u^*_{de,s-1})$ y la suma $\sum_{s'=s}^{f} u^*_{des'}$ sea mayor que ϵ, con $f = \text{mín}\{s + v(d), \bar{s}\}$. Usando una ventana deslizante de $v(d)$ *slots* para calcular la suma, *i.e.*, agregando un nuevo término al final cuando se elimina el primero, esta rutina puede ser ejecutada en $\mathcal{O}(|D||E|\bar{s})$.

4.1.16. Contigüidad del modelo ASCC

La familia *contiguityASCC* para soluciones enteras establece que si dos *slots* $s_1, s_2 \in S$ con $s_1 + 1 < s_2$ son utilizados por una demanda $d \in D$ en un arco $e \in E$, entonces el *slot* $s_1 + 1$ también debe ser usado por d en dicho arco. Esto es,

$$u_{des_1} + u_{des_2} \leq u_{de(s_1+1)} + 1 \qquad \forall d \in D,\ \forall e \in E,\ \forall s_1, s_2 \in S,\ s_2 > s_1.$$

Para hallar todas las desigualdades violadas por una solución fraccionaria u^*, el procedimiento de separación debe hallar todas las demandas $d \in D$, arcos $e \in E$ y pares de *slots* s_1, s_2 tales que $1 \leq s_1 < \bar{s} - 1$, $s_1 + 2 \leq s_2 < \bar{s}$ y $u^*_{des_1} + u^*_{des_2} - u^*_{de,s_1+1} > 1 + \epsilon$. Dado que para iterar sobre los *slots* se requieren $\frac{\bar{s}(\bar{s}-1)}{2}$ pasos, este algoritmo ejecuta en $\mathcal{O}(|D||E|\bar{s}^2)$.

4.1.17. No utilizables debido a la contigüidad y al no solapamiento

Para separar las desigualdades violadas de la familia *nonOver*, *i.e.*,

$$u_{d_2es_1} + u_{d_1es_2} + u_{d_2es_3} \leq 2 \qquad \forall e \in E,\ \forall d_1 \neq d_2 \in D,\ \forall s_1 < s_2 < s_3 \in S,$$

dada una solución fraccionaria u^* iteramos sobre cada arco $e \in E$, y para cada par de demandas $d_1, d_2 \in D$ buscamos tres *slots* $1 \leq s_1 < s_2 < s_3 < \bar{s}$ tales que $u^*_{d_2es_1} + u^*_{d_1es_2} + u^*_{d_2es_3} > 2 + \epsilon$. Para la variación *nonOverBySum*, en lugar de sobre una sola demanda d_2, calculamos la suma sobre todas las otras demandas distintas a d_1. Esta mejora es implementada en $\mathcal{O}(|D||E|\bar{s}(|D| + \bar{s}^2))$ mientras que el procedimiento original ejecuta en $\mathcal{O}(|D|^2|E|\bar{s}^3)$.

Las versiones reducidas de estas familias, *nonOverFixNextLow*, y su mejora, *nonOverFixNextLowBySum*, se obtienen fijando $s_3 = s_2 + 1$, permitiendo a s_1 y s_2 moverse en los rangos $[1, \bar{s} - 2]$ y $[s_1 + 1, \bar{s} - 1]$, respectivamente, mientras que *nonOverFixNextHi* y *nonOverFixNextHiBySum* son obtenidas fijando $s_1 = s_2 - 1$, siendo $s_2 \in [2, \bar{s} - 2]$ y $s_3 \in [s_2 + 1, \bar{s}]$, por lo tanto sus procedimentos de separación requieren una iteración menos sobre el rango S, reduciendo así por un factor de $\bar{s}$ sus complejidades computacionales.

4.1.18. Conjunto de demandas que exceden la capacidad del arco

El Algoritmo 6 genera la estructura necesaria para separar las desigualdades violadas de las familias *KDemandsDoNotExceedS* y *KDemandsDoNotExceedSBySum*, a saber,

$$\sum_{s' \in S} u_{des'} \leq v(d) \sum_{d' \in D' \setminus \{d\}} \left(v(d') - \sum_{s' \in S} u_{d'es'} \right) \qquad \forall d \in D', \forall e \in E,$$

$$\sum_{s \in S} \sum_{d \in D'} u_{des} \leq \sum_{d \in D'} v(d) - \min_{d \in D'} v(d) \qquad \forall e \in E,$$

así como las respectivas subfamilias que toman $K = 2$, *i.e.*, *twoDemandsDoNotExceedS* y *twoDemandsDoNotExceedSBySum*. Este procedimiento, que puede ser también implementado recursivamente, es ejecutado una única vez por instancia. La estructura retornada es un diccionario cuyos valores son conjuntos de conjuntos de demandas y cada índice representa la longitud de los conjuntos apuntados por éste. Cada conjunto de demandas está compuesto por demandas que no pueden estar todas juntas en un arco porque la suma de sus volúmenes excede $\bar{s}$. Cada uno de estos conjuntos es además minimal, en el sentido de que si se remueve una demanda cualquiera, la suma de los volúmenes de las restantes es menor o igual a $\bar{s}$.

Algorithm 6 Procedimiento que genera un diccionario cuyas claves son enteros indicando la cantidad de demandas de los conjuntos guardados como valores. Cada valor representa un conjunto minimal de demandas tal que la suma de sus volúmenes excede $\bar{s}$, pero, si se remueve cualquier demanda, la suma de los volúmenes de las restantes es menor o igual a $\bar{s}$.

```
1:  procedure getNSetsOfDemandsThatExceedS
2:      D̄ : {}
3:      D′ : []
4:      ds : getSortedDemands()
5:      idx ← |ds| − 1
6:      while True do
7:          D′.append(idx)
8:          if ∑_{d∈D′} v(d) > s̄ then
9:              D̄[ |D′| ].append(D′)
10:             idx ← D′.pop() - 1
11:         else
12:             idx ← D′.last() - 1
13:         end if
14:         while idx < 0 do
15:             if |D′| = 0 then
16:                 return D̄
17:             end if
18:             idx ← D′.pop() - 1
19:         end while
20:     end while
21: end procedure
```

El algoritmo comienza inicializando algunas variables que serán utilizadas luego. La variable $\bar{D}$ va a guardar la mencionada estructura resultante, *i.e.*, un diccionario de listas

de demandas indexadas por la longitud de dichas listas, por lo que comienza siendo un diccionario vacío. La variable *ds* almacena la lista completa de las demandas ordenada de acuerdo a sus volúmenes en orden ascendente, *i.e.*, *ds* es tal que $v(ds[i-1]) \leq v(ds[i])$ para cada $i \in [2, |D|]$, lo cual es ejecutado en $\mathcal{O}(|D|\log(|D|))$ por la función *getSortedDemands* utilizando *DualPivotQuickSort*. La variable *idx* va a llevar cuenta del índice en *ds* de la última demanda agregada a D', que es la lista temporal que almacena las demandas antes de agregarlas a $\bar{D}$. El algoritmo itera sobre la lista de demandas *ds* desde la de mayor volumen hasta la de menor volumen, por lo que *idx* comienza apuntando al último elemento de dicha lista. En cada iteración, el algoritmo agrega la demanda de la posición *idx* a D' y revisa si el volumen acumulado es mayor que $\bar{s}$, en cuyo caso agrega la lista D' al resultado y remueve la última demanda agregada. Luego de eso, y en ambos casos, almacena en *idx* la demanda anterior a la última agregada con la idea de intentar cada posible combinación.

Para cada iteración i, los elementos que contiene el *array* D' serán casi los mismos que los contenidos por el *array* utilizado en la iteración $i-1$, por lo tanto la suma $\sum_{d\in D'} v(d)$ para la iteración i será generalmente la misma que para $i-1$ más $v(ds[idx])$. Entonces, para evitar calcular cada vez dicha suma, la actualizamos siempre que se agregue o elimine un índice a D'.

El paso final del ciclo *while* es otro bucle anidado que se ejecuta siempre que $idx < 0$, lo cual puede ocurrir únicamente si para el conjunto de demandas almacenado en D', el algoritmo ya ha intentado agregar todas las otras demandas en *ds*. En ese caso el algoritmo remueve el último elemento agregado a D', *i.e.*, $D'.pop()$ e intenta agregar a *ds* el elemento anterior –si es que hay alguno–, caso contrario no habría más combinaciones y podría retornar $\bar{D}$. El ciclo es para eliminar código repetido, dado que el último elemento en D' podría ser el primero en *ds* y lo tiene que chequear cada vez.

Para cada tamaño de grupo $k \in [2, |D|]$ la cantidad de combinaciones de demandas que podemos hallar es $\binom{|D|}{k}$. Debido al *teorema del binomio*, el cual establece que

$$\sum_{k=0}^{n} \binom{n}{k} r^k = (1+r)^n,$$

para cada $n, k, r \in \mathbb{Z}_0^+$, y dado que $\binom{n}{0} = \binom{n}{1} = 1$, entonces la complejidad del algoritmo es

$$\mathcal{O}\left(|D|\log(|D|) + \sum_{k=2}^{n} \binom{|D|}{k}\right) \leq \mathcal{O}\left(2^{|D|}\right).$$

Por lo tanto, para evitar muchas iteraciones cuando los volúmenes son muy pequeños con respecto a $\bar{s}$, limitamos la cantidad total de grupos a ser agregados a una constante N, retornando D' cuando dicho límite es alcanzado. En los experimentos utilizamos $N = 10000$.

Finalmente, en order a reducir el tiempo computacional promedio del presente algoritmo, agregamos una poda al árbol de búsqueda. Generamos un *array* denominado $volsSum$ en el cual cada item i representa la suma de los volúmenes de las demandas que están desde la posición 0 a la $i-$ésima en *ds*. De este modo, se podan aquellos nodos para los cuales ni siquiera agregando todas las demandas restantes se puede alcanzar la capacidad del arco. La rutina para inicializar $volSum$ se muestra en el Algoritmo 7 y debe ser agregada antes del ciclo *while* del Algoritmo 6, mientras que la poda misma, observable en el Algoritmo 8, debe ser agregada al comienzo de dicho ciclo.

Algorithm 7 Inicialización de la poda. La variable *volsSum* almacena la lista de la suma máxima de volúmenes agregando las demandas de la lista ordenada *ds* desde la posición i-ésima hacia la izquierda.

```
1: volsSum ← new int[|D|]
2: acum ← 0
3: for d ∈ ds do
4:     acum += v(d);
5:     volsSum.add(acum);
6: end for
```

Algorithm 8 Poda: Cuando el resultado de agregar a D' las demandas de la sublista $ds[0 : idx]$ no excede la capacidad del arco, el algoritmo no intenta agregar idx, sino que remueve el último elemento de D' o retorna $\bar{D}$ si la lista está vacía.

```
1: if Σ_{d∈D'} v(d) + volSum[idx] ≤ s̄ then
2:     if |D'| > 0 then
3:         idx ← D'.pop() - 1
4:     end if
5: else
6:     return D̄
7: end if
```

Teniendo precalculadas las listas de demandas, los procedimientos de separación de desigualdades violadas para las familias mencionadas son los siguientes. Para separar las desigualdades de la familia *KDemandsDoNotExceedS*, para cada clave k del diccionario $\bar{D}$ el algoritmo itera sobre cada lista $ls \in \bar{D}[k]$ y cada arco $e \in E$ salteando la lista para la cual no toda demanda $d \in ls$ usa e. Luego precalcula $sum_d = \sum'_{s \in S} u^*_{des}$ para cada $d \in ls$, y la suma de estas sumas. Finalmente, para cada demanda $d \in ls$ revisa si la suma $sum_d - v(d)(\sum_{d' \in ls_d} v(d') - \sum_{d' \in ls_d} sum'_d) > \epsilon$, siendo $ls_d = ls \setminus \{d\}$. Ambas sumas son precalculadas para cada $d \in ls$, por lo que alcanza con restar los valores correspondientes a la actual demanda d. Iteramos sobre el diccionario completo, el cual está compuesto por a lo sumo N grupos de demandas, siendo N la constante que limita el número total de grupos agregados por el algoritmo anterior. Finalmente, con cada grupo de demandas del diccionario agregamos la suma total de sus volúmenes, con lo que la complejidad del procedimiento de separación para cada lista de demandas ls de esos N grupos es $\mathcal{O}(|E||ls|\bar{s})$.

Dado que la implementación del diccionario en *Java* es mediante un *HashMap*, el acceso a cada lista es realizado en $\mathcal{O}(1)$ en promedio, asumiendo suficientes *buckets* y que la función de *hash* dispersa uniformemente, pero requiere $\mathcal{O}(n)$ en el peor caso, por lo que la separación de las desigualdades violadas de la familia *twoDemandsDoNotExceedS* es ejecutada en $\mathcal{O}(|E|N\bar{s})$ en el peor escenario y en $\mathcal{O}(|E||ls_2|\bar{s})$ en promedio, donde ls_2 es la lista de pares de demandas que exceden $\bar{s}$.

La familia *KDemandsDoNotExceedSBySum* tiene el siguiente procedimiento de separación. Para cada lista de demandas ls del total de N grupos precomputados, calcula $\theta_{ls} = \sum_{d' \in ls} v(d') - \text{mín}_{d \in ls} v(d)$ e itera sobre cada arco buscando si

$$\sum_{s \in S} \sum_{d' \in ls} u^*_{d'es} - \theta_{ls} > \epsilon.$$

Precomputar la suma $\sum_{s\in S} u^*_{d'es}$ para cada $d \in D$ y $e \in E$ toma $\mathcal{O}(|D||E|\bar{s})$, y para cada grupo de demandas ls, la complejidad de esta rutina es $\mathcal{O}(|ls||E|)$. Por lo tanto, la complejidad temporal total es $\mathcal{O}(|D||E|\bar{s} + \sum_{ls\in D'} |ls||E|) = \mathcal{O}(|E|(|D|\bar{s} + \sum_{ls\in D'} |ls|))$. Entonces, las desigualdades de la subfamilia *twoDemandsDoNotExceedSBySum* son separadas en $\mathcal{O}(|E|(|D|\bar{s} + N))$.

4.1.19. No cabe a causa de la posición

Estas familias de desigualdades, motivadas por el hecho de que la posición de un *slot* utilizado por una demanda muchas veces imposibilita a otra demanda a utilizar una serie de *slots* sin un solapamiento, fueron implementadas sólo para el camino privado más simple, *i.e.*, un arco. Aquí presentamos el procedimiento de separación para cada una de dichas familias. Cada vez que una familia tenga una familia simétrica identificada explícitamente en el capítulo anterior, presentamos el caso para los *slots* más bajos. El procedimiento de separación de la familia simétrica es análogo.

posFitLow2DBySlots Con el objeto de hallar desigualdades violadas de esta familia, a saber,

$$u_{d_1es_1} + u_{d_2es_2} \leq 1 \qquad \begin{array}{r} \forall e \in E,\, \forall d_1, d_2 \in D,\, d_1 \neq d_2, \\ \forall s_2 \in \{2, \ldots, v(d_1)\},\, \forall s_1 \leq \text{máx}\{s_2, v(d_2)\}, \end{array}$$

para una solución fraccionaria u^*, el algoritmo busca un arco $e \in E$, dos demandas diferentes $d_1, d_2 \in D$, un *slot* $s_2 \in [1, v(d_1)]$ utilizado por la segunda demanda, *i.e.*, $u^*_{d_2es_2} > 0$, y un *slot* $s_1 \in [1, \text{mín}\{s_2 - 1, v(d_2)\}$, tales que $u^*_{d_1es_1} + u^*_{d_2es_2} > 1 + \epsilon$. La complejidad de este procedimiento es $\mathcal{O}(|D|^2|E|\bar{s}^2)$.

posFitLow2DBySlotsOneVsAll La diferencia del algoritmo que busca desigualdades violadas de esta familia, *i.e.*,

$$u_{d_1es_1} + \sum_{d'\in D\setminus\{d_1\}} u_{d'es_2} \leq 1 \qquad \begin{array}{r} \forall e \in E,\, \forall d_1 \in D,\, \forall s_2 \in \{2, \ldots, v(d_1)\}, \\ \forall s_1 \leq \text{máx}\{s_2, \text{mín}_{d'\in D\setminus\{d_1\}}(v(d'))\}, \end{array}$$

con el procedimiento que separa la familia anterior recae en el segundo término, para el cual aquí se realiza la suma de variables asociadas al conjunto de todas las otras demandas distintas a d_1, y los límites para el *slot* s_1. Con el fin de calcular la cota superior del *slot* s_1 para esta familia, el algoritmo busca el volumen mínimo de las demandas $d' \in D \setminus \{d_1\}$. Si preseleccionamos las dos demandas con volumen mínimo, entonces para cada demanda d_1 se requiere una sola comparación para hallar el mínimo. Por lo tanto, la complejidad total de este procedimiento está en el mismo orden que la del algoritmo anterior.

positionalFittingLower Para separar las desigualdades violadas de esta familia, a saber,

$$\sum_{s'=1}^{s_1} u_{d_1es'} \leq s_2(1 - u_{d_2,es}) \qquad \forall s \in \{1, \ldots, v(d_1)\},\, \forall e \in E,\, \forall d_1, d_2 \in D,\, d_1 \neq d_2,$$

donde $s_1 = s_2 = s$, el procedimiento, mostrado en los Algoritmos 9 y 10, se comporta como sigue para cada solución fraccionaria u^*. Por cada arco $e \in E$, selecciona una demanda

$d_2 \in D$ y busca un *slot* s comenzando desde el primer *slot* utilizado por d_2 en e –si existe alguno–, y recorriendo hasta $\max_{d \in D \setminus \{d_2\}} v(d)$. Para una solución fraccionaria, un *slot* se considera utilizado por una demanda en un arco si la variable asociada es mayor a 0. Entonces para cada s, se itera sobre toda otra demanda d_1 con volumen mayor o igual a s verificando si la diferencia de la suma de las variables relacionadas con los *slots* utilizados por d_1 en el rango $[1, s_1]$ y el término $s_2(1 - u^*_{d_2es})$ es mayor que ϵ, tomando $s_2 = s_1 = s$. La complejidad computacional es $\mathcal{O}(|D|^2|E|\bar{s})$ en el peor caso, sin embargo el desempeño puede ser mejorado en la práctica si se preordenan las demandas acorde a sus volúmenes.

Algorithm 9 Procedimiento que dada una solución u^* y un ϵ separa las desigualdades de la familia *positionalFittingLower* que son violadas por al menos ϵ.

```
1: procedure POSITIONALFITTINGLOWER(Float ε, Solution u*)
2:     for e ∈ E do
3:         for d2 ∈ D do
4:             low_d2 ← getLowestSlotUsed(d2, e, u*)
5:             if low_d2 > s̄ then
6:                 continue
7:             end if
8:             for d1 ∈ D \ {d} do
9:                 vd1 ← d1.volume()
10:                if low_d2 > vd1 then
11:                    continue
12:                end if
13:                for s ∈ [low_d2, vd1] do
14:                    if Σ_{s'=1}^{s} u*_{d1es'} + ε > s(1 − u*_{d2es}) then
15:                        addViolatedInequality(e, d1, d2, s)
16:                    end if
17:                end for
18:            end for
19:        end for
20:    end for
21: end procedure
```

Algorithm 10 Procedimiento que, dada una solución u^*, una demanda d, y un arco e, retorna el primer *slot* s tal que $u^*_{des} > 0$; en caso contrario devuelve $\bar{s} + 1$.

```
1: procedure GETLOWESTSLOTUSED(Demand d, Arc e, Solution u*)
2:     for low_d ∈ [1, s̄] do
3:         if u*_{de,low_d} > 0 then
4:             return low_d
5:         end if
6:     end for
7:     return s̄ + 1
8: end procedure
```

positionalFittingLowerTight Las diferencias con el algoritmo que separa la familia de desigualdades anterior radica en los valores para s_1 y s_2, siendo $s_1 = \text{máx}\{s, v(d_2)\}$ y $s_2 = \text{mín}\{v(d_1), s_1\}$. Por lo tanto, la complejidad temporal en el peor caso es la misma.

positionalFittingLowerOneVsAll El procedimiento implementado para separar las desigualdades violadas de esta familia es similar al algoritmo implementado para separar la familia anterior, excepto por el hecho de que para éste ahora se calcula la suma para cada demanda diferente a la d_2 seleccionada, y el valor de s_1, el cual aquí es el máximo entre s y el volumen mínimo de las otras demandas distintas a d_2. Nuevamente, con el fin de mejorar la complejidad computacional, el procedimiento precomputa las dos demandas con volumen mínimo al igual que el algoritmo que separa las desigualdades de *posFitLow2DBySlotsOneVsAll*.

posFitLow2DBySlots2SetsOfDs El procedimiento de separación de estas igualdades es levemente diferente de los otros. Esta vez debemos dividir las demandas en dos grupos de acuerdo a sus volúmenes. Para cada *slot* s_2, por un lado necesitamos agrupar todas las demandas con volúmenes menores a s_2. Este grupo es definido como $D^{<}_{s_2}$ en las desigualdades de esta familia, a saber,

$$\sum_{d' \in D^{\geq}_{s_2}} u_{d' e s_1} + \sum_{d' \in D^{<}_{s_2}} u_{d' e s_2} \leq 1 \qquad \begin{array}{r} \forall e \in E,\, \forall s_1 \in \{1, \ldots, s_2 - 1\}, \\ \forall s_2 \in \{2, \ldots, \text{máx}_{d \in D}\, v(d)\}, \end{array}$$

mientras que las demandas con volumen mayor a s_2 se definen como $D^{\geq}_{s_2}$. Como el *slot* s_2 varía en el rango $[2, \text{máx}_{d \in D}\, v(d)]$, para optimizar los tiempos computacionales, preordenamos las demandas en orden creciente y utilizamos un índice para marcar el límite entre ambos grupos de demandas. Para cada arco e dicho índice *idx* comienza en la última posición del *array* de demandas, *i.e.*, apuntando a la demanda con mayor volumen, y es movido hacia la izquierda para cada *slot* s_2 buscando a la primera demanda con volumen menor que s_2. Luego se calcula la suma de los valores de las variables relacionadas con e, s_2 y a las demandas del grupo de la izquierda del índice, *i.e.*, las demandas con volumen menor a s_2, y finalmente se busca otro *slot* s_2 en el rango $[1, s_2 - 1]$ tal que la suma anterior agregada $\sum_{d' \in D^{\geq}_{s_2}} u^*_{d' e s_2}$ sea mayor a $1 + \epsilon$. En ese caso se halló una desigualdad violada. Como s_2 toma valores en su rango en orden decreciente, cuando ninguna demanda es menor a s_2, no hay más desigualdades violadas en el arco e. El pseudocódigo se muestra en el Algoritmo 11. Aquí nuevamente *ds* contiene la lista de demandas ordenadas en orden creciente según sus volúmenes, lo cual es calculado en $\mathcal{O}(|D| \log(|D|))$ por la función *getSortedDemands* como en el Algoritmo 6. La complejidad computacional total del procedimiento es $\mathcal{O}(|E|\bar{s}\ (|D| + \bar{s}))$ en el peor caso; sin embargo el rendimiento en la práctica mejora cuando el volumen de las demandas es mucho menor que $\bar{s}$.

posFit3DBySlots Para separar las desigualdades violadas pertenecientes a esta familia, el algoritmo implementado busca un arco $e \in E$, tres demandas $d_1, d_2, d_3 \in D$ y tres *slots* $s_1, s_2, s_3 \in S$ tales que $v(d_1) \leq s_1 < s_2 < s_3 \leq \bar{s} - v(d_3) + 1$ y $s_3 - s_1 \leq v(d_2)$ y $u^*_{d_1, e, s_1} + u^*_{d_2 e s_2} + u^*_{d_3, e, s_3} > 2 + \epsilon$. La complejidad de este procedimiento es $\mathcal{O}(|D|^3 ||E| \bar{s}^3)$.

posFit3DBySum Dada una solución fraccionaria u^*, el procedimiento de separación que encuentra desigualdades violadas de esta familia –esto es, la familia dada por las

Algorithm 11 Procedimiento que dada una solución u^* y un ϵ separa las desigualdades de la familia *posFitLow2DBySlots2SetsOfDs* que son violadas por al menos ϵ.

```
1: procedure posFitLow2DBySlots2SetsOfDs(Float ε, Solution u*)
2:     ds : getSortedDemands()
3:     idx ← ds[0]
4:     for e ∈ E do
5:         for s2 ∈ [ds.last().volume(), 1] do
6:             while idx ≥ 0 y  ds[idx].volume() ≥ s2 do
7:                 idx ← idx − 1
8:             end while
9:             if idx < 0 then
10:                break
11:            end if
12:            sum2 ← Σ_{d'∈D^≤_{s2}} u*_{d'es2}
13:            if sum2 = 0 then
14:                continue
15:            end if
16:            for s1 ∈ [1, s2] do
17:                sum1 ← Σ_{d'∈D^≥_{s2}} u*_{d'es1}
18:                if sum1 + sum2 > 1 + ε then
19:                    addViolatedInequality(e, ds, idx, s1, s2)
20:                end if
21:            end for
22:        end for
23:    end for
24: end procedure
```

desigualdades

$$\sum_{d' \neq d_2} u_{d',e,s_1} + u_{d_2 e s_2} + \sum_{d' \neq d_2} u_{d',e,s_3} \leq 2 \qquad \forall e \in E, \forall d_2 \in D,$$

con s_1, s_2, s_3 tres *slots* tales que $\min_{d' \in D \setminus d_2} v(d') \leq s_1 < s_2 < s_3 \leq \bar{s} + 1 - \min_{d' \in D \setminus d_2} v(d')$, y $s_3 - s_1 \leq v(d_2)$– itera sobre cada arco $e \in E$ y cada demanda $d \in D$, buscando un *slot* $s_1 \in [1, \bar{s} - 1]$ tal que $sum_1 = \sum_{d' \in D \setminus \{d\}} u^*_{d' e s_1} > 0$, cuando lo halla busca otro *slot* $s_3 \in [s_1 + 2, \bar{s}]$ tal que $sum_3 = \sum_{d' \in D \setminus \{d\}} u^*_{d' e s_3} > 0$, y finalmente busca un tercer *slot* $s_2 \in [s_1 + 1, s_3 - 1]$ tal que $sum_1 + u^*_{d e s_2} + sum_3 > 2 + \epsilon$. Para reducir el tiempo de ejecución, una vez que se selecciona una demanda d, generamos un *array* para almacenar la suma sobre las otras demandas en cada *slot* con el fin de evitar calcularlas muchas veces. Otra mejora para reducir el tiempo de separación es que salteamos cuando la demanda d no usa el arco o cuando las otras sumas no son mayores a 0. Este algoritmo ejecuta en $\mathcal{O}(|D||E|\bar{s}^3)$.

posFit3DByImplication Dado que esta familia es una reformulación de la anterior, la única diferencia en sus procedimientos de separación radica en que en lugar de usar directamente $u^*_{d e s_2}$ ahora se calcula $sum_2 = \sum_{s' \geq s_1 + 1}^{s_3 - 1} u^*_{d e s'}$ y se reescribe la comparación

como $sum_2 - (s_3 - s_1 - 1)\ (2 - sum_1 - sum_3) > \epsilon$. Por lo tanto, la complejidad de este algoritmo en el peor caso es similar a la complejidad del que separa la familia previa.

4.1.20. *Slots* centrales con un límite fijo

Dada una solución fraccionaria u^*, el algoritmo implementado para separar las desigualdades violadas de la familia *centralSsBetweenLowBounds*, *i.e.*,

$$\sum_{s' \in Y} u_{des'} \geq |Y| \Big(\sum_{d' \in D \setminus \{d\}} u_{d'es_2} + u_{des_1} - 1 \Big) \qquad \forall d \in D,\ \forall e \in E,$$

con $s_2 \in \{v(d)+1, \ldots, 2v(d)\}$, $s_1 < s_2$, e $Y = \{s_2 - v(d), \ldots, v(d)\}$, itera sobre cada arco $e \in E$ y cada demanda $d \in D$ tales que el primer *slot* utilizado por d en e es menor o igual a $v(d)$. Luego, para cada *slot* $s_2 \in [vd, \text{mín}\{2v(d) - 1, \bar{s} - 1\}]$, calcula $sum_2 = \sum_{d' \neq d} u^*_{d'es_2}$, y para cada una que resulta mayor a cero, el algoritmo calcula la suma de las variables relacionadas con d y con los *slots* centrales, *i.e.*, $sum_d = \sum_{s' \in Y} u^*_{des'}$ con $Y = [s_2 - v(d), v(d)]$. Finalmente busca un *slot* s_1 desde el primer *slot* utilizado por d hasta s_2, tal que $\epsilon < |Y|(sum_2 + u^*_{des_1} - 1) - sum_d$. De este modo, cuando ϵ es mayor a 0, algunas desigualdades no se consideran como violadas. Este procedimiento ejecuta en $\mathcal{O}(|D||E|\bar{s}\ (|D| + \bar{s}))$.

4.1.21. *Slots* centrales entre demandas

La familia *centralSsBetweenDs* es muy parecida a la anterior, pero agregando una demanda como cota inferior para los *slots* centrales, es decir,

$$\sum_{s' \in Y} u_{des'} \geq |Y| \Big(\sum_{d' \in D \setminus \{d\}} u_{d'es_1} + \sum_{d' \in D \setminus \{d\}} u_{d'es_3} + u_{des_2} - 2 \Big) \qquad \forall d \in D,\ \forall e \in E,$$

donde $s_1 \in \{1, \ldots, \bar{s} - 2v(d)\}$, $s_3 \in \{s_1 + v(d) + 1, \ldots, s_1 + 2v(d)\}$, $s_1 < s_2 < s_3$, e $Y = \{s_3 - v(d), \ldots, s_1 + v(d)\}$. Para una solución fraccionaria u^* el procedimiento de separación itera sobre cada arco $e \in E$ y cada demanda $d \in D$ tales que el primer *slot* usado por d en e es menor o igual a $v(d)$. Luego calcula $sum_1 = \sum_{d' \neq d} u^*_{d'es_1}$ para cada $s_1 \in [1,\ \bar{s} - v(d) - 1]$. Cada vez que dicha suma es mayor a 0, el algoritmo también calcula $sum_3 = \sum_{d' \neq d} u^*_{d'es_3}$ para cada $s_3 \in [s_1 + v(d) + 1, \text{mín}\{s_1 + 2v(d), \bar{s}\}]$, la cual va a reemplazar a sum_2 en el algoritmo previo. Cada vez que esta suma resulte en un valor positivo, calcula el intervalo Y como $Y = [s_3 - v(d), s_1 + v(d)]$ y la suma de las variables asociadas a d y a los *slots* pertenecientes a dicho rango, al igual que con la familia anterior. Finalmente, el procedimiento busca cada $s_2 \in [\text{máx}\{s_1, low_d\}, s3]$, con low_d el primer *slot* usado por la demanda d en e, tal que $\epsilon < |Y|(sum_1 + sum_3 + u^*_{des_2} - 2) - sum_d$. La complejidad de este procedimento, el cual es mostrado en el Algoritmo 12 (donde *getLowestSlotUsed* es la función que retorna el primer *slot* utilizado por una demanda dada en un arco particular, *i.e.*, lo detallado en el Algoritmo 10), es peor que la complejidad del algoritmo anterior porque aquí aparece una nueva iteración sobre los *slots* y sobre las demandas, a pesar de que mejora en el caso promedio cuando los volúmenes son mucho menores a $\bar{s}$ si se almacenan los resultados de las sumas para evitar recomputarlos. La complejidad resultante es $\mathcal{O}(|D||E|\bar{s}(|D|+\bar{s}(|D|+\bar{s})))$. Para mejorarla, podríamos calcular para cada arco, la suma de las variables utilizadas por todas las demandas en cada *slot* en S, y luego en cada iteración restar u^*_{des}, reduciendo así la complejidad a $\mathcal{O}(|D||E|(\bar{s}+\bar{s}^3))$. Otra posibilidad es utilizar programación dinámica.

Algorithm 12 Procedimiento que dada una solución u^* y un ϵ separa las desigualdades de la familia *centralSsBetweenDs* que son violadas por al menos ϵ.

```
1: procedure centralSsBetweenDs(Float ε, Solution u*)
2:     for e ∈ E do
3:         for d ∈ D do
4:             vd ← d.volume()
5:             low_d ← getLowestSlotUsed(d, e, u*)
6:             if low_d > vd then
7:                 continue
8:             end if
9:             for s_1 ∈ [1, s̄ − vd − 1] do
10:                sum_1 ← Σ_{d'∈D\{d}} u*_{d'es_1}
11:                if sum_1 = 0 then
12:                    continue
13:                end if
14:                for s_3 ∈ [s1 + vd + 1, mín{s_1 + 2vd, s̄}] do
15:                    sum_3 ← Σ_{d'∈D\{d}} u*_{d'es_3}
16:                    if sum_3 = 0 then
17:                        continue
18:                    end if
19:                    Y ← [s_3 − vd, s_1 + vd]
20:                    sum_d ← Σ_{s'∈Y} u*_{des'}
21:                    for s_2 ∈ [s_1, s_3] do
22:                        if sum_d + ε < |Y|(sum_1 + sum_3 + u*_{des_2} − 2) then
23:                            addViolatedInequality(e, d, s_1, s_2, s_3)
24:                        end if
25:                    end for
26:                end for
27:            end for
28:        end for
29:    end for
30: end procedure
```

4.2. Estrategias de selección

Para gestionar los procedimientos de separación presentados en la sección anterior, evaluamos diferentes estrategias. En algunas de ellas, empleamos un *coeficiente de efectividad* φ para cada procedimiento, definido como el número de cortes generados dividido por el número de llamadas al procedimiento. Cuando se calibra ϵ, el coeficiente φ proporciona una forma de comparar el comportamiento de las familias. Las diferentes estrategias implementadas son las siguientes:

- *Brute Force* [BRUTE FORCE]: Ejecuta todos los procedimientos de separación.
- *Random* [RND]: Mezcla la lista de procedimientos e itera sobre ésta hasta hallar al menos un corte de h familias diferentes, o hasta llegar al final de la lista.
- *Most Effective* [EFF]: Ordena la lista de procedimientos de acuerdo con sus coefi-

cientes de efectividad, e itera sobre dicha lista al igual que con la estrategia *Random*.

- *Most Effective With Random* [EFF&RND]: Itera sobre la lista ordenada de procedimientos al igual que en *Most Effective* pero, de forma aleatoria y con una probablidad predefinida, llama a uno de los procedimientos aún no ejecutados.
- *Weighted Selection* [WEIGHTED]: Itera sobre la lista ordenada de procedimientos y ejecuta cada uno con una probabilidad calculada en función de su coeficiente de efectividad.

Las estrategias *Most Effective* y *Most Effective With Random* también contemplan la variación de preordenar la lista de procedimientos en función de los resultados de experimentos anteriores, mientras que las otras estrategias, incluída *Weighted Selection*, comienzan con un orden aleatorio. Con el objeto de forzar a todos los procedimientos a ejecutar al menos una vez, el valor inicial del coeficiente φ es infinito para cada procedimiento en todas las estrategias que lo usan, *i.e.*, en *Most Effective*, *Most Effective With Random* y *Weighted Selection*. Dado que este coeficiente φ –diferente para cada familia– depende del número acumulado de cortes generados por procedimiento y del número de llamados a cada uno de ellos, es actualizado en cada iteración resultando en un valor mayor para aquellos procedimientos que en promedio generan una mayor cantidad de cortes por iteración. Todas las estrategias, excepto *Brute Force*, tienen un parámetro h que indica la cantidad de familias de las cuales tomar los cortes. Para mantener la lista de procedimientos actualizada, antes de retornar del *callback*, el algoritmo itera sobre aquellos que fueron llamados y los reposiciona. Este proceso toma $\mathcal{O}(|cp|\ |tp|)$ siendo $|cp|$ y $|lp|$ la cantidad de procedimientos llamados y la cantidad total de ellos, respectivamente. A pesar de que estos valores pueden llegar hasta 60 en nuestro caso, el tiempo total requerido por este algorimo de ordenamiento es despreciable en comparación con el requerido por la ejecución de los procedimientos de separación, como vimos en la sección anterior.

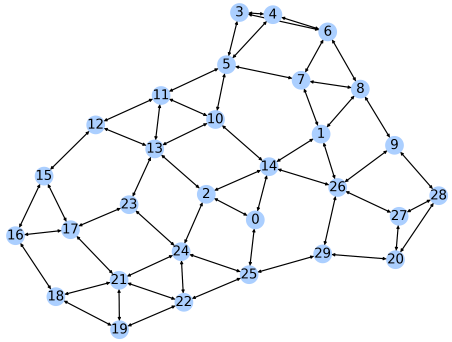
4.3. Resultados computacionales

Presentamos ahora nuestra experiencia computacional para evaluar la efectividad en la práctica de las familias de desigualdades presentadas en el Capítulo 3. La implementación fue realizada utilizando el entorno Cplex 12.10, y los experimentos fueron llevados a cabo en una computadora con un CPU Intel(R) Xeon(TM) 2.80GHz con 4 GB de memoria RAM. Excepto para algunos experimentos –que mencionamos explícitamente– tanto para el branch-and-cut como para el branch-and-bound desactivamos todas las heurísticas primales, el *pre-solver* y la paralelización de Cplex.

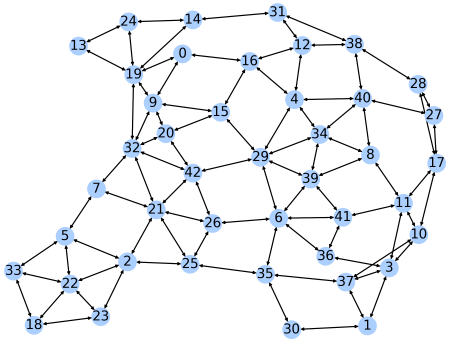
4.3.1. Instancias

Dado que no hay instancias completas disponibles en la literatura, sino sólo algunas topologías y parámetros, y siendo una práctica aún muy común en el estudio de RSA y sus problemas asociados el construir instancias propias, decidimos implementar un *script* generador de instancias, el cual está disponible en [16] con el fin de que también pueda ser utilizado en trabajos futuros y así contribuir a formar un *benchmark* estándar. El conjunto de instancias utilizado para comparar las estrategias de branch-and-cut y luego la heurística presentada en el siguiente capítulo se generó mediante este *script* basado en la literatura (ver el Apéndice 9 para obtener los detalles completos).

Se utilizaron 19 topologías reales, con $|V| \in \{6, \ldots, 43\}$ y $|E| \in \{9, \ldots, 176\}$. Las dos más grandes de ellas, conocidas como *30n-112m-Spain* y *43n-176m-EuroLarge*, se pueden ver en las Figuras 4.4a y 4.4b, respectivamente. El número $\bar{s}$ de *slots* disponibles depende de la instancia, y va desde 5 hasta 200. Las demandas son generadas de forma aleatoria con distribución uniforme, con volúmenes en el rango de 1 a 124 *slots*.



(a) 30n-112m-Spain



(b) 43n-176m-EuroLarge

Fig. 4.4: Dos topologías extraídas de la literatura.

Este conjunto de 100 instancias así generadas es suficientemente difícil como para realizar los experimentos con el *branch-and-cut*. Sin embargo, al agregar las heurísticas primales que presentamos en el siguiente capítulo, el tiempo necesario para resolverlas disminuye notablemente. Analizando las soluciones obtenidas observamos que la suma de los volúmenes de las demandas utilizadas en cada arco era muy inferior a $\bar{s}$, por lo que en muchas de ellas incluso era factible asignar a cada una un camino mínimo. Este enfoque casi reduciría el problema al SA, como explicaremos en el capítulo dedicado a las heurísticas. Por lo tanto, decidimos comparar las instancias en función de su *densidad*, además del número de variables y restricciones. Esto nos llevó a la necesidad de aclarar el concepto de *densidad*. A continuación describimos tres definiciones diferentes de este término usadas en el presente trabajo.

1. **Arcs-density**: Porcentaje de enlaces de la topología sobre el total posible en un grafo completo. Se utilizó este concepto en el Capítulo 2 para comparar los distintos modelos. También es conocido como *densidad de un grafo* Δ, y depende únicamente de la topología. Se calcula como

$$\Delta = \frac{|E|}{|V| \times (|V| - 1)}.$$

2. **Demands-Density**: Relación entre el máximo volumen requerido por las demandas y la capacidad de cada arco. Sea $\mathcal{I} = (G, D, \bar{s})$ una instancia del RSA. Definimos

$$maxSD_{\mathcal{I}} = \max_{d \in D} v(d),$$

como el máximo volumen requerido por todas las demandas para la instancia $\mathcal{I}$ dada. Entonces esta segunda densidad se define como

$$\bar{\Delta}_{\mathcal{I}} = \frac{maxSD_{\mathcal{I}}}{\bar{s}}.$$

3. ***Slots*-density**: Cantidad de *slots* utilizados por la *mejor solución posible* en relación al total de *slots* disponibles. Dado que para la mayoría de las instancias es imposible en la práctica tener la *mejor solución posible*, cuando dicho valor no está disponible, estimamos con una cota la cantidad de *slots* necesarios. Estudiando las soluciones disponibles, observamos que un buen estimador de este parámetro es el mínimo número de *slots* que puede tener una solución factible, esto es, la cantidad de *slots* utilizados por una solución en la cual a cada demanda se le asigna el camino mínimo, asumiendo suficientes *slots* en cada arco.

A nuestro leal saber y entender, la tercera definición se propone por primera vez en el presente trabajo con el fin de medir la dificultad para resolver una instancia. El generador de instancias que desarrollamos utiliza sólo las dos primeras definiciones, es decir, *arcs-density* y *demands-density*. Dado que *arcs-density* depende de las topologías, y éstas vienen dadas, sólo podemos establecer *demands-density*. Por lo tanto, en orden a generar cada instancia, el *script*, junto con el valor de $\bar{s}$ y la topología $G = (V, E)$, recibe un parámetro real $p \in (0, 1]$ utilizado para calcular $maxSD = p \times \bar{s}$, de modo tal que cada demanda utiliza una cantidad aleatoria de *slots* en el rango $[\frac{1}{2}maxSD,\ maxSD]$, y un factor opcional f para limitar la cantidad de demandas, a saber,

$$|D| = \frac{f \times (|V| - 1) \times \Delta \times \bar{s}}{maxSD},$$

con Δ la *arc-density* del grafo G y $f = 2$ por defecto.

Para los experimentos pertenecientes a éste y los capítulos siguientes generamos seis conjuntos diferentes con distintos rangos de densidades, lo cual resultó en un total de 549 instancias basadas en 19 topologías. Son los siguientes.

Low-density: 100 instancias con *slot-density* menor al 20 % usando 19 topologías de entre 6 y 43 nodos, con $\bar{s}$ hasta 200 en la menor de ellas y hasta 150 en las otras, y utilizando hasta 236 demandas cada una.

Lowest-density: 22 instancias extraídas del conjunto *Low-density*, utilizando únicamente 11 topologías, con hasta 100 *slots* por arco, y 51 demandas.

Medium-density: 160 instancias usando cinco topologías, desde 6 a 20 nodos, con $\bar{s}$ hasta 400 y $|D|$ menor a 118. La *slot-density* de este conjunto es cercana al 40 %.

Sub-Med-density: Un subconjunto conformado por 65 instancias de *Medium-density* utilizando sólo cuatro topologías y hasta 75 demandas cada una.

Full-range-density: 250 instancias, con *slot-density* en el rango completo, muchas de ellas requiriendo hasta el 100 % del total de *slots* (*i.e.*, probablemente no factibles), con $\bar{s}$ menor a 400 y $|D|$ hasta 178.

High-Density: 39 instancias con *slot-density* entre el 50 % y el 100 %, utilizando siete topologías con $\bar{s}$ hasta 200 y a lo sumo 234 demandas.

Topología	Lowest			Low			Sub-Med			Medium			High			Full-Range		
	$\bar{s}$	$\|D\|$	ζ	$\bar{s}$	$\|D\|$	ζ	$\bar{s}$	$\|D\|$	ζ	$\bar{s}$	$\|D\|$	ζ	$\bar{s}$	$\|D\|$	ζ	$\bar{s}$	$\|D\|$	ζ
6n-9m-n6s9	[5, 80]	[2, 20]	[15, 100]	[5, 200]	[2, 124]	[4, 100]	[10, 400]	[11, 43]	[40, 100]	[10, 400]	[11, 74]	[40, 100]	-	-	-	[10, 400]	[31, 124]	[40]
10n-44m-SmallNet	[20, 100]	[7, 23]	[20, 100]	[20, 150]	[7, 23]	[20, 100]	[15, 80]	[22, 75]	[50, 53]	[15, 100]	[20, 118]	[31, 53]	-	-	-	-	-	-
11n-52m-COST239	[20, 30]	[35, 42]	[20, 20]	[20, 150]	[11, 236]	[3, 80]	[80, 150]	[29, 43]	[50]	[80, 200]	[29, 43]	[50]	-	-	-	[10, 200]	[34, 163]	[40]
14n-42m-NSF	[20, 60]	[11, 28]	[20, 42]	[20, 150]	[5, 28]	[20, 100]	-	-	-	[80, 150]	[17, 30]	[50]	[10, 200]	[42, 93]	[40]	-	-	-
14n-46m-DT	[15, 40]	[12, 43]	[13, 30]	[15, 150]	[5, 43]	[13, 80]	-	-	-	-	-	-	-	-	-	[10, 150]	[29, 97]	[40]
15n-46m-NSF	-	-	-	[60, 150]	[9, 44]	[8, 53]	-	-	-	-	-	-	-	-	-	-	-	-
16n-46m-EURO	[15]	[10]	[53]	[15, 150]	[3, 22]	[25, 80]	-	-	-	-	-	-	-	-	-	-	-	-
19n-76m-EON19	[20, 30]	[9, 51]	[13, 50]	[20, 150]	[7, 51]	[13, 53]	-	-	-	-	-	-	-	-	-	-	-	-
20n-62m-ARPANet	[80]	[6]	[100]	[40, 150]	[6, 71]	[8, 100]	[10, 100]	[20, 64]	[40]	[10, 150]	[15, 70]	[40]	[10, 20]	[55, 57]	[40]	[10, 200]	[22, 103]	[40]
20n-78m-EON20	-	-	-	[60, 150]	[8, 50]	[12, 53]	-	-	-	-	-	-	-	-	-	-	-	-
21n-70m-Spain	-	-	-	[60, 150]	[13, 48]	[13, 30]	-	-	-	-	-	-	-	-	-	-	-	-
21n-72m-Italian	[30]	[18]	[33]	[30, 150]	[4, 18]	[33, 83]	-	-	-	-	-	-	-	-	-	-	-	-
21n-78m-UKNet	-	-	-	[60, 150]	[6, 20]	[30, 83]	-	-	-	-	-	-	-	-	-	-	-	-
22n-70m-BT	[20]	[22]	[20]	[20, 150]	[19, 150]	[3, 25]	-	-	-	-	-	-	-	-	-	-	-	-
24n-86m-UBN24	-	-	-	[40, 150]	[8, 19]	[33, 80]	-	-	-	-	-	-	-	-	-	[10, 100]	[25, 119]	[40]
28n-68m-EON	-	-	-	[80, 150]	[3, 17]	[20, 100]	-	-	-	-	-	-	-	-	-	-	-	-
28n-82m-EURO28	-	-	-	[60, 150]	[5, 17]	[17, 100]	-	-	-	-	-	-	[10, 150]	[49, 112]	[40]	-	-	-
30n-112m-Spain	[15]	[8]	[67]	[15, 150]	[5, 163]	[4, 83]	-	-	-	-	-	-	-	-	-	[10, 150]	[51, 157]	[40]
43n-176m-Euro	-	-	-	[10, 150]	[5, 47]	[13, 80]	-	-	-	-	-	-	[60, 150]	[130, 234]	[40]	[10, 100]	[31, 178]	[40]

Tab. 4.1: Topologías y rangos de los parámetros según el conjunto de instancias.

La Tabla 4.1 sumariza los parámetros utilizados para los conjuntos de instancias mencionados. Sea I un conjunto de instancias, el porcentaje que *maxSD* representa de $\bar{s}$ para cada conjunto de instancias I pertenece al rango

$$\zeta^I = \left[\min_{i \in I} \left(100 \frac{maxSD_i}{\bar{s}_i} \right), \max_{i \in I} \left((100 \frac{maxSD_i}{\bar{s}_i} \right) \right].$$

4.3.2. Coeficiente τ

Para comparar los resultados definimos un coeficiente τ para cada ejecución del siguiente modo. Sea t el tiempo de ejecución en minutos, sea $g \in [0, 1]$ el gap de optimalidad normalizado, y sea $p = t/4$ un término de penalización. El coeficiente τ se define como

$$\tau = \begin{cases} t & \text{si la instancia fue resuelta antes del tiempo límite,} \\ t + p + g\ p & \text{si la instancia no fue resuelta dentro del tiempo límite pero se halló una solución factible,} \\ t + 2p & \text{si no se halló ninguna solución factible dentro del límite de tiempo.} \end{cases}$$

De este modo, penalizamos la falta de certidumbre cuando se alcanza el límite, con una mayor penalización si no se halló ninguna solución factible. Cuanto menor sea el valor de τ, la solución será considerada mejor. El coeficiente τ total de múltiples ejecuciones es calculado como la suma de los mejores coeficientes de cada ejecución particular.

Para las siguientes secciones, cada experimento es ejecutado al menos dos veces (tres veces para la calibración de los parámetros) seleccionando el mejor resultado.

4.3.3. Selección de las familias de cortes optimales y desigualdades e igualdades válidas más efectivas

Con el fin de calibrar el parámetro ϵ para cada procedimiento, hemos seleccionado el conjunto de instancias *Lowest-density*. A causa de la gran cantidad de experimentos, el tiempo de ejecución fue limitado a 4 minutos. Para cada familia i experimentamos con $\epsilon \in \{0, 0.1, \ldots, M_i\}$, siendo M_i un valor diferente para cada familia i, que hace que ningún corte sea agregado (o solamente algunos pocos para ciertos procedimientos particulares. Típicamente $M_f = 4$). Para cada uno de los valores de ϵ calculamos el coeficiente τ para el mejor resultado de cada instancia.

Para medir el desempeño de cada familia de acuerdo con cada ϵ seleccionado reportamos la suma de los coeficientes τ sobre todas las instancias, resultando en un único coeficiente para cada combinación. La Tabla 4.2 muestra una parte de estos resultados para gran parte de las familias con mejor rendimiento. Es interesante notar que los mejores resultados fueron obtenidos con valores pequeños para ϵ, lo cual sugiere que estos cortes son de hecho efectivos. La Figura 4.6 muestra esta situación gráficamente con dos de las familias de mejor rendimiento. Puede observarse, no sólo la relación directa entre la cantidad de cortes agregados por el algoritmo y el coeficiente de efectividad, sino también la relación inversa entre éstos y el coeficiente τ. Dado que la mayoría de los procedimientos se comporta así, estos resultados no sólo sugieren que los cortes son efectivos como se mencionó, sino que también soportan la elección de este coeficiente de efectividad como parámetro para medir la efectividad de una familia de cortes.

Familia \ ϵ	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
contiguityIneqs	6.99	9.38	13.14	8.98	15.27	18.27	23.59	36.45	30.75	31.36
symmContiguityIneqs	9.58	8.25	15.62	18.39	15.97	17.01	15.67	20.54	22.37	22.32
symmetricalBFcontiguity	20.11	21.20	18.00	17.80	18.07	24.63	23.47	20.51	20.90	22.87
notBranchFromSrc	27.90	28.04	24.21	26.57	31.08	25.60	24.44	31.02	19.59	29.49
farSlotsOff	26.43	25.87	29.05	23.15	23.61	23.98	29.95	27.14	31.22	23.81
ppalSlotsToDst	24.17	25.04	24.21	24.66	26.53	25.83	23.28	23.91	23.82	23.56
ppalSlotsFromSrc	25.34	25.00	24.77	23.78	25.77	24.24	25.06	27.05	23.56	25.09
contiguityEqs	25.57	34.98	26.61	23.86	31.36	30.97	25.62	37.81	39.13	23.90
exactlyVdFromSrc	26.00	27.65	49.47	73.94	53.16	26.74	45.48	27.05	28.45	31.34
notBranchFromV	29.41	32.05	37.49	26.90	38.96	29.21	31.93	30.17	30.55	30.48
onePpalSlotsFromSrc	38.43	32.15	27.62	36.98	33.86	35.53	27.35	29.66	35.32	46.84
exactlyVdFromV	36.56	36.45	36.23	37.37	37.33	37.57	43.42	33.28	39.17	36.99
positionalFittingLowerTight	43.30	38.16	42.61	38.39	43.13	36.54	37.41	32.68	31.03	32.62
highPositionForces	34.33	35.94	34.51	36.85	35.69	35.51	36.70	36.56	33.00	29.31
contiguityASCC	38.47	33.86	37.00	34.21	29.45	38.74	38.14	39.20	41.94	43.25
outcomingDBrooms	43.56	32.18	35.29	35.41	35.78	35.86	36.23	36.58	38.71	40.89
positionalFittingGreaterTight	34.10	34.84	31.69	32.06	31.30	37.78	47.21	44.11	41.54	41.25
incomingDBrooms	35.74	33.50	32.59	32.58	32.45	32.72	39.11	31.68	31.51	31.59
lowPositionForces	41.77	37.62	43.94	35.05	46.03	36.71	45.11	44.24	43.91	37.38
centralSsBetweenBounds	39.71	46.10	41.02	46.17	44.18	43.62	41.86	41.47	32.17	41.08
simplestCycles	33.60	33.92	35.10	35.85	36.06	39.23	34.32	34.95	35.18	37.75
positionalFittingLower	34.20	35.61	33.04	37.69	38.98	39.28	43.18	45.69	44.91	45.88
eqAmountOfAsForTheSumOfSs	34.34	33.17	36.91	36.22	40.56	38.07	39.84	37.68	33.61	37.13
posFitHigh2DBySlotsOneVsAll	38.54	33.70	43.32	36.77	46.67	37.49	44.39	40.47	48.80	47.36
centralSsBetweenHighBounds	49.11	41.51	40.20	40.78	44.90	44.01	45.76	45.28	34.85	42.53
eqAmountOfAsForEachUsedS	43.70	40.07	38.11	36.13	34.66	38.04	36.98	35.94	40.87	34.52
posFit3DByImplication	38.65	35.27	40.80	40.93	42.39	40.36	40.37	39.93	39.57	39.70
simplestCyclesImplication	36.78	36.39	35.28	40.71	39.05	43.94	44.35	41.04	40.79	43.99
oneSlotOnceFromV	46.50	36.54	36.15	41.15	42.48	43.31	42.86	41.68	42.61	40.94
positionalFittingLowerOneVsAll	42.83	41.69	37.30	35.51	43.22	41.71	39.43	36.74	36.45	36.35
centralSsBetweenLowBounds	40.90	46.92	41.65	36.17	44.88	41.58	46.13	42.97	38.97	43.54
nonOverFixNextHiBySum	36.06	39.18	43.62	41.07	42.33	42.10	44.89	39.17	40.37	45.06
positionalFittingGreaterOneVsAll	36.36	41.09	39.05	38.59	38.61	36.10	40.82	44.37	43.44	43.34
oneSlotOnceToDst	42.52	42.62	36.78	41.47	40.32	41.74	42.01	41.79	44.09	43.27
posFitLow2DBySlotsOneVsAll	37.88	40.69	44.85	40.43	37.00	37.13	41.78	37.10	43.18	42.98
posFit3DBySum	42.17	41.03	46.44	37.56	38.96	46.87	48.27	43.46	42.09	42.83
oneSlotOnceToV	42.60	43.02	38.06	39.31	41.36	40.85	43.97	44.16	44.25	41.17
nonOverBySum	38.23	47.89	39.29	44.88	41.48	39.88	40.99	39.08	40.12	38.48
nothingFromDst	42.85	43.71	43.68	43.69	38.47	38.25	38.23	38.57	40.94	40.97
positionalFittingGreater	39.90	41.07	41.55	41.15	38.32	38.27	40.54	45.65	44.70	44.99
nonOverFixNextLowBySum	46.47	48.16	40.35	44.42	42.78	38.51	39.59	46.79	42.66	39.06
oneSlotOnceFromSrc	40.93	44.09	42.95	46.60	50.08	46.64	46.50	40.31	40.34	42.08
simplestCyclesBySlot	41.06	41.81	41.72	41.40	41.03	44.00	43.97	44.00	40.83	40.80
posFitHigh2DBySlots	51.09	48.11	49.03	48.34	58.36	50.15	46.46	42.18	51.03	62.02
centralSlotsFromSourceBySlot	47.50	47.50	48.58	48.54	48.55	48.85	45.67	48.82	42.51	42.21
twoDemandsDoNotExceedS	44.28	43.61	42.60	42.61	42.65	42.67	42.62	42.63	42.63	42.67
twoDemandsDoNotExceedSBySum	48.06	49.45	46.42	49.46	49.43	49.45	43.28	43.31	43.34	43.29
cyclesBySlot	51.67	51.56	48.05	50.28	44.30	47.28	52.45	46.15	47.20	49.25
nonOverFixNextHi	43.73	52.38	58.02	57.26	47.49	50.19	50.21	47.92	52.61	52.60
centralSlotsFromSource	44.38	47.49	48.54	48.50	48.49	48.51	48.51	48.51	48.53	47.93

Tab. 4.2: Coeficiente τ para algunos de los procedimientos de separación con mejor desempeño.

	Mejor combinación				Peor combinación			
Nombre de la familia	ϵ	τ	% Cortes	% φ	ϵ	τ	% Cortes	% φ
contiguityIneqs	0.00	6.99	100.00	100.00	4.00	44.84	0.00	0.00
symmContiguityIneqs	0.10	8.25	70.21	71.16	1.50	47.53	4.83	0.66
symmetricalBFcontiguity	0.30	17.80	8.56	26.79	1.90	34.29	3.27	5.18
notBranchFromSrc	0.80	19.59	0.96	0.82	1.30	31.60	0.67	0.34
farSlotsOff	0.30	23.15	36.31	55.53	1.40	47.36	1.58	4.09
ppalSlotsToDst	0.60	23.28	14.83	14.07	1.50	45.20	0.86	0.21
ppalSlotsFromSrc	0.80	23.56	7.78	7.34	4.00	44.77	0.00	0.00
contiguityEqs	0.30	23.86	15.96	17.82	1.90	51.06	0.54	0.10
exactlyVdFromSrc	0.00	26.00	100.00	100.00	0.30	73.94	0.28	1.86
notBranchToDst	1.40	26.87	0.54	0.17	4.00	38.59	0.00	0.00
notBranchFromV	0.30	26.90	1.09	1.02	0.40	38.96	0.91	1.27
onePpalSlotsFromSrc	0.60	27.35	8.56	12.81	0.90	46.84	5.46	6.43
exactlyVdFromV	1.60	28.52	1.28	0.85	0.60	43.42	2.00	1.53
positionalFittingLowerTight	1.10	28.56	2.41	14.42	4.00	46.75	0.00	0.00
highPositionForces	0.90	29.31	2.64	17.33	1.70	49.37	1.44	19.38
contiguityASCC	0.40	29.45	9.25	7.35	1.00	44.92	0.00	0.00
outgoingDBrooms	2.70	30.78	3.24	1.26	0.00	43.56	100.00	100.00
positionalFittingGreaterTight	0.40	31.30	5.94	7.20	2.00	47.61	0.86	0.00
incomingDBrooms	0.80	31.51	9.18	8.00	4.00	43.21	0.00	0.00
lowPositionForces	0.05	31.57	11.36	16.67	4.00	48.07	0.00	0.00
centralSsBetweenBounds	0.80	32.17	18.77	27.99	0.30	46.17	13.15	28.06
simplestCycles	1.30	32.52	0.40	0.51	4.00	39.52	0.00	0.00
positionalFittingLower	0.20	33.04	18.65	42.30	1.00	46.73	21.15	26.73
eqAmountOfAsForTheSumOfSs	0.10	33.17	93.37	91.42	1.90	42.98	18.41	17.00
posFitHigh2DBySlotsOneVsAll	0.10	33.70	8.79	9.95	0.80	48.80	0.00	0.00
centralSsBetweenHighBounds	1.60	34.15	45.16	37.92	0.00	49.11	52.38	100.00
eqAmountOfAsForEachUsedS	0.90	34.52	48.22	7.45	4.00	44.70	64.50	69.00
posFit3DByImplication	0.10	35.27	10.17	38.00	1.90	47.92	2.67	7.47
simplestCyclesImplication	0.20	35.28	43.77	49.74	0.60	44.35	57.03	68.78
oneSlotOnceFromV	0.15	35.33	17.18	20.16	0.00	46.50	100.00	100.00
positionalFittingLowerOneVsAll	0.30	35.51	10.90	20.47	1.80	45.45	5.81	14.36
centralSsBetweenLowBounds	1.00	35.52	100.00	65.73	1.90	48.54	25.45	33.43
nonOverFixNextHiBySum	0.00	36.06	100.00	100.00	1.00	45.14	0.00	0.00
positionalFittingGreaterOneVsAll	0.50	36.10	4.21	4.69	4.00	47.68	0.00	0.00
oneSlotOnceToDst	0.20	36.78	21.24	28.89	4.00	44.81	0.00	0.00
posFitLow2DBySlotsOneVsAll	0.40	37.00	4.31	10.81	0.20	44.85	12.26	39.71
posFit3DBySum	0.30	37.56	12.87	16.25	0.60	48.27	6.47	5.37
oneSlotOnceToV	0.20	38.06	41.43	26.52	2.00	44.80	0.71	0.38
nonOverBySum	0.00	38.23	100.00	100.00	0.10	47.89	52.12	32.52
nothingFromDst	0.60	38.23	38.10	11.11	0.10	43.71	57.14	55.56
positionalFittingGreater	0.50	38.27	54.70	69.29	4.00	47.79	0.00	0.00
nonOverFixNextLowBySum	0.50	38.51	3.04	3.83	0.10	48.16	30.06	67.51
oneSlotOnceFromSrc	0.75	40.09	14.36	30.99	0.40	50.08	11.33	31.63
simplestCyclesBySlot	0.90	40.80	2.20	0.00	0.70	44.00	36.26	20.97
posFitHigh2DBySlots	0.70	42.18	100.00	16.14	0.90	62.02	28.34	0.00
centralSlotsFromSourceBySlot	0.90	42.21	27.04	18.79	0.50	48.85	32.65	18.98
KDemandsDoNotExceedS	0.00	42.39	100.00	100.00	0.10	45.54	20.00	19.35
twoDemandsDoNotExceedS	1.00	42.58	0.47	30.40	4.00	44.71	0.00	0.00
twoDemandsDoNotExceedSBySum	4.00	42.94	0.00	0.00	0.30	49.46	3.31	0.21
cyclesBySlot	1.30	43.62	2.02	1.75	1.00	60.40	5.39	3.70
nonOverFixNextHi	0.00	43.73	100.00	100.00	0.20	58.02	37.99	30.95
centralSlotsFromSource	0.00	44.38	100.00	100.00	0.20	48.54	23.08	38.46
cycles	6.00	46.47	2.46	0.00	2.20	85.06	0.00	5.28
posFitLow2DBySlots	1.00	46.92	68.73	2.00	0.92	57.26	55.06	12.43
extendedCycles	1.80	47.68	40.15	52.85	0.00	59.22	100.00	100.00
nonOverFixNextLow	0.10	47.74	46.57	75.99	0.40	56.92	15.78	28.32
KDemandsDoNotExceedSBySum	0.10	48.41	0.00	0.00	0.20	48.57	0.00	0.00
extendedCyclesBySlot	0.90	48.76	34.74	28.05	6.00	55.92	0.00	0.00
posFit3DBySlots	0.00	65.77	99.64	100.00	0.50	71.02	45.57	56.72
nonOver	0.00	67.90	100.00	100.00	0.90	72.72	0.00	0.00

Tab. 4.3: Coeficiente τ total, número normalizado de cortes generados y coeficiente φ normalizado, para los valores de ϵ con mejor y peor desempeño para cada familia.

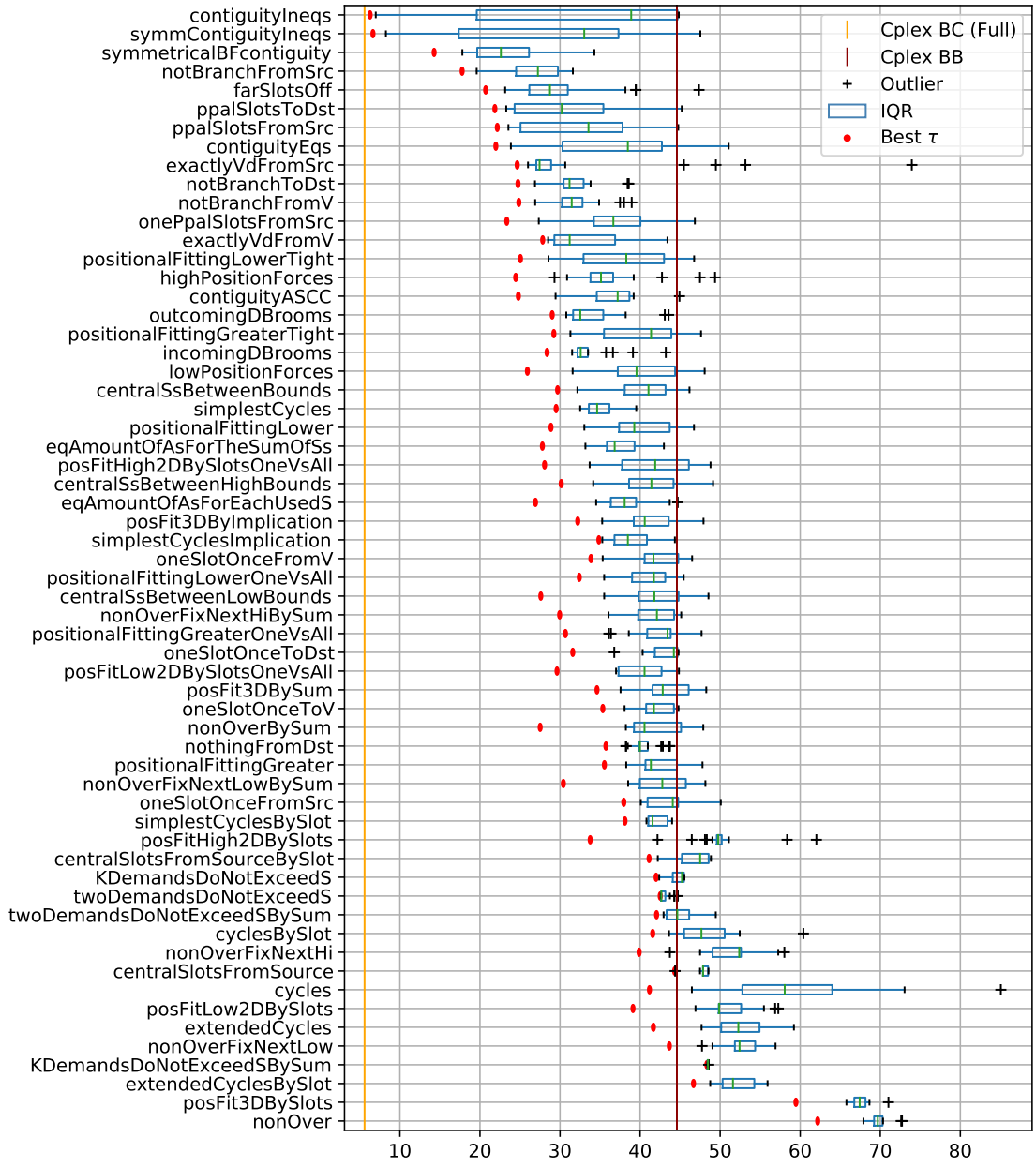
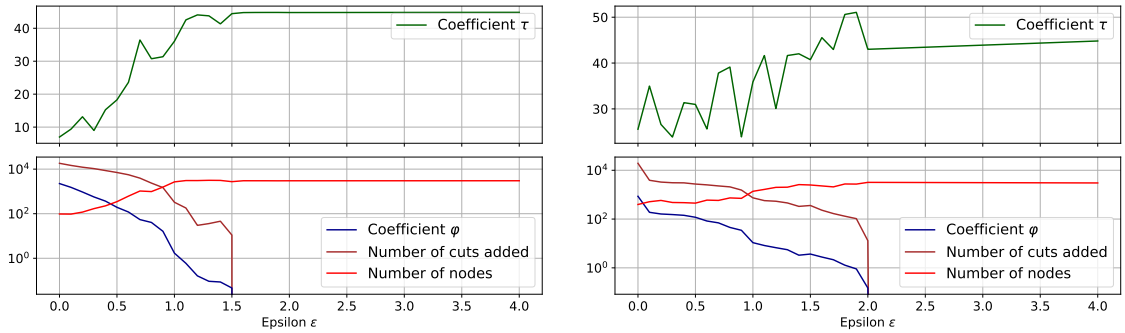


Fig. 4.5: Comparación de los gráficos de caja de la variación de τ para cada familia de cortes según el valor asignado a ϵ. Estos valores son también comparados con las implementaciones del *branch-and-bound* y el *branch-and-cut* de Cplex, habilitando el *pre-solver* y heuristicas para el segundo. La figura también muestra el mejor τ que podemos obtener para cada familia cuando se selecciona el ϵ con mejor desempeño para cada instancia.

La Tabla 4.3 muestra resultados adicionales de todas las familias, reportando para el mejor y peor valor de ϵ, dicho coeficiente, el coeficiente τ total, el número de cortes generados y el coeficiente φ, estos dos últimos normalizados. La tabla está ordenada por el mejor τ obtenido. La normalización de los cortes y el coeficiente φ para cada procedimiento



(a) ContiguityIneqs **(b)** ContiguityEqs

Fig. 4.6: Coeficiente τ, cantidad total de cortes agregados y coeficiente de efectividad φ en función del ϵ para dos de las familias con mejor desempeño.

i y ϵ' se obtiene como

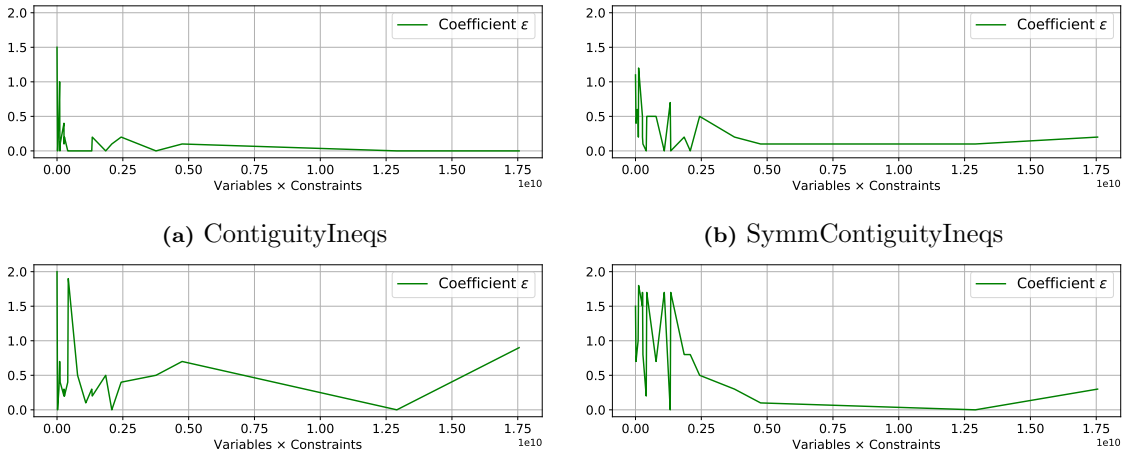
$$100\ \frac{m_{\epsilon'} - l}{h - l},$$

siendo h y l el mayor y menor valor otbenido en todas las ejecuciones relacionadas con i, y $m_{\epsilon'}$ el valor obtenido cuando $\epsilon = \epsilon'$. Esto significa, por ejemplo, que para la familia *contiguityIneqs* el mejor resultado fue obtenido cuando ambos –los cortes agregados y el coeficiente de efectividad– fueron los máximos para esta familia. Es interesante notar que estos valores normalizados son 34.06 y 33.67 para la cantidad de cortes y φ, respectivamente, en promedio para todas las familias cuando los mejores valores para ϵ son seleccionados, mientras que son 17.61 y 19.57, respectivamente, para los peores valores del coeficiente ϵ. Para las diez mejores y las diez peores combinaciones, estos resultados son mejores, dando 35.51 % y 39.37, respectivamente para las mejores, mientras que para las diez peores, están ambos cercanos al 1 %. También podemos observar que el mejor y peor ϵ son 0.71 y 1.57, respectivamente, en promedio para todas las familias.

Otro resultado notable es que la adición de casi todos los procedimientos de separación de manera aislada, con una buena calibración de sus parámetros, tiene mejor rendimiento que el algoritmo del tipo *branch-and-bound* implementado por Cplex, el cual obtiene una suma del coeficiente de rendimiento de 44.598. Este comportamiento es válido incluso cuando el parámetro ϵ no es ajustado de la mejor manera. La Figura 4.5 muestra la comparación de la lista completa de familias junto con el *branch-and-bound* genérico de Cplex y su configuración *FullCplex*, *i.e.*, el *solver* con cortes genéricos, heurísticas, *pre-solver* y paralelización. Notar que los coeficientes obtenidos por los mejores procedimientos de separación propuestos, cuando el coeficiente ϵ suficientemente bien calibrado, son comparables con la mejor configuración de Cplex, la cual obtiene una suma de los coeficientes τ igual a 5.584.

También buscamos medir de algún modo la efectividad en la selección del parámetro ϵ como método para filtrar cortes válidos. La Figura 4.7 muestra el valor para el mejor ϵ como una función del tamaño de la instancia para las cuatro familias de cortes con mejor desempeño. El tamaño de cada instancia es calculado como $|variables| \times |restricciones|$. Las formas de las curvas presentadas parecieran sugerir que, especialmente para instancias pequeñas pero también para algunas más grandes, es preferible no agregar una gran cantidad de cortes. A pesar del hecho de que este comportamiento es observado en la mayoría

de los cortes, si para alguno de ellos miramos a la suma de los τ obtenidos al seleccionar el mejor ϵ para cada instancia, podemos ver que no hay mucha diferencia en utilizar un único ϵ por familia para todas las intancias.



(a) ContiguityIneqs **(b)** SymmContiguityIneqs

(c) PpalSlotsFromSrc **(d)** NotBranchFromSrc

Fig. 4.7: Coeficiente ϵ con mejor rendimiento en función del tamaño de la instancia, calculada como $|variables| \times |constraints|$, para cuatro de las familias de cortes con mejor desempeño.

Como resultado final de estos experimentos hemos decidido no utilizar como cortes para el *branch-and-cut* las familias de desigualdades *cycles*, *extendedCycles*, *extendedCyclesBySlot*, *nonOver*, *nonOverFixNextLow*, *KDemandsDoNotExceedSBySum*, *posFit3DBySlots* y *posFitLow2-DBySlots* dado que no parecen proporcionar mejoras por sobre un algoritmo *branch-and-bound* genérico. Creemos que la razon de su desempeño pobre es principalmente el tiempo de separación, dado que justamente estos cortes pertenecen a familias muy grandes.

4.3.4. Comparación de las estrategias de selección

En esta subsección reportamos los experimentos realizados con el fin de evaluar el desempeño de las distintas estrategias de selección propuestas en la Sección 4.2. Con el objeto de calibrar el parámetro h, el cual limita la cantidad de procedimientos diferentes de los cuales obtener cortes, utilizamos el mismo subgrupo de 22 instancias de la subsección anterior, *i.e.*, el conjunto *Lowest-density*. Experimentamos con $h \in [5, 10, 15, 20, 25, 30]$. Para cada estrategia y valor de h, consideramos la suma del coeficiente de rendimiento τ sobre todas las instancias como una aproximación al rendimiento total. También contemplamos preordenar o no los procedimientos de separación basándonos en los resultados de experimentos anteriores. Los resultados obtenidos están sumarizados en la Tabla 4.4.

Una vez encontrado el mejor valor de h para cada estrategia, experimentamos con las 100 instancias con un tiempo límite de 15 minutos. Nuestros cortes son agregados al algoritmo *branch-and-bound* genérico implementado por Cplex sin *pre-solver* ni heurísticas primales. La comparación es contra esta configuración con el fin de mostrar que los cortes propuestos son de hecho efectivos, y contra el *branch-and-cut* implementado por Cplex para probar si éstos son mejores que los cortes genéricos. La Tabla 4.5, la Figura 4.8, y la

Estrategia	5	10	15	20	25	30
MOST EFFECTIVE	5.60	4.58	4.43	4.93	4.69	4.72
MOST EFFECTIVE Pre-sorted	3.78	4.46	4.99	5.10	4.59	5.51
MOST EFFECTIVE RND	5.36	4.32	4.59	5.23	4.79	6.97
MOST EFFECTIVE RND Pre-sorted	4.35	4.62	4.58	5.03	5.47	4.38
RANDOM	6.24	5.59	5.99	4.56	4.88	4.86
WEIGHTED	5.54	5.42	4.63	5.26	5.51	5.12
WEIGHTED Pre-sorted	5.66	6.06	5.49	5.21	5.19	5.23

Tab. 4.4: Rendimiento de las estrategias de separación para distintos valores del parámetro h sobre el conjunto de 22 instancias.

Figura 4.9 reportan estos resultados, mostrando que todas estas estrategias se comportan mejor que los dos procedimientos implementados por Cplex. Para las instancias resueltas hemos mejorado todos los tiempos de resolución con cada estrategia, mientras que la mejor estrategia propuesta fue capaz de resolver hasta el óptimo casi un tercio más de instancias que la implementación del *branch-and-cut* de Cplex y casi el doble que la del *branch-and-bound*.

	τ	tiempo [hs]	Opt.	Fact.	Desc.	Mem.
Most Eff. with Rnd. 10 sorted	1121.56	9.08	73	0	27	4
Most Effective 5 sorted	1237.84	9.96	68	1	31	4
Most Eff. with Rnd. 5 sorted	1273.96	10.38	68	0	32	4
Most Effective 10 sorted	1316.44	10.27	64	1	35	4
Random 20	1350.32	11.51	64	0	36	3
Weighted 15	1385.75	11.06	63	0	37	4
Weighted 25	1393.53	11.14	63	0	37	4
Brute Force	1577.47	12.30	55	0	45	4
CplexBC	1858.67	15.02	40	10	50	4
CplexBB	2264.61	18.24	29	5	66	4

Tab. 4.5: Desempeño de las mejores estrategias de separación sobre el conjunto de 100 instancias con un tiempo límite de 15 minutos. Las dos primeras columnas corresponden la suma de los coeficientes τ y al tiempo de ejecución total, respectivamente, mientras que las columnas restantes indican el número de instancias resueltas por optimalidad, con solución entera pero sin óptimo probado, sin solución entera hallada y con problemas de memoria, respectivamente.

También es interesante que, excepto por una instancia, las estrategias de Cplex fueron las únicas con las cuales el procedimiento completo retornó soluciones factibles subóptimas. De aquellas 14 instancias, la peor estrategia resolvió siete instancias hasta el óptimo, mientras que las mejores dos resolvieron 13 instancias.

Hay solamente 27 instancias resueltas hasta el óptimo dentro del tiempo límite por todos los algoritmos, y 39 resueltas por todos excepto por el *branch-and-bound* implementado por Cplex. Para estas instancias, las mejores estrategias demostraron la optimalidad en la mitad del tiempo requerido por la implementación de Cplex del *branch-and-cut*. La Figura 4.10 sumariza estos resultados. Es interesante notar que la estrategia con segundo mejor rendimiento según el coeficiente τ, *i.e.*, *Most Effective with 5 families pre-sorted*,

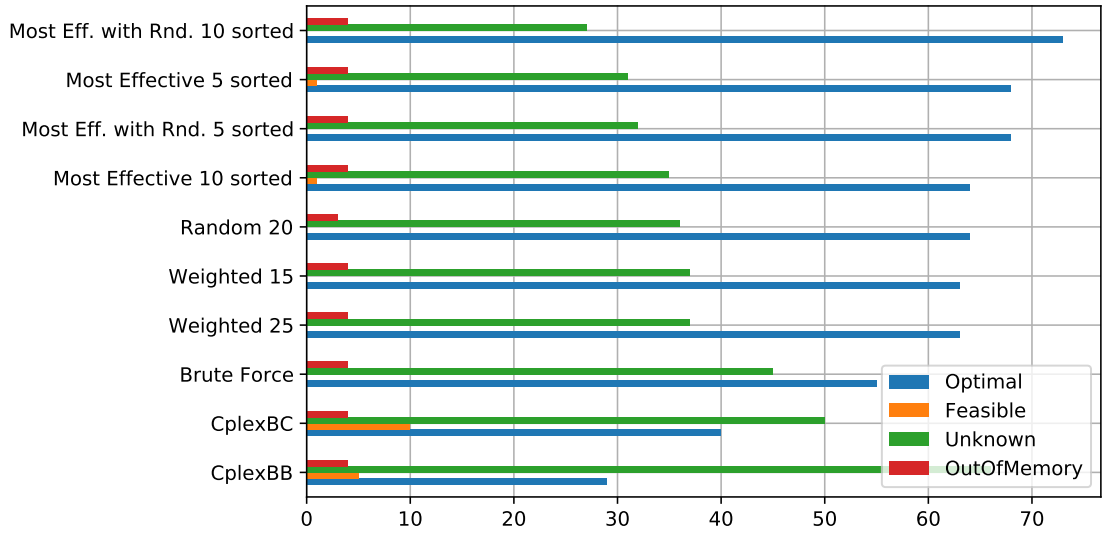


Fig. 4.8: Cantidad de instancias resultantes en cada categoría para las estrategias con mejor desempeño y los algoritmos *branch-and-cut* y *branch-and-bound* genéricos implementados por Cplex, sobre el conjunto de 100 instancias, y con un tiempo límite de 15 minutos.

es una de las peores para definir optimalidad. Sospechamos que el relativamente pequeño número de cortes que esta estrategia agrega (la cual utiliza sólo cinco familias) le permite resolver instancias más grandes sin complicar demasiado el modelo, con la desventaja de perder efectividad al querer resolver las instancias más simples. Este argumento gana fuerza dado que este análisis está particularmente enfocado en instancias resueltas hasta la optimalidad por todas las estrategias, en particular por el *branch-and-cut* implementado por Cplex.

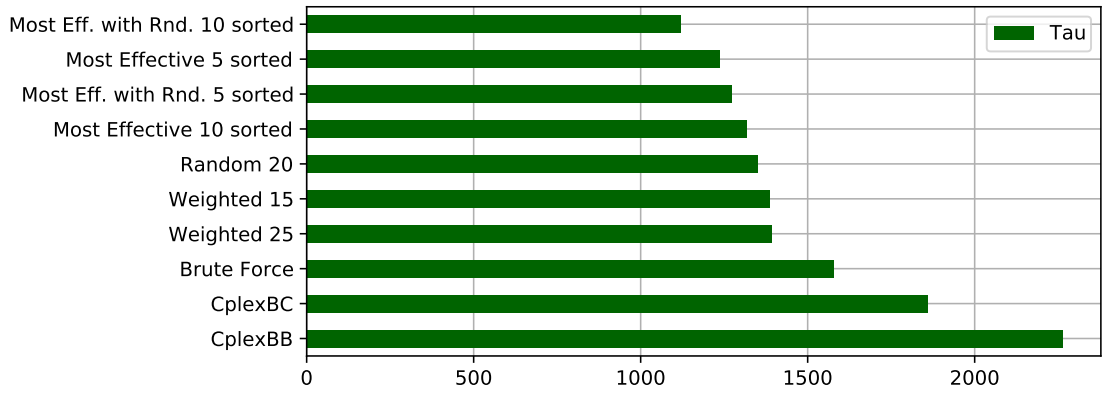


Fig. 4.9: Comparación del mejor valor obtenido para el coeficiente τ por las estrategias con mejor desempeño contra las implementaciones genéricas del *branch-and-cut* y *branch-and-bound* de Cplex sobre el conjunto de 100 instancias con un tiempo límite de 15 minutos.

Respecto de los problemas de memoria reportados, algunas particularidades requieren

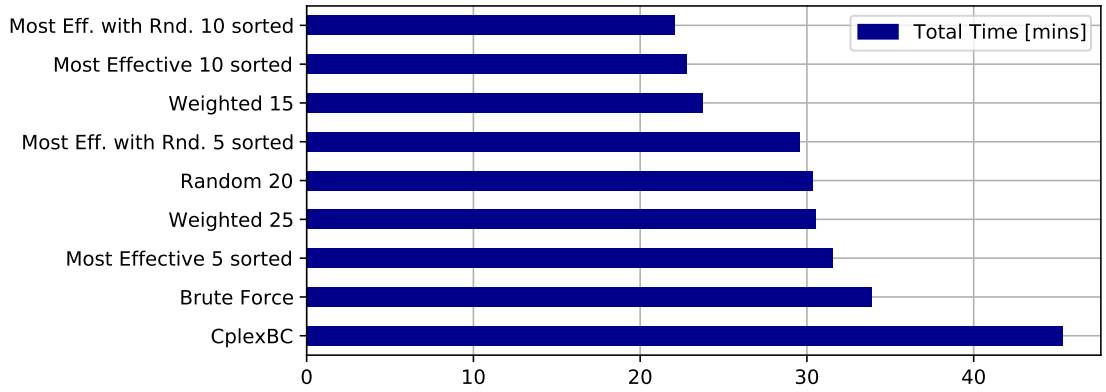


Fig. 4.10: Comparación de los tiempos requeridos para demostrar optimalidad para las 39 instancias que todas las estrategias, excepto el *branch-and-bound*, pudieron resolver dentro del tiempo límite de 15 minutos.

una explicación. Es interesante notar que para tres de las cuatro instancias con problemas de memoria la implementación no logró armar el modelo, por lo tanto es necesaria una heurística o un *pre-solver* para manejarlas con un *branch-and-bound* o un *branch-and-cut*. Asimismo, la cuarta instancia, *i.e.*, `43n-176m-EuroLarge_150_20_47`, no pudo ser resuelta únicamente por la estrategia *Random* en una sola de sus ejecuciones, lo cual puede ser un caso aislado, por lo tanto dicha estrategia es probablemente peor que las dos *Weighted*. Otro detalle que las figuras no muestran es que la implementación genérica del *branch-and-bound* de Cplex tuvo problemas de memoria en una de las ejecuciones de la instancia `6n-9m-n6s9_200_8_108`.

4.4. Conclusiones

En este capítulo hemos propuesto un algoritmo *branch-and-cut* para el RSA. En primer lugar, presentamos los procedimientos implementados en orden a separar las desigualdades e igualdades válidas y los cortes optimales pertenecientes a las diferentes familias propuestas en el capítulo anterior. Para cada algoritmo hemos calculado su complejidad temporal y propuesto para la mayoría de ellos algunas posibles modificaciones, generalmente mediante el uso de estructuras particulares, para mejorar el rendimiento al menos en el caso promedio o al limitarse a las instancias utilizadas en el presente trabajo. No implementamos algunas de estas mejoras debido al bajo rendimiento del *branch-and-cut* al agregar los cortes asociados, sin embargo dejamos como trabajo futuro implementar las otras modificaciones. También hemos presentado varias estrategias de selección, es decir, las rutinas encargadas de decidir en cuál de estas familias aplicar los procedimientos de separación dentro del algoritmo *branch-and-cut*. Finalmente, hemos presentado el *benchmark* de instancias que utilizamos en este y el siguiente capítulo, así como los diversos experimentos realizados con el fin de calibrar los diferentes parámetros tanto de los procedimientos de separación como de las estrategias de selección. Estos resultados sugieren que los cortes son de hecho efectivos, obteniendo generalmente mejores resultados cuando el algoritmo utiliza una cantidad considerable de ellos. Asimismo, los resultados parecen mostrar que la gran mayoría de las familias presentadas son, cada una por separado, suficientemente efectivas, no sólo para permitir que el *branch-and-cut* propuesto resuelva en

menos tiempo y con mejores resultados el total de las instancias probadas en comparación con un *branch-and-bound* genérico, sino también para competir en estas métricas contra un *branch-and-cut* con cortes genéricos. Los resultados de la comparación entre las diferentes estrategias, que utilizan las familias de mejor desempeño, sugieren que, al calibrar adecuadamente sus parámetros, el *branch-and-cut* propuesto tiene un mejor desempeño que el *branch-and-cut* genérico implementado por Cplex, sin *pre-solver*, paralelización o heurísticas, sobre los conjuntos de instancias testeadas.

5

Un procedimiento heurístico para DSL-BF

Dado que al estudiar el comportamiento de las diferentes variantes del *branch-and-cut* propuesto en el Capítulo 4 se vio que la primera cota dual obtenida al resolver el RSA con el modelo DSL-BF solía ser muy cercana o el mismo óptimo, con el fin de ayudar a podar el árbol de búsqueda, es natural implementar heurísticas primales iniciales que encuentren rápidamente soluciones factibles de la mejor calidad posible. En este capítulo se propone una heurística y algunas variaciones explicando los detalles de su implementación junto con la serie de experimentos realizados, sus resultados y conclusiones obtenidas.

5.1. Introducción

Dada la complejidad del RSA, innumerables heurísticas han sido propuestas para este problema. Muchas de ellas utilizan *solvers* comerciales de programación entera como caja negra para resolver formulaciones simplificadas [70, 72, 79, 108], generalmente considerando un conjunto reducido de caminos y canales [43, 58, 147]; otras propuestas adaptan metaheurísticas, y un grupo bastante grande desarrolla sus propias heurísticas dedicadas [23, 32, 36, 38, 40].

Dentro de este universo es muy común dividir el RSA en dos fases o subproblemas, resolviendo primero el *routing* (R) y luego el *spectrum allocation* (SA) [1, 3, 7, 9, 10, 12, 22, 24, 29, 30, 31]. Dado que este abordaje intenta reducir el espacio de búsqueda, la primera fase es generalmente resuelta calculando un conjunto de k-caminos mínimos para cada demanda, usualmente utilizando los algoritmos de Yen [168] o de Johnson [64] o modificaciones al algoritmo de Dijkstra [41, 100, 155]. Una vez que el camino para cada demanda está determinado, el problema que resta resolver es el SA, el cual pertenece a la clase $\mathcal{NP}$-difícil incluso para caminos, si su longitud es de al menos tres arcos, y para cualquier tipo de anillos [132, 138]. Sin embargo, cada uno de estos subproblemas o fases es probablemente más fácil de resolver en la práctica que el RSA completo.

Hay varias publicaciones que buscan resolver el SA por optimalidad recurriendo a formulaciones de ILP pero usualmente atacan los problemas de ruteo precomputando caminos [144, 145, 159]. Sin embargo, la mayoría utiliza heurísticas incluso en esta segunda

fase.

Por ejemplo, en 2011, Wang et al. [158] propusieron resolver el RSA por medio de dos algoritmos heurísticos, a saber, el algoritmo *balanced load spectrum allocation* (BLSA) y el *shortest path with maximum spectrum reuse* (SPSR). Estas heurísticas, con leves diferencias, precomputan los k-caminos mínimos para cada par de nodos utilizando el algoritmo de Yen. Luego, para cada demanda, seleccionan el camino que minimiza el espectro utilizado en los arcos, comenzando por la demanda con mayor volumen, finalmente asignan el espectro utilizando un algoritmo goloso, *i.e.*, *first-fit* (FF ó FF-SA).

En el transcurso del mismo año, Christodoulopoulos et al. [32] estudiaron el *routing, modulation level, and spectrum allocation problem* (RMLSA) en un grafo dirigido presentando un algoritmo heurístico basado en ILP con el fin de minimizar el espectro. Dicho algoritmo precomputa k (tomando $k = 3$) caminos para cada demanda implementando un algoritmo de k-caminos mínimos, el cual a cada paso selecciona un camino mínimo y el costo de sus arcos es doblado en el grafo a fin de que los caminos hallados en futuras iteraciones no los utilicen. En febrero del mismo año, Klinkowski y Careglio [70] habían presentado otro procedimiento heurístico basado en ILP para el RSA utilizando también caminos precomputados.

En abril de 2013, Shirazipourazad et al. [132] propusieron tres heurísticas que precomputan los caminos y luego asignan el espectro para resolver el *spectrum constrained RSA problem* (SCRSA). En la fase de asignación de espectro de dos de ellas, las rutas calculadas son particionadas en conjuntos disjuntos de caminos, comenzando por el camino con la demanda más grande. Con el fin de precomputar los caminos mínimos, los autores proponen un proceso iterativo. En cada iteración el algoritmo recorre la lista de demandas no asignadas –ordenada de mayor a menor volumen– y busca el camino mínimo que satisface cada una de ellas, pero no permitiendo utilizar en dichos caminos los arcos ya utilizados por otra demanda en la misma iteración. Luego, cuando no puede satisfacer más demandas, asigna los slots y repite el proceso sobre las demandas que restan por asignar permintiéndoles nuevamente utilizar todos los arcos del grafo. Este proceso se repite hasta que no haya demandas sin asignar.

En 2014 [3] y 2015 [71, 73], Walkowiak et al. utilizan caminos mínimos y el menor *slot* disponible, *i.e.*, *first-fit*, para mejorar las cotas superiores del *branch-and-price* en cada nodo. El orden en el cual iteran sobre las demandas depende de una heurística llamada VSA, presentada en 2014 por Albin y Walkowiak [3]. La propuesta es intercambiar las demandas por pares calculando con ello un valor ω; este proceso se repite una cierta cantidad de veces, quedándose con el orden que provee el mejor ω.

Un inconveniente con todos estos trabajos es que, no sólo las instancias que manejan son de tamaños reducidos, sino que las soluciones obtenidas no tienen garantía de optimalidad debido a que son resultados o bien de heurísticas o de reducciones del espacio de búsqueda en modelos enteros (resultando también por esto mismo heurísticas en su naturaleza).

En este capítulo implementamos una heurística matemática –también conocida como *matheuristic*– que iterativamente resuelve el modelo ILP propuesto en capítulos anteriores, *i.e.*, el DSL-BF, pero con un gran número de variables prefijadas, con el objeto de obtener rápidamente una solución factible. No hemos hallado más que un trabajo, la tesis de Dao Tahn [36], que utiliza una heurística como *warm-start* para un algoritmo que hace uso de una formulación matemática. Sin embargo, el modelo ILP utilizado está basado en la configuración enlace-camino, y para resolverlo, en la tesis se precomputó un subconjunto de las rutas posibles para cada demanda.

Siguiendo el enfoque recientemente mencionado de dividir el problema en dos fases, creemos que un método prometedor es fijar para una gran cantidad de demandas –no necesariamente todas–, los caminos utilizados con el objeto de proveer un problema más simplificado al *solver*. Este problema será principalmente el SA con algunas rutas prefijadas. En la mayoría de las soluciones presentadas en el capítulo anterior, casi todas las demandas eran satisfechas utilizando un camino mínimo desde el origen hasta el destino. Entonces, el camino candidato a prefijar para cada demanda será uno de sus caminos más cortos. Siempre que no sea posible fijar uno o más de ellos, relegamos al *solver* de programación entera la difícil tarea de encontrarlos junto con la de resolver el SA. Esto nos daría una solución factible para usar como *warm-start* para el *branch-and-cut* implementado en capítulos anteriores.

El presente capítulo es organizado como sigue. La Sección 5.2 describe la heurística propuesta con detalles de implementación incluyendo un optimizador de soluciones y un verificador de (no)factibilidad. La Sección 5.3 presenta los resultados computacionales divididos como sigue: la Subsección 5.3.1 presenta el coeficiente utilizado para comparar los algoritmos y las diversas configuraciones de los parametros de la heurística (tipo de esclavo, cota inferior, optimizador y tiempo límite para el esclavo) cuyos experimentos son presentados en las Subsecciones 5.3.2 y 5.3.3; la Subsección 5.3.4, presenta una comparación de la heurística propuesta contra una configuración de Cplex, mientras que la Subsección 5.3.6 muestra una comparación del algoritmo completo propuesto contra la mejor configuración de Cplex seguido de una breve discusión.

5.2. Heurística primal

En esta sección presentamos la heurística *prefix shortest paths* (PSP), sus detalles, y los algoritmos desarrollados para mejorarla. El pseudocódigo de su estructura principal puede observarse en el Algoritmo 13. Como primer paso, la heurística precomputa un camino mínimo para cada par de nodos del grafo. Para calcular dichas rutas en un modo eficiente sobre las topologías existentes, se implementaron dos algoritmos, el Algoritmo de Dijkstra y el Algoritmo de Floyd-Warshall, comparando sus desempeños y resultando levemente mejor el Algoritmo de Dijkstra.

A continuación, se calcula una cota inferior trivial al valor de la solución óptima. Se obtiene esta cota resolviendo el problema asumiendo que cada demanda utiliza el camino mínimo –para lo cual también se asume una cantidad suficiente de *slots* en cada arco–. Denominamos a este valor como *cota inferior* (LB) y el procedimiento para calcularlo puede verse en el Algoritmo 14. Luego se inicializa un conjunto de variables necesarias para el ciclo principal. El entero *maxS* guarda en cada iteración el máximo número de *slots* que pueden ser fijados por arco, mientras que la variable de punto flotante *overlap*, íntimamente relacionada con *maxS*, indica el porcentaje del arco que puede ser utilizado, *i.e.*,

$$overlap = \frac{maxS}{\bar{s}}.$$

La idea es gradualmente relajar dicho porcentaje e ir ampliando el margen con el fin de evitar la no factibilidad. Finalmente, la variable *start* almacena el *timestamp* del inicio del algoritmo con el objeto de medir el tiempo transcurrido.

El núcleo de la heurística es el ciclo, el cual es repetido hasta que se supera un tiempo límite o hasta que el porcentaje *overlap* sea 0 (en cuyo caso *maxS* será también 0). En este

Algorithm 13 Procedimiento que prefija manualmente la mayor cantidad de rutas posibles para las demandas, relegando a un *solver* de ILP la tarea de hallar los caminos para el resto y resolver el SA.

```
1: procedure Prefix Shortest Paths (PSP)
2:     computeShortestPaths()
3:     LB ← computeLowerBound()
4:     maxS ← s̄,   overlap ← 1.0,   start ← now()
5:     while now() − start ≤ tlim_heur y  overlap > 0 do
6:         maxS ← s̄ × overlap
7:         D̄ ← shuffleDemands()
8:         fixShortestPaths(D̄, maxS)
9:         solution ← solver.run()
10:        if solution.isFeasible() then
11:            if solution.getBestObjValue() = LB then
12:                return solution
13:            end if
14:            return optimize(solution)
15:        end if
16:        if solution.isInfeasible() then
17:            overlap ← overlap − 0.10
18:            if overlap < IB then
19:                return Probably Infeasible
20:            end if
21:        else
22:            tlim_solver ← tlim_solver × 1.2
23:            overlap ← overlap − 0.05
24:        end if
25:    end while
26:    return Unknown
27: end procedure
```

ciclo, se realiza el intento de fijar los caminos a ser utilizados por algunas de las demandas y luego se ejecuta un algoritmo *branch-and-cut* buscando una asignación de *slots* tal que cumpla con las restricciones del RSA. Denominamos como *esclavo* al *solver* utilizado para resolver este subproblema. El algoritmo reordena aleatoriamente la lista de demandas en cada iteración con el objeto de cubrir mejor el espacio de búsqueda.

Como fue estudiado en capítulos previos, el modelo DSL-BF permite que las soluciones factibles tengan ciclos espurios y bifurcaciones, por lo tanto la primera solución recuperada por un *solver* IP, la cual puede ser hallada por su *pre-solver* o por sus heurísticas primales, puede tener este tipo de características no deseadas. Por este motivo, a pesar de ser una solución factible, puede estar muy lejos de ser óptima. Para estos casos, dado que el problema presenta suficiente complejidad y estamos dentro del contexto de una heurística, hemos desarrollado una subheurística con el fin de eliminar estos elementos del mejor modo posible. La misma es denominada *optimizer* y su pseudocódigo puede observarse en los Algoritmos 15 y 16. Presentamos este procedimiento en detalle en las siguientes secciones.

Una novedad de la heurística es que tiene un mecanismo para detectar temprano las

Algorithm 14 Dado un diccionario que contiene un camino mínimo para cada par de nodos, el procedimiento retorna el resultado de la función objetivo asumiendo $\bar{s}$ suficientemente grande como para asignar a cada demanda el camino asociado a su par origen-destino.

```
1: procedure computeLowerBound({(Vertex, Vertex): {Arc}} shortestPaths)
2:     sum ← 0
3:     for d ← D do
4:         sp ← shortestPaths.get(s(d), t(d))
5:         sum ← sum + sp.size()
6:     end for
7:     return sum
8: end procedure
```

Algorithm 15 Dada una solución y un diccionario que contiene un camino mínimo para cada par de nodos, el algoritmo intenta optimizar el camino para cada demanda removiendo *slots*, ciclos y caminos espurios en la solución.

```
1: procedure optimize(Solution sol, {(Vertex, Vertex): {Arc}} shortestPaths)
2:     u ← new double[D.size()][m][s̄]
3:     for d ← D do
4:         u′ ← sol.getSolutionValues()
5:         if Σ_{e∈E} Σ_{s∈S} u′_des ≥ v(d) × shortestPaths.get(s(d), t(d)).length() then
6:             optimizeDemandPath(d, sol, u)
7:         end if
8:     end for
9:     sol.setSolutionValues(u)
10: end procedure
```

instancias que proponemos llamar *probablemente no factibles*, es decir, instancias tales que si continuamos el proceso de resolución probablemente resulten no factibles. La forma de detectar estas instancias es modificando el algoritmo inicial contando el número de veces que los subproblemas de una instancia resultan no factibles. Como el límite de tiempo del *solver* se estima para que no tenga que iterar más de media docena de veces, el valor de *overlap* depende casi exclusivamente del número de *no factibles* del esclavo. Entonces, cuando este valor sea menor o igual a un cierto porcentaje, llamado *cota de no factibilidad* (IB) y estimado en 75 % en la práctica (es decir, $overlap = 0.75$), la heurística retorna *probablemente no factible*. Aunque las pruebas empíricas no alcanzan para asegurar una conclusión definitiva, sugieren que el algoritmo implementado es suficientemente robusto. En todos los experimentos llevados a cabo, se detectó aproximadamente el 90 % de las instancias no factibles, no arrojando más que un falso positivo en una configuración particular no demasiado tolerante.

En resumen, lo que distingue a esta heurística de las existentes es que se enfoca principalmente en la factibilidad relegando la optimalidad a una segunda fase, que esta heurística usa un *solver* como esclavo para resolver un subproblema en forma iterativa, y que es capaz de de detectar instancias no factibles con un alto porcentaje de aciertos sin perder demasiado tiempo en ello.

Algorithm 16 Dada una demanda d comienza desde el origen $s(d)$ y, utilizando un procedimiento BFS modificado, busca el camino mínimo hasta $t(d)$ que tiene al menos $v(d)$ *slots* contiguos.

```
1: procedure optimizeDemandPath(Demand d, Solution sol, double[][][] u)
2:     parent ← {}
3:     visited ← {s(d) : [1, s̄]}
4:     leaves ← [(s(d), 1, s̄)]
5:     while not leaves.empty() do
6:         v ← leaves.first()
7:         for adj ← nodesReachableFrom(v, sol) do
8:             rFromVtoAdj ←intersection(adj.range, v.range)
9:             if not visitedWithSuperRange(visited, adj.id, rFromVtoAdj) then
10:                visited[adj.id].append(rFromVtoAdj)
11:                if rFromVtoAdj.slots ≥v(d) then
12:                    node ← new Node(adj.id, rFromVtoAdj.left, rFromVtoAdj.rigth)
13:                    leaves.add(node)
14:                    parent.put(node, v)
15:                    if node.id =t(d) then
16:                        u ← narrowestPath(rFromVtoAdj, parent, d, node)
17:                        return
18:                    end if
19:                end if
20:            end if
21:        end for
22:    end while
23: end procedure
```

5.2.1. Detalles de la implementación

Después de calcular un camino mínimo para cada par de nodos, la heurística genera dos estructuras: *arcToPaths* para cada arco relacionándolo con el conjunto de todos los caminos mínimos que lo utilizan, y *pathToArcs* que para cada camino minimo (asociado e identificado con un par origen-destino), almacena el conjunto de arcos compartidos con otros caminos mínimos. El procedimiento para obtener dichos diccionarios puede observarse en el Algoritmo 17.

El modo de limitar la cantidad de demandas a fijar ha ido variando a lo largo del trabajo. Primero intentamos fijar el valor de solapamiento y mantenerlo igual para todas las iteraciones. Así fue que definimos el parámetro *overlap*. Sin embargo, en las pruebas con distintos valores de $overlap \in [0.5, 1.0]$, los mejores resultados fueron obtenidos alrededor de 0.75, *i.e.*, ocupando a lo sumo un 75 % de los *slots* de cada arco. Luego decidimos variar este parámetro de forma automática, lo cual resultó en una mejora notable en los tiempos. El algoritmo resultante es el siguiente. Comienza con un valor de solapamiento del 100 % ($overlap = 1.0$), en la primera iteración sobre la lista de demandas, *i.e.*, buscando fijar todos los caminos mínimos incluso si saturan todos los arcos. Una vez que no es posible agregar más demandas, un algoritmo *branch-and-cut* es ejecutado resolviendo el resto del problema. Si el problema generado resulta no factible, las demandas son reordenadas aleatoriamente y el proceso comienza nuevamente reduciendo el valor de *overlap* por un

Algorithm 17 Dado un diccionario que para cada par de nodos tiene asociado un camino mínimo, el procedimiento retorna dos diccionarios: *arcToShortestPaths*, el cual relaciona cada arco con el conjunto de caminos mínimos que coinciden en dicho arco; y *pathToArcs*, que para camino minimo recibido, almacena el conjunto de arcos compartidos con al menos uno de los demás caminos mínimos

```
1: procedure GENERATESTRUCTURES({(Vertex, Vertex): {Arc}} shortestPaths, {Arc:
   {(Vertex, Vertex)}} arcToPaths, {(Vertex, Vertex): {Arc}} pathToArcs)
2:     arcToPaths ← {}
3:     pathToArcs ← {}
4:     for path ← shortestPaths.keySet() do
5:         for arc ← shortestPaths.get(path) do
6:             if arc ∈ arcToPaths.KeySet() then
7:                 pathsOfArc ← arcToPaths.get(arc)
8:                 setOfPaths ← {};
9:                 if pathsOfArc.length = 1 then
10:                    setOfPaths.add(pathsOfArc.getOne())
11:                end if
12:                setOfPaths.add(path)
13:                for pair ← setOfPaths do
14:                    if pair ∈ pathToArcs.KeySet() then
15:                        arcsOfPath ← pathToArcs.get(pair)
16:                    else
17:                        arcsOfPath ← {}
18:                    end if
19:                end for
20:                arcsOfPath.add(arc)
21:                pathToArcs.put(p, arcsOfPath)
22:            else
23:                pathsOfArc ← {}
24:            end if
25:            pathsOfArc.add(path)
26:            arcToPaths.put(arc, pathsOfArc)
27:        end for
28:    end for
29: end procedure
```

porcentaje. Para los experimentos utilizamos el 10 %, *i.e.*, el nuevo valor de *overlap* se define como

$$overlap \leftarrow overlap - 0.10.$$

Dado que cada camino fijado bloquea el uso de una cierta cantidad de slots en los arcos que utiliza, si al iterar la lista de demandas hay un camino mínimo que no se puede establecer por falta de espacio en alguno de los arcos que necesita, el procedimiento saltea la demanda y procede con la siguiente, relegando al *solver* la tarea de encontrar la ruta a ser utilizada por esa demanda (además de resolver el SA).

El algoritmo *branch-and-cut* tiene a su vez un límite de tiempo denominado $tlim_{solver}$, que fue estimado analizando resultados previos. Para fijar este valor, suponemos que el

tiempo de resolución de una instancia es lineal con respecto al número de variables y restricciones presentados por el modelo generado, *i.e.*, $\mathcal{O}(variables + restricciones)$. Por lo tanto decidimos realizar un pequeño conjunto de experimentos con el fin de hallar una función que retornara un límite suficientemente holgado como para resolver una cantidad razonable de instancias factibles, pero al mismo tiempo suficientemente ajustado para no destinar demasiado tiempo a instancias no factibles. Finalmente hemos arribado a la siguiente definición:

$$tlim_{solver} = \left\lceil \frac{variables + restricciones}{k \times \gamma} \right\rceil,$$

donde $k = 3000$ es una constante, y γ es un factor que permite una calibración un poco más precisa. Realizamos algunos experimentos con γ en el intervalo $[0.1, \ldots, 2]$ y seleccionamos $\gamma = 0.5$.

Si el *solver* no puede hallar ninguna solución ni determinar no factibilidad dentro del este tiempo límite el valor de *overlap* es reducido en un pequeño porcentaje, que en los experimentos se fijó en el 5 %. Asimismo, en estos casos el tiempo límite del solver es también extendido un 20 %, *i.e.*, $tlim_{solver} = 1.2 \times tlim_{solver}$.

La forma de fijar los caminos es agregando restricciones al modelo inicial. En este paso también hubo una modificación que resultó en una mejora notable. En una versión inicial de la heurística, para cada demanda d se fijaba el camino correspondiente P_d forzando a la demanda d a usar $v(d)$ *slots* en cada arco $e \in P_d$, *i.e.*, agregando la igualdad

$$\sum_{s \in S} u_{des} = v(d) \qquad \forall d \in D', \forall e \in P_d,$$

llamando $D' \subseteq D$ al conjunto de demandas a las cuales la heurística fue capaz de fijar el camino. Pero hemos descubierto que si también forzábamos a las variables relacionadas con la demanda y los arcos no utilizados por ésta a ser 0, es decir, agregando la igualdad

$$\sum_{s \in S} u_{des} = 0 \qquad \forall d \in D', \forall e \notin P_d,$$

la resolución computacional del modelo resultante mejora enormemente. Con esta mejora, la heurística pudo obtener soluciones en segundos para algunas de las instancias que previamente no se podían resolver luego de varios minutos.

Optimizador de soluciones

Como se mencionó, el modelo DSL-BF permite que las soluciones factibles tengan ciclos espurios, bifurcaciones y múltiples caminos espurios. Asimismo, dado que le pedimos al *solver* que devuelva la primera solución factible, es posible que ésta haya sido encontrada por el *pre-solver* o por las heurísticas, por lo que podría tener estas características no deseadas. De hecho, en una primera implementación, las soluciones obtenidas contenían una gran variedad de ciclos y caminos, además de la ruta utilizada para satisfacer cada demanda. Algunos de ellos se pueden eliminar fácilmente, por ejemplo los ciclos que no cruzan el camino principal, pero algunos pueden ser mucho más complicados de remover, como el ejemplo real de la Figura 5.1 o los casos ficticios de la Figura 5.2. El caso presentado en la Figura 5.2a complica la búsqueda por medio de un procedimiento BFS, dado que en este algoritmo, comenzando desde $s(d)$, el nodo v_2 será alcanzado a través del arco

$s(d) \to v_2$ en lugar de utilizando el camino principal. Un caso más complicado es presentado en la Figura 5.2b. Aquí un BFS estándar llegaría hasta el nodo v_3 a través del ciclo espurio antes de darse cuenta de que es un ciclo.

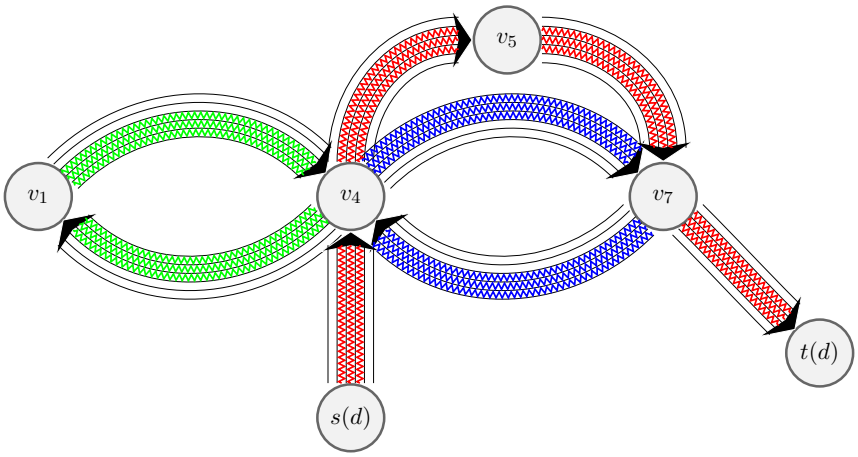
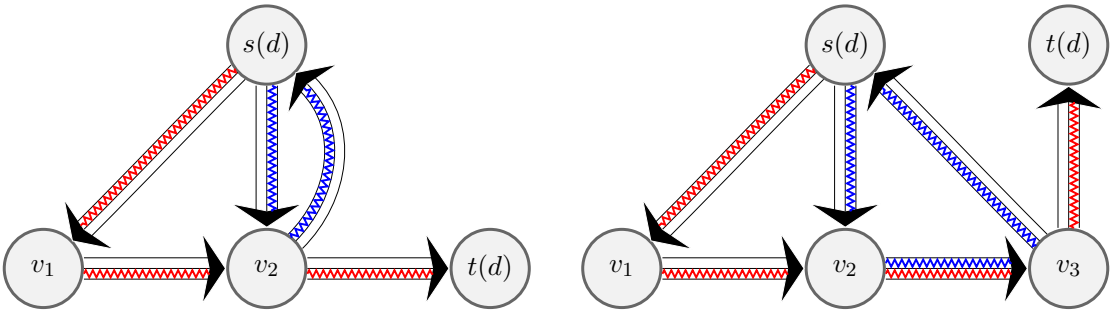


Fig. 5.1: Solución subóptima para una demanda d con dos ciclos espurios (discriminados por color) que interseca con el camino óptimo. Los *slots* fueron agrupados para mayor claridad. Estos grupos están compuestos por 18, 7, 32, 18 y 7 *slots*, respectivamente, y el volumen de la demanda $v(d)$ es 57.



(a) El ciclo espurio interseca el camino óptimo en dos nodos.

(b) El ciclo espurio interseca el camino óptimo en dos arcos.

Fig. 5.2: Escenarios posibles de soluciones subóptimas para una demanda d con $v(d) = 1$. La solución tiene un ciclo espurio en azul. Un algoritmo BFS simple no es capaz de detectar estos dos casos, entre otros.

Estas consideraciones dieron pie al desarrollo del procedimiento presentado en esta sección, que denominamos *optimizador* y cuyo pseudocódigo se puede observar en los Algoritmos 15 y 16. Para cada demanda d, este procedimiento comienza desde el origen $s(d)$ e intenta rearmar en paralelo todos los caminos que alcanzan $t(d)$ utilizando al menos $v(d)$ *slots* contiguos. A modo de poda, se detiene cuando halla el mínimo. Por precondición, dicho camino existe, a pesar de que puede no ser único. Este camino puede ser hallado heurísticamente por medio de un DFS o de modo exacto utilizando BFS. La ventaja de usar un BFS es que cuando alcanzamos el destino el camino es mínmo, mientras que la ventaja de utilizar un DFS es la velocidad en la cual se llega a la solución. Nosotros decidimos implementar un BFS. Sin embargo, como ya mencionamos, este algoritmo no

funciona con su implementación estándar, sino que cambiamos la forma en la cual los nodos son marcados como visitados. Podría ser que un camino más largo, pero que alcance el destino, requiera utilizar un nodo ya visitado por otro camino más corto pero no óptimo. Dicho caso es representado en la Figura 5.2. Hemos decidido para cada nodo mantener un seguimiento del rango de *slots* para el cual ya fue visitado, dado que no tiene sentido volver a intentar utilizar un nodo con un subintervalo de los *slots* ya usados, sino con al menos un *slot* diferente. Además, dado que es un BFS, cada nodo es alcanzado primero con el camino mínimo tal que utiliza dicho rango de *slots*. Para este fin se implementó una estructura denominada *Node*, la cual asocia cada *id* del vértice en el grafo G con el rango con el cual fue alcanzado en el árbol de búsqueda y es actualizada cada vez que un nodo es visitado. Para representar un *slot*, se utiliza un alias del tipo *int* denominado *Slot*, y para representar un rango de *slots*, se implementó la estructura *Range*. Las dos estructuras entonces son las siguientes,

```
Node {
        int id;
        Range range;
}
```

```
Range {
        Slot left;
        Slot right;
}
```

El algoritmo comienza con un árbol vacío e inicializa la lista de hojas con un único *Node*, éste está compuesto por el origen de la demanda y por el rango completo de *slots*, *i.e.*, como si dicho nodo hubiera sido visitado utilizando los $\bar{s}$ *slots*. El ciclo principal del algoritmo, que es el BFS modificado, es ejecutado hasta que no hay más hojas en la lista. Por lo tanto, en cada iteración toma una de las hojas de dicha lista utilizando *first-in-first-out* (FIFO) como en BFS. Para obtener los vecinos del nodo tomado de la lista, se cuenta con un diccionario que para cada *Node* v almacena, sobre la marcha, su vecindario con el rango. Este diccionario actúa como una especie de caché para optimizar el tiempo de cómputo.

Entonces, si los vecinos del nodo tomado de la lista fueron ya cargados en dicho diccionario, el algoritmo los trae de allí, en caso contrario los busca utilizando la función *nodesReachableFrom* y los almacena en el conjunto *adjs* con el fin de evitar repetir el paso en futuras iteraciones. Dicha función itera sobre el conjunto $\delta^+(v.id)$ en el grafo G, quedándose sólo con aquellos vértices que son alcanzables a través de un rango de *slots* incluidos en el rango de *slots* del *Node* v, *i.e.*, en $v.range$. Si un nodo vecino a $v.id$ puede ser alcanzado por medio de un arco con dos o más rangos disjuntos de *slots*, se lo toma como si estuviera conectado con varios arcos.

El algoritmo continúa luego calculando el rango de intersección para el nodo adyacente seleccionado (*adj*), esto es, los *slots* en los cuales coinciden. El siguiente condicional, i.e., la función *visitedWithSuperRange* determina si *adj* ya fue visitado con un rango mayor o igual que el calculado. En caso negativo, entonces, después de anotar dicha visita en el conjunto, el algoritmo revisa si el rango puede satisfacer la demanda, es decir si es mayor o igual a su volumen. Si esto es cierto agrega el nuevo *Node* n tal que $n.id = adj$ al final del árbol, como una hoja del *Node* v. Finalmente, si $n.id$ es el destino de la demanda, el algoritmo recupera el camino más angosto y retorna. Esto último lo hace la función *narrowestPath* recorriendo el árbol desde la hoja $t(d)$ hasta el origen $s(d)$ y utilizando el rango de $v(d)$ *slots* consecutivos comenzando por el primer *slot* del rango *rFromVtoAdj*.

Verificadores de límite inferior, densidad e no factibilidad Con el fin de tener

una cota contra la cual comparar la calidad de una solución en aquellos casos en los cuales el óptimo no está disponible, calculamos por cada instancia el resultado de la función objetivo de la solución que asigna el camino mínimo a cada demanda asumiendo infinitos *slots* disponibles por arco, *i.e.*, la cota inferior. Los resultados mostraron que este valor es muy útil incluso para determinar optimalidad. Si la función objetivo de una solución –proveniente del solver esclavo o del primario– iguala esta cota inferior, dicha solución puede ser considerada óptima, dado que no hay modo de mejorarla, ni reduciendo los *slots* de cada demanda, los cuales son exactamente su volumen, ni reduciendo los caminos asignados, que son los más cortos. La rutina encargada de realizar esta comparación es denominada *verificador de cota inferior* (LB-*checker*). Cuando el optimizador es utilizado, esta comparación es realizada antes y después de ejecutar dicha rutina.

Utilizando la misma relajación del problema, una cota inferior al número de *slots* requerido por cualquier solución factible es calculada antes de comenzar la heurística. Este valor es usado para filtrar instancias no factibles que requieren más *slots* que los disponibles. Denominamos a la rutina que realiza dicha comparación como *verificador de densidad* (D-*checker*). Finalmente, llamamos *verificador de no factibilidad* (I-*checker*) al mecanismo por el cual la heurística define una instancia como *probablemente no factible*.

Salida Si la heurística, debido al tamaño del modelo o del árbol de búsqueda, se queda sin memoria, la instancia es etiquetada como *memoria*. En esta configuración, los posibles estados que la heurística propuesta puede devolver son los siguientes: *óptima* (OPT) cuando el valor objetivo iguala la cota inferior, *factible* (FAC) cuando una solución factible pero sin garantía de optimalidad fue hallada dentro del límite de tiempo, *probablemente no factible* (PNFAC) cuando la cantidad de subproblemas sin resolver excede la cota de no factibilidad, *no factible* (NFAC) cada vez que el número de *slots* requeridos por las demandas sea mayor que la capacidad total de la red y *memoria* (MEM), cuando la memoria disponible no es suficientemente grande. Asimismo al presentar los resultados, agregamos la categoría *desconocida* (DESC) cuando no se halló ninguna solución factible, y la cual también incluye a todas las ejecuciones que retornan *memoria* o *probablemente no factible*.

5.3. Resultados computacionales

En esta sección presentamos la experiencia computacional con la heurística descripta en este capítulo, en combinación con la implementación del *branch-and-cut* presentada en el capítulo anterior. La implementación se realizó dentro del entorno Cplex 12.10 y los experimentos se llevaron a cabo en una computadora con una CPU Intel (R) Xeon (TM) 2.80GHz con 4 GB de memoria RAM. Para el *branch-and-cut* activamos todas las funciones de *pre-solve* y heurísticas primales de Cplex, excepto para unos pocos conjuntos específicos de pruebas, en las que desactivamos el *pre-solver*. También utilizamos Cplex como heurística al fijar el parámetro *solLim* en 1, es decir, hasta hallar la primera solución entera. Llamamos a esta configuración particular como *full-Cplex-feasibility (FCF)*.

Los experimentos fueron realizados sobre los seis conjuntos de instancias presentados en la Sección 4.3.1. Estos son *low-density*, con 22 instancias, *lowest-density*, con 100 instancias, *medium-density*, con 160 instancias, *sub-med-density*, con 65 instancias, *full-range-density*, con 250 instancias y *high-density*, con 39 instancias.

5.3.1. Coeficiente $\bar{\tau}$

Para comparar los resultados de la heurística definimos el coeficiente $\bar{\tau}$ para cada ejecución de forma análoga a la definición del coeficiente τ utilizado para comparar las estrategias del *branch-and-cut*. Sea t el tiempo de ejecución en minutos y $p = t/10$ un coeficiente de penalización. Sean $\bar{x}$ y x^* el valor de la mejor solución entera hallada y el valor de la solución óptima cuando éste es conocido, respectivamente. Definimos $g = \frac{\bar{x}}{x^*}$. Cuando el óptimo x^* no es conocido asumimos el peor caso aproximándolo mediante la cota inferior, *i.e.*, asignándole a cada demanda un camino mínimo. De esta manera, el coeficiente $\bar{\tau}$ es definido por

$$\bar{\tau} = \begin{cases} t & \text{si retorna óptima, no factible o probablemente no factible} \\ & \text{(conocido o no),} \\ t + p * g & \text{si fue hallada una solución factible,} \\ 2t & \text{si el resultado es desconocido,} \\ 4t & \text{si retorna } \textit{memoria} \text{ o si retorna } \textit{probablemente no factible} \\ & \text{para una instancia que se conoce factible.} \end{cases}$$

De este modo, penalizamos las instancias con problemas de memoria y los falsos positivos al sugerir no factibilidad. Asimismo, y en menor grado, penalizamos la incertidumbre y, por debajo de ella, sólo la factibilidad según la diferencia entre el valor obtenido y el óptimo conocido. Cuanto menor sea el valor del coeficiente $\bar{\tau}$, mejor será el resultado. El coeficiente total $\bar{\tau}$ de múltiples ejecuciones se calcula como la suma del mejor coeficiente para cada ejecución particular.

5.3.2. Configuración del *solver* esclavo

En esta sección presentamos los experimentos llevados a cabo para elegir la mejor configuración para el *solver* esclavo. Los mismos se ejecutaron sobre el conjunto de 160 instancias, *i.e.*, el llamado *medium-density*, con un límite total de tiempo fijado en 10 minutos. Asimismo se estableció un límite de tiempo holgado para el *solver* esclavo, con el parámetro de calibración fijado como $\gamma = 0.5$. Las tres configuraciones del *solver* esclavo comparadas fueron

- **FC:** *Full-Cplex*, *i.e.*, Cplex con la configuración por defecto, con *re-start*, *pre-solver*, heuristicas, paralelización y cortes genéricos;
- **ME-BAC:** el *branch-and-cut* desarrollado en capítulos previos aplicado sobre el *branch-and-bound* implementado por Cplex y utilizando la estrategia *Most Efficient* (ME) con 10 familias preordenadas; y
- **FE-ME-BAC:** *FC* combinado con ME-BAC.

Al observar la similitud de resultados entre agregar o no los cortes propuestos a la configuración FC y la diferencia con usar sólo ME-BAC, supusimos que, además de las heurísticas, la paralelización y las otras funcionalidades que Cplex desactiva cuando se proporciona una *user-callback* para manejar los cortes, la gran ventaja radicaba principalmente en su *pre-solver*. Para intentar fundamentar empíricamente este argumento, se agregó esta variación a la comparación, a saber,

- **FC-NPS:** FC sin *pre-solver*.

La Tabla 5.1 muestra el tiempo en horas, el coeficiente $\bar{\tau}$, la cantidad de instancias resueltas, y el número de instancias sin resultado. En este segundo grupo, a su vez, se discriminan las instancias etiquetadas como *probablemente no factibles* y aquellas con problemas de memoria.

Solver	$\bar{\tau}$	Tiempo	Resuelta	Desconocida	PNF	MEM
FC-NPS	2185.980	17.448	56	104	20	13
FC	1127.126	8.408	96	64	23	1
FC-ME-BAC	1135.654	8.797	96	64	23	0
ME-BAC	2428.267	20.609	24	136	24	3

Tab. 5.1: Comparación de las cuatro configuraciones para el *solver* esclavo sobre el subconjunto de instancias denominado *Medium-dense*.

Como podemos ver, la diferencia entre las dos mejores configuraciones es mínima, resultando un poco mejor no agregar los cortes a Cplex, a pesar de que da problemas de memoria para una instancia. A partir de este hecho y mirando los valores de la configuración sin *pre-solver* y la del *branch-and-cut* propuesto en este trabajo, llegamos a la conclusión de que *pre-solver* pareciera ser la parte más importante del *solver* para hallar rápidamente soluciones factibles. Sin embargo, un hecho interesante es que dicha configuración, *i.e.*, FC-NPS, fue la única que halló el óptimo en 13 de las 160 instancias evaluadas. Es decir, la primera solución encontrada con esta configuración de esclavo tenía el mismo valor objetivo que la cota inferior. Este hecho creemos se debe a la cantidad de ciclos y caminos espurios de las soluciones halladas por el *pre-solver* de Cplex en las otras configuraciones. En la siguiente subsección veremos esto más en detalle. También es importante notar que el *branch-and-cut* propuesto en esta tesis fue calibrado para resolver el RSA por lo que, dado que aquí estamos buscando resolver un SA casi puro, dejamos como trabajo a futuro probar si mejora el comportamiento del esclavo al re-ajustar sus parámetros para este problema particular.

Respecto a la no factibilidad, es notable que las 25 instancias que alguna de las variaciones de la heurística etiquetó como *probablemente no factibles* resultaron de hecho no factibles, sin ningún falso positivo.

5.3.3. Optimizador y verificador de cota inferior

Al añadir el optimizador a la heurística se mejoran notablemente los valores de las soluciones resultantes y, agregando el verificador de cota inferior luego de la optimización, es posible determinar la optimalidad de gran parte de las instancias directamente al final de la heurística.

Para evaluar el impacto de estas dos mejoras, se ejecutó la variación de la heurística con FC como esclavo, con un tiempo límite de 10 minutos sobre todos los conjuntos de instancias, *i.e.*, sobre las 549 instancias.

Del total de 549 ejecuciones, descartamos aquellas etiquetadas como *no factibles* o *probablemente no factibles*, y aquellas sin resultado. De los 228 resultados restantes, 50 fueron mejorados por el optimizador, es decir, un 21,93 %, mejorando la suma total de sus funciones objetivo en 992 arcos, lo que representa un 18,92 % de la cantidad total de arcos, y quedando sólo a 198 arcos sobre la suma total de las cotas inferiores. Esto

	No mejoradas	Mejoradas						
	Cantidad	Cantidad	Valor total de la func. obj.			Tiempo [seg.]		
			Esclavo	Optimizadas	LB	Heurística	Optimizador	LB-*checker*
Factible	37	45	4922	3949	3751	2391.785	1.089	0.005
Óptima	141	5	322	303	303	284.626	0.037	0.000
Resuelta	178	50	5244	4252	4054	2676.411	1.126	0.005

Tab. 5.2: Resultados de agregar el optimizador y el verificador de cota inferior a la heurística con FC como esclavo. Los experimentos fueron realizados con un tiempo límite de 10 minutos sobre el total de 549 instancias, seleccionando sólo aquellas para las cuales se obtuvo una solución (*i.e.*, óptima o factible).

sugiere que la cota inferior es adecuada, al menos para los casos resueltos por la heurística. Asimismo, del total de 228 instancias resueltas, se determinó la optimalidad de 146 de ellas, *i.e.*, un 62,28 %, porque sus funciones objetivo igualaron a la cota inferior, 5 de las cuales habían sido mejoradas por el optimizador. El optimizador removió en promedio 20 arcos por instancia, alcanzando 160 arcos en la que más eliminó. Estos resultados se presentan en la Tabla 5.2. En ella tenemos dos grandes columnas y, a su vez, para cada una de ellas separamos los resultados según fueran factibles u óptimos. La primera columna representa la cantidad de instancias resueltas para las que el optimizador no pudo mejorar los resultados, mientras que la segunda columna muestra las instancias mejoradas por el optimizador. Ésta se divide en tres grupos. El primer grupo muestra la cantidad de instancias. El segundo grupo de columnas presenta las sumas de los diferentes valores para la función objetivo, es decir, los valores devueltos por el *solver* esclavo, los valores obtenidos después del proceso de optimización y el mejor valor posible cuando $\bar{s}$ es relajado, *i.e.*, la cota inferior. Aquí podemos ver cuánto se mejoraron las soluciones y qué tan cerca están sus valores resultantes del límite teórico, *i.e.*, de la cota inferior. La tercera subcolumna para las instancias mejoradas corresponde al tiempo dedicado a cada subproceso. Como puede verse, los tiempos de optimización y verificación son insignificantes en comparación con el tiempo total requerido por la heurística.

5.3.4. Comparación de las heurísticas

La mejor configuración de la heurística PSP, es decir, utilizando FC como esclavo, con el optimizador y los verificadores de cota inferior y de densidad, se comparó con *full-Cplex-factibility* (FCF) es decir, la configuración predeterminada de Cplex con paralelización, *re-start*, *pre-solver*, heurísticas y cortes genéricos, ejecutando hasta la primera solución entera o hasta que se confirme la no factibilidad. Para estos experimentos hemos utilizado nuevamente todos los conjuntos de instancias. Los resultados obtenidos se pueden ver en la Tabla 5.3.

Podemos observar que el coeficiente $\bar{\tau}$ obtenido por PSP es más de un tercio mejor que el obtenido por Cplex. También podemos notar que encontramos soluciones factibles para más instancias, resultando 146 óptimas, que es más de un tercio más que la cantidad encontrada por Cplex. La pequeña cantidad de casos con problemas de memoria también es remarcable.

En estos resultados se vuelve a apreciar la gran ventaja del verificador de densidad. De

	$\bar{\tau}$	Tiempo	OPT	FAC	DESC	PNF	NFAC	MEM
FCF	9006.37	67.32	95	66	315	0	73	100
PSP	5983.98	48.77	146	82	311	71	10	33

Tab. 5.3: Comparación entre la mejor configuración de PSP y FCF sobre el total de 549 instancias.

las 8 instancias que PSP marcó como no factibles por tener una densidad superior al 100 %, Cplex pudo determinar la no factibilidad sólo de 7, para lo cual requirió 727 segundos. Asimismo, podemos observar la ventaja de agregar el verificador de cota inferior, el cual fue capaz de determinar fácilmente la optimalidad de 146 soluciones en 1 hora y 20 minutos, de las cuales Cplex, en más de 10 horas de ejecución, pudo resolver sólo 76 por optimalidad, encontró solución factible a 26, y tuvo problemas de memoria con 7 de las 44 restantes que terminaron como *desconocidas*.

También es importante aclarar que las instancias descubiertas como no factibles por esta configuración de Cplex se suman como resueltas, mientras que las etiquetadas como *probablemente no factibles* por la heurística PSP suman como *desconocidas*; y, dado que 69 de estas instancias fueron halladas de hecho no factibles por algún algoritmo durante los experimentos del presente trabajo (las dos instancias restantes aún no pudieron ser resueltas por ningún algoritmo), se podría decir que la diferencia real entre las heurísticas es incluso mayor que la expresada por la Tabla 5.3. También cabe destacar que al medir los tiempos de Cplex no estamos sumando el tiempo destinado a construir el modelo, mientras que para la heurística se tienen en cuenta todos los tiempos de modelado de los subproblemas.

La heurística propuesta supera a Cplex no sólo en la cantidad de instancias resueltas con optimalidad, sino especialmente en el tiempo necesario para encontrar cada solución óptima. Midiendo los tiempos para las 76 instancias que tanto Cplex como PSP marcaron como óptimas podemos observar que Cplex no resolvió ninguna antes que PSP. Más aún, el total del tiempo para resolver esas instancias es de 14 segundos frente a más de 2 horas, respectivamente. En la Figura 5.3 mostramos los tiempos ordenando las instancias por densidad.

Para medir la calidad de las soluciones factibles –subóptimas o al menos no confirmada su optimalidad– encontradas por las heurísticas, no podemos usar el *gap* devuelto por el *solver* esclavo porque hemos mejorado más de la mitad de ellas con el optimizador; es decir, el *gap* devuelto por el *solver* probablemente esté lejos del *gap* real de la solución una vez mejorada. Por lo tanto hacemos la comparación midiendo la diferencia con la cota inferior. Calculamos entonces el *gap* como el porcentaje de la cota inferior que representa la diferencia entre el resultado y la cota inferior, es decir,

$$\text{gap} \;=\; 100\,\frac{\text{solución} - \text{cota inferior}}{\text{cota inferior}}.$$

Como primer resultado, es destacable que el máximo de la distancia entre las soluciones halladas por PSP y la cota inferior propuesta es de 22 arcos, mientras que el máximo para FCF es de 151 arcos. La diferencia en los promedios está en el mismo orden, es decir, alrededor de 3 contra 17. Estos resultados se vuelven más significativos si pensamos que estamos tomando en cuenta 82 instancias para PSP y sólo 66 para FCF.

Si nos quedamos sólo con las 40 instancias para las que ambas heurísticas devolvieron

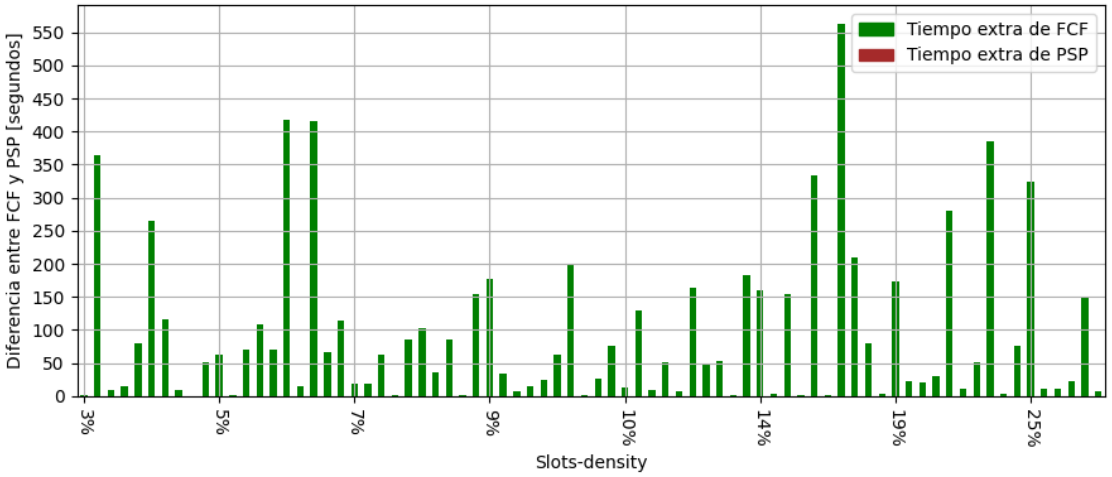


Fig. 5.3: Diferencia entre el tiempo que necesita FCF y el requerido por PSP para las 76 instancias en las que ambos encontraron la solución óptima dentro del límite de tiempo. La heurística propuesta supera a Cplex en todos los casos.

factible –sin saber por lo tanto si la solución es óptima–, entonces podemos hacer una gráfica como se ve en la Figura 5.4 que muestre la diferencia entre los *gaps* de esos resultados, tomando, como se mencionó, el *gap* como el porcentaje de la cota inferior que separa el resultado de dicho límite. La heurística PSP supera claramente a FCF, especialmente en instancias de baja densidad. Existe la posibilidad de que las soluciones factibles halladas por la heurística –que están cerca de la cota inferior– sean también óptimas.

Rango de densidad	Tot.	FCF					PSP				
		% OPT	% FAC	% NFAC	% DESC	% MEM	% OPT	% FAC	% PNF	% DESC	% MEM
0 % to 10 %	41	80.49 %	4.88 %	0.0 %	14.63 %	2.44 %	95.12 %	4.88 %	0.0 %	0.0 %	0.0 %
10 % to 20 %	74	55.41 %	13.51 %	0.0 %	31.08 %	8.11 %	78.38 %	20.27 %	0.0 %	1.35 %	0.0 %
20 % to 30 %	59	22.03 %	22.03 %	0.0 %	55.93 %	13.56 %	50.85 %	27.12 %	0.0 %	22.03 %	0.0 %
30 % to 40 %	55	7.27 %	21.82 %	3.64 %	67.27 %	18.18 %	12.73 %	34.55 %	3.64 %	52.73 %	1.82 %
40 % to 50 %	66	0.0 %	33.33 %	9.09 %	57.58 %	10.61 %	15.15 %	28.79 %	10.61 %	56.06 %	3.03 %
50 % to 60 %	58	5.17 %	8.62 %	15.52 %	70.69 %	25.86 %	3.45 %	13.79 %	13.79 %	82.76 %	12.07 %
60 % to 70 %	53	1.89 %	3.77 %	16.98 %	77.36 %	26.42 %	0.0 %	5.66 %	13.21 %	94.34 %	7.55 %
70 % to 80 %	54	0.0 %	0.0 %	27.78 %	72.22 %	27.78 %	0.0 %	0.0 %	33.33 %	100.0 %	14.81 %
80 % to 90 %	41	0.0 %	0.0 %	34.15 %	65.85 %	34.15 %	0.0 %	0.0 %	39.02 %	100.0 %	12.2 %
90 % to 100 %	38	0.0 %	0.0 %	26.32 %	73.68 %	23.68 %	0.0 %	0.0 %	34.21 %	100.0 %	15.79 %

Tab. 5.4: Comparación de la mejor configuración de la heurística PSP contra FCF sobre un total de 549 instancias agrupadas por densidad (menor o igual al 100 %), con un límite de tiempo de 10 minutos.

Si discriminamos las instancias por densidad como se muestra en la Tabla 5.4, podemos observar que la heurística propuesta tiene un mejor desempeño para todas las instancias pero especialmente para aquellas con una densidad menor al 50 %. Cplex encontró mayor cantidad de soluciones óptimas sólo para algunas de las instancias con una densidad de entre el 50 % y el 70 %, pero la cantidad de casos con problemas de memoria fue menor para PSP. Podemos apreciar que la cantidad de instancias desconocidas, si también sumamos

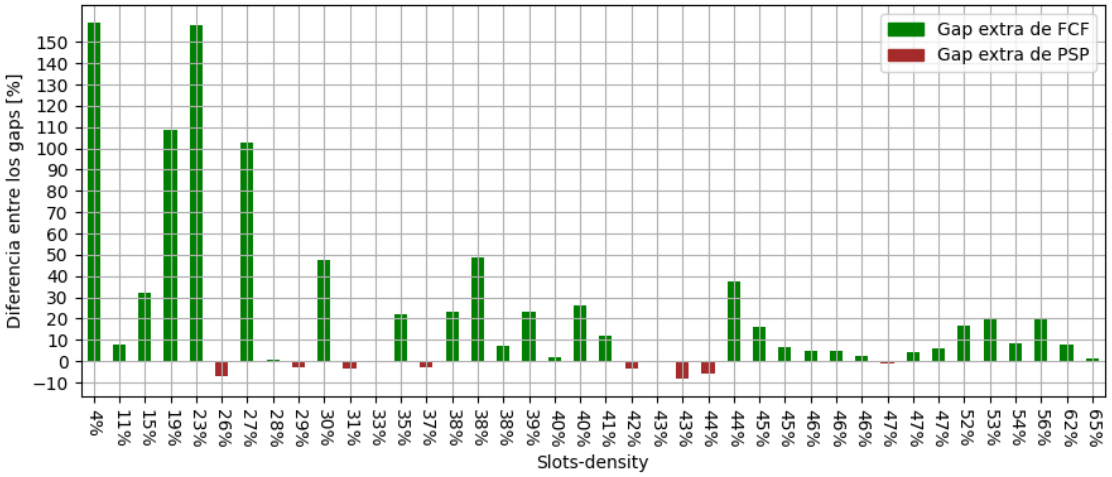


Fig. 5.4: Diferencia entre los *gaps* de las soluciones factibles obtenidas por FCF y las obtenidas por PSP para las instancias que arrojaron una solución factible (pero no determinada su optimalidad). El *gap* se calcula como el porcentaje de la cota inferior que separa el resultado de la cota inferior.

las *probablemente no factibles* como resueltas, es menor excepto por el rango entre 60 % y 70 %.

5.3.5. Calibración del algoritmo completo

El siguiente paso en la experimentación es comparar el desempeño del algoritmo completo agregando la heurística PSP al *branch-and-cut* desarrollado y descripto en capítulos anteriores utilizando el resultado proporcionado por la heurística como *warm-start*. Para estos experimentos se utilizó el subconjunto de 160 instancias, *i.e.*, *medium-density*, y un límite de tiempo de 10 minutos. Ejecutamos el mismo conjunto de instancias con el mismo límite de tiempo para las diferentes variaciones *branch-and-cut* y también para *Full-Cplex procedure* (FCP) con el fin de comparar la calidad de los resultados. En cuanto a las heurísticas, se utilizó FC como *solver* esclavo con un factor de precisión $\gamma = 0.5$ para su límite de tiempo, y se desactivaron tanto el verificador de cota inferior como el optimizador. Con esta heurística se probaron las dos mejores estrategias del *branch-and-cut* propuestas en capítulos anteriores, es decir, ME y MER, con $h = 5$ y $h = 10$, y con el preordenador de familias activado. Los resultados obtenidos se pueden ver en la Tabla 5.5, de los cuales se desprende claramente que las cuatro configuraciones propuestas superan a Cplex en casi todas las categorías. Los $\bar{\tau}$ obtenidos con todas las variaciones de FRSAP son un tercio menores que los obtenidos con FCP. Casi no hay instancias con problemas de memoria, al menos un tercio más de instancias se resolvieron hasta la factibilidad o no factibilidad, y el óptimo se alcanzó en hasta 12 instancias más que con FCP. La única desventaja sigue siendo el número de instancias no factibles detectadas. Creemos que esto se debe casi exclusivamente al *pre-solver* de Cplex. En cuanto a la comparación entre las distintas estrategias, si sólo nos fijamos en el coeficiente $\bar{\tau}$, resulta que la mejor es la que tiene menor cantidad de instancias resueltas y en segundo lugar la que tiene mayor número de *memoria*, coincidiendo con los resultados obtenidos en el capítulo anterior sin

heurística primal.
Dado que Cplex ofrece un parámetro para decidir si el énfasis del algoritmo es puesto más

algoritmo	B&C estrategia	familias	Tiempo [hs]	Coef. $\bar{\tau}$	Res.	DESC	FAC	OPT	NFAC	PNF	MEM
FCP	-	-	18.04	2116.71	92	70	19	48	25	0	18
FRSAP	ME	5.0	16.03	1476.19	120	42	40	58	22	23	2
		10.0	16.39	1457.40	119	43	42	55	22	25	0
	MER	5.0	15.87	1449.13	118	44	34	60	24	24	0
		10.0	16.71	1504.89	121	41	45	53	23	23	1

Tab. 5.5: Comparación entre FCP y cuatro variaciones para FRSAP en 162 instancias densas de tamaño mediano. La tabla muestra el tiempo en horas, el coeficiente $\bar{\tau}$ y la cantidad de instancias resueltas o las sin resultado. En el primer grupo discrimina la cantidad de factible, óptima o no factible, y en el segundo hay algunas con problemas de memoria. Las marcadas como *probablemente no factibles* por las heurísticas pertenecen a ambos grupos.

sobre la factibilidad o sobre la optimización, *i.e.*, el parámetro *énfasis*, también hemos probado modificar este parámetro para el *branch-and-cut* del *solver* primario de acuerdo con el estado de la solución proporcionado por la heurística. Si la instancia es *probablemente no factible* según ésta, se pone mayor énfasis en la factibilidad, mientras que si es *factible* se hace foco en la optimalidad. El mecanismo que realiza esta comparación y modificación se denomina *modificador de énfasis*. Para medir el impacto de esta modificación, se seleccionaron del grupo *medium-density* todas las instancias con una densidad inferior al 60 % tales que la heurística devolvió *factible* o *probablemente no factible* (*i.e.*, el conjunto *sub-med-density*). El conjunto resultante quedó entonces integrado por 55 instancias del primer grupo y 10 del segundo. Los resultados de ejecutar la mejor versión del algoritmo propuesto con optimizador y verificador de cota inferior sobre el mencionado conjunto de instancias se pueden ver en la Tabla 5.6. Se eligió la estrategia *Most Effective Rnd.* (MER) con cinco familias de cortes, y se compararon las opciones de agregar el optimizador y el verificador de cota inferior (tanto al de la heurística como al *solver* primario) así como también establecer o no el énfasis del *solver* primario.

Optimizador y LB-checker	Énfasis	Coef. $\bar{\tau}$	time [hs]	OPT	NFAC	PNF	FAC	DESC
Desactivado	Desactivado	528.67	6.89	22	9	9	28	6
Activado	Desactivado	519.56	6.68	21	10	9	28	6
Activado	Activado	483.22	6.41	23	10	9	28	4

Tab. 5.6: Comparación entre agregar el optimizador y/o el modificador de énfasis al algoritmo completo que usa la estrategia MER con 5 familias de cortes. La tabla muestra el tiempo en horas, el coeficiente $\bar{\tau}$ y la cantidad de instancias resueltas o sin resolver. La cantidad total de instancias tenidas en cuenta es de 65, tomadas del conjunto *medium-density*, siendo las que la heurística retornó *factible* o *probablemente no factible*.

Para analizar el impacto de agregar estas variaciones para determinar la no factibilidad,

filtramos las nueve instancias que las tres configuraciones detectaron como no factibles. En casi todas, la última modificación determina la no factibilidad mucho antes que las otras variaciones, requiriendo sólo 1097.318 segundos frente a los 1132.876 segundos que requiere el que usa sólo el optimizador y los 1296.693 segundos del que no usa ninguna de las mejoras; es decir, alrededor de un 5 % y un 20 % mejor que éstos respectivamente. Además, mirando las ocho instancias para las que Cplex también determina la no factibilidad, los tiempos de la mejor configuración propuesta son un 10 % mejores que los tiempos del *solver* comercial.

En cuanto a la optimalidad, en 29 del total de 65 instancias el *gap* fue *cerrado –i.e.*, se llevó a 0 encontrando el óptimo– con al menos una configuración. En sólo cinco de estas instancias la diferencia fue de 1 ó 3, mientras que en las 24 restantes el valor resultante coincidió con la cota inferior. El hecho de que estas instancias no hayan sido resueltas por la heurística es un aliciente para mejorarla buscando variaciones en los caminos mínimos de cada demanda. Asimismo, el hecho de que la configuración con optimizador y modificador de énfasis haya hallado el óptimo para 23 de ellas sugiere la utilidad de agregar también el verificador de cota inferior al *solver* primario.

También se puede ver que al variar el énfasis, los tiempos y el número de instancias totales resueltas mejoran levemente, dejando sólo 4 en la categoría *desconocida*.

5.3.6. Comparación de los algoritmos completos

Finalmente, se comparó la mejor configuración del algoritmo completo propuesto, a saber, *Full RSA Procedure* (FRSAP) con FCP con un límite de tiempo de 15 minutos sobre un total de 549 instancias. Los resultados se pueden ver en la Tabla 5.7. A partir de estos primeros resultados y mirando los obtenidos por el *branch-and-cut* implementado en el capítulo anterior, podemos concluir que la heurística y las modificaciones posteriores mejoran el comportamiento del algoritmo completo. Recordando que el coeficiente $\bar{\tau}$ intenta condensar el comportamiento en un solo número, es notable que FRSAP supere a FCP en más de un cuarto en ese valor. Aquí también podemos ver la efectividad del detector de no factibilidad de la heurística, ya que predice dos tercios del número total de instancias no factibles, con sólo un falso positivo. Analizamos esta funcionalidad más adelante.

Algoritmo	Tiempo [hs]	Coef. $\bar{\tau}$	Res.	DESC	FAC	OPT	NFAC	PNF	MEM
FCP	89.847	13466.468	256	293	33	137	86	-	114
FRSAP	81.315	10418.294	312	236	52	170	90	65	61

Tab. 5.7: Comparación de FRSAP vs FCP sobre el conjunto completo de 549 instancias fijando el límite de tiempo en 15 minutos.

Si eliminamos aquellas instancias para las cuales ambos algoritmos devolvieron *desconocida* (sin importar si fue por problemas de memoria), obtenemos el conjunto de 323 instancias que se discriminan en la Tabla 5.8. De esta forma, se puede apreciar más claramente la mejora relativa de FRSAP. Como resumen, podemos ver un coeficiente $\bar{\tau}$ que es menos de la mitad del obtenido con Cplex. También se puede ver que FCP sólo resolvió una instancia (no factible), la cual resultó en problemas de memoria con FRSAP, mientras que el primero tuvo problemas con 20 instancias que FRSAP no. Aún más, 14 de estas 20

instancias fueron resueltas de manera óptima por FRSAP, cuatro resultaron no factibles y para las otras dos el algoritmo podría devolver una solución factible si se ampliara el límite de tiempo.

Algoritmo	Tiempo [hs]	Coef. $\bar{\tau}$	Res.	DESC	FAC	OPT	NFAC	PNF	MEM
FCP	35.446	3866.468	256	67	33	137	86	0	20
FRSAP	26.368	1883.284	312	11	52	170	90	64	1

Tab. 5.8: Comparación de FRSAP vs FCP sobre el conjunto completo de 549 instancias fijando el límite de tiempo en 15 minutos.

Es interesante notar que FCP pudo resolver con optimalidad sólo 11 instancias que FRSAP no pudo, mientras que FRSAP pudo encontrar 44 que FCP no pudo. Además, para 10 de esas 11, FRSAP halló una solución factible, mientras que FCP devolvió *desconocida* para 39 de las 44 mencionadas (dando problemas de memoria en 14 de ellas). Si nos enfocamos en las 126 instancias resueltas con optimalidad por ambos algoritmos, el tiempo que necesitó FRSAP es casi la mitad del requerido por FCP, siendo el primero sólo de 2h 13m 37.78s contra 4h 13m 58.71s del segundo. La Figura 5.5 muestra en verde el tiempo extra que Cplex necesitó para encontrar la solución óptima una vez que FRSAP lo hizo, y en rojo lo contrario. Es interesante que ambos algoritmos pudieron encontrar el óptimo para instancias con una densidad de casi el 60 %. También hay que tener en cuenta que debido a la heurística, FRSAP encuentra mucho más rápido el óptimo para las instancias de baja densidad, es decir, aquellas con hasta un 34 % de arcos requeridos.

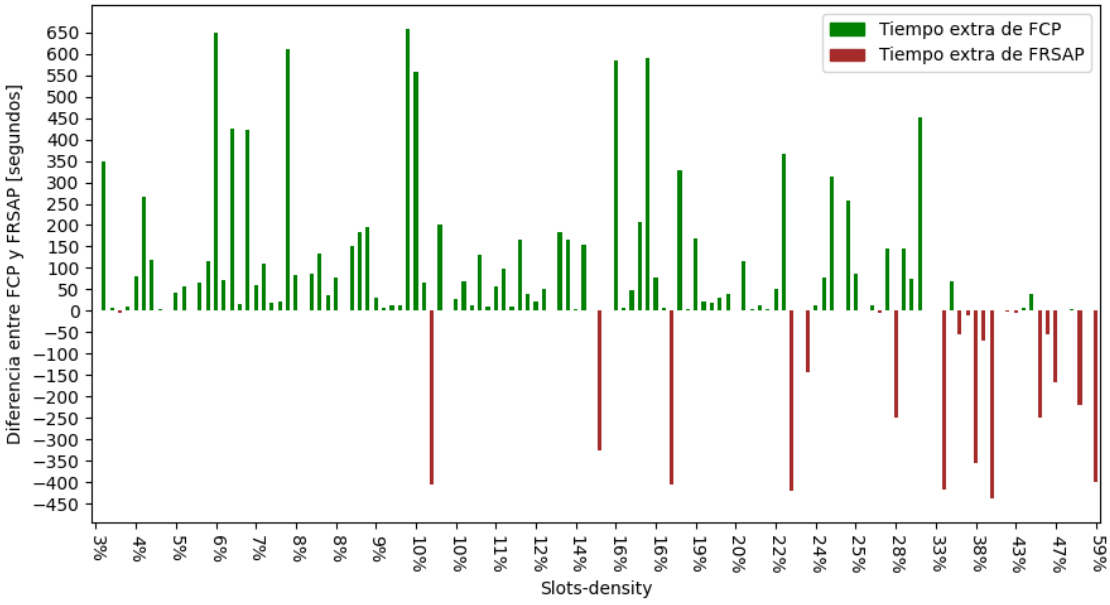


Fig. 5.5: Diferencia entre el tiempo que necesita FCP y el que necesita FRSAP para las instancias en las que ambos encontraron la solución óptima dentro del límite de tiempo, en función de la densidad de la instancia.

En cuanto a la calidad de las soluciones obtenidas, los experimentos parecen sugerir que FRSAP también supera a FCP. El algoritmo propuesto arrojó una solución factible (no

necesariamente óptima) para un total de 52 instancias con un *gap* promedio del 8.094 % con un máximo de 100 % y un mínimo de 0.47 %. Cplex resolvió 33 instancias obteniendo un *gap* medio del 10,42 %, un mínimo del 0,54 % y también un máximo del 100 %. Si nos enfocamos sólo en las 25 instancias marcadas como factibles por ambos algoritmos y comparamos el *gap* de sus soluciones, ambos son muy parejos. La suma de los *gaps* de FRSAP es de 114 % y la de Cplex es de 111 % dando una media de 4,566 % y 4,44 %, respectivamente. Sin embargo, el peor *gap* obtenido por Cplex es del 21,03 % mientras que el obtenido por FRSAP es del 15,31 %. La Figura 5.6 muestra la diferencia entre los *gaps* ordenando las instancias de menor a mayor densidad. Resulta interesante ver que FRSAP pareciera comportarse mucho mejor en instancias de baja densidad, mientras que Cplex lo estaría haciendo para las de densidad media. De todas formas creemos que la cantidad de instancias no es lo suficientemente grande para medir correctamente la diferencia.

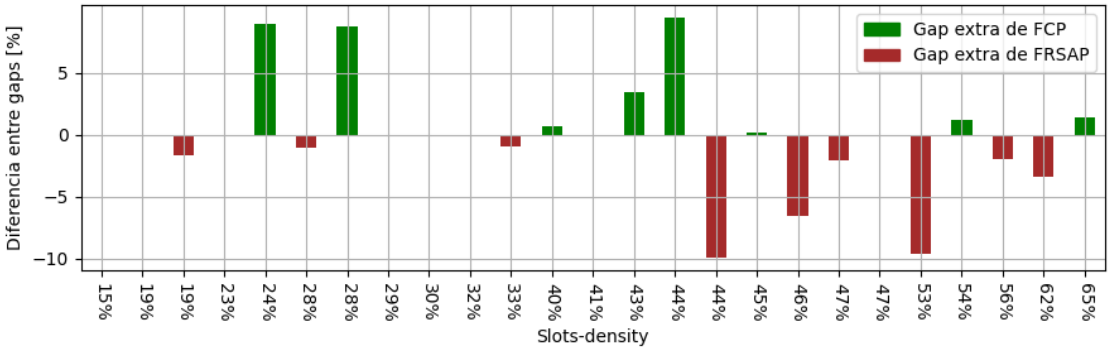


Fig. 5.6: Diferencia entre los *gaps* de las soluciones factibles obtenidas por FCP y las obtenidas por FRSAP para las instancias que ambas marcaron como *factibles* dentro del límite de tiempo en función de la densidad de la instancia.

Para un estudio más profundo, los resultados de la Tabla 5.7 fueron discriminados dividiendo las instancias por grupos según su densidad. El detalle se puede ver en la Tabla 5.9. A partir de los resultados, parece estar claro, como fue con el caso de la heurística y como consecuencia de ello, que FRSAP tiene un comportamiento mucho mejor en instancias con densidad baja y media. De hecho, si miramos las 229 instancias con una densidad menor o igual al 40 %, FRSAP resuelve el 65 % de las instancias con optimalidad frente al 50 % de FCP, y reduce a la mitad el número de instancias sin resolución, dejando sólo el 20 % frente al 40 % de Cplex. Para instancias de densidad media a alta, FRSAP no difiere mucho de FCP, y lo supera ligeramente en términos de instancias resueltas. Sin embargo, donde se obtiene una mejora significativa en estos casos es en la cantidad de *memoria*, que se reduce a casi la mitad con FRSAP.

La Tabla 5.10 muestra los resultados de las 229 instancias mencionadas con densidad baja a media, discriminadas por topología. Allí podemos ver claramente la ventaja de FRSAP sobre FCP en todas las categorías.

Beneficios del verificador de cota inferior propuesto

Si nos centramos únicamente en las 189 instancias resueltas con optimalidad en el presente trabajo, para 171 de ellas el valor resultante coincide con la cota inferior pro-

Rango de densidad	Tot.	FCP % OPT	FCP % FAC	FCP % NFAC	FCP % DESC	FCP % MEM	FRSAP % OPT	FRSAP % FAC	FRSAP % NFAC	FRSAP % DESC	FRSAP % MEM
0 % to 10 %	41	87.80 %	0.00 %	0.00 %	12.20 %	4.88 %	100.00 %	0.00 %	0.00 %	0.00 %	0.00 %
10 % to 20 %	74	63.51 %	8.11 %	0.00 %	28.38 %	10.81 %	87.84 %	10.81 %	0.00 %	1.35 %	0.00 %
20 % to 30 %	59	40.68 %	8.47 %	0.00 %	50.85 %	20.34 %	54.24 %	22.03 %	0.00 %	23.73 %	8.47 %
30 % to 40 %	55	20.00 %	10.91 %	3.64 %	65.45 %	20.00 %	23.64 %	14.55 %	3.64 %	58.18 %	7.27 %
40 % to 50 %	66	19.70 %	15.15 %	9.09 %	56.06 %	12.12 %	21.21 %	21.21 %	7.58 %	50.00 %	7.58 %
50 % to 60 %	58	8.62 %	6.90 %	18.97 %	65.52 %	27.59 %	8.62 %	10.34 %	18.97 %	62.07 %	17.24 %
60 % to 70 %	53	1.89 %	3.77 %	18.87 %	75.47 %	28.30 %	0.00 %	5.66 %	22.64 %	71.70 %	16.98 %
70 % to 80 %	54	0.00 %	0.00 %	35.19 %	64.81 %	27.78 %	0.00 %	0.00 %	31.48 %	68.52 %	25.93 %
80 % to 90 %	41	0.00 %	0.00 %	36.59 %	63.41 %	36.59 %	0.00 %	0.00 %	43.90 %	56.10 %	19.51 %
90 % to 100 %	38	0.00 %	0.00 %	39.47 %	60.53 %	28.95 %	0.00 %	0.00 %	39.47 %	60.53 %	18.42 %
100 % to 230 %	10	0.00 %	0.00 %	80.00 %	20.00 %	10.00 %	0.00 %	0.00 %	100.00 %	0.00 %	0.00 %
Total	549	137	33	86	293	114	170	52	90	237	62
%Total	100.00 %	24.95 %	6.01 %	15.66 %	53.37 %	20.77 %	30.97 %	9.47 %	16.39 %	43.17 %	11.29 %

Tab. 5.9: Comparación de FRSAP vs FCP sobre un total de 549 instancias, con un límite de tiempo de 15 minutos.

Nodos	Tot.	FCP % OPT	FCP % FAC	FCP % NFAC	FCP % DESC	FCP % MEM	FRSAP % OPT	FRSAP % FAC	FRSAP % NFAC	FRSAP % DESC	FRSAP % MEM
6	24	75.00 %	8.33 %	4.17 %	12.50 %	4.17 %	83.33 %	8.33 %	4.17 %	4.17 %	0.00 %
10, 11, 14, 15, 16	52	73.08 %	7.69 %	0.00 %	19.23 %	9.62 %	80.77 %	11.54 %	0.00 %	7.69 %	1.92 %
19, 20, 21, 22	83	51.81 %	7.23 %	1.20 %	39.76 %	10.84 %	71.08 %	13.25 %	1.20 %	14.46 %	0.00 %
24, 28, 30	32	34.38 %	12.50 %	0.00 %	53.12 %	9.38 %	65.62 %	15.62 %	0.00 %	18.75 %	0.00 %
43	38	21.05 %	2.63 %	0.00 %	76.32 %	39.47 %	23.68 %	13.16 %	0.00 %	63.16 %	21.05 %

Tab. 5.10: Comparación de FRSAP vs FCP en las 229 instancias de densidad baja a media, con un límite de tiempo de 15 minutos, discriminando los resultados por topología.

puesta. Eso significa que todas sus demandas se satisfacen utilizando alguno sus caminos más cortos. Para los otros 18 casos, 11 de ellos difieren en un arco, cinco tienen dos arcos más que el límite más bajo y sólo uno difiere en tres arcos. Eso significa que la diferencia total es de 27 arcos con respecto a la cota inferior: 11261 contra 11288 arcos. Podemos ver estos resultados discriminados por densidad en la Tabla 5.11. Como consecuencia de estos resultados, no parece haber una relación fuerte entre la densidad y la diferencia con el límite.

En otras palabras, para casi todas las instancias resueltas, el porcentaje de caminos mínimos en sus soluciones óptimas es cercano al 100 %. Asimismo, analizando el número de arcos utilizados por esas 18 instancias que difieren, se observa que la cota inferior se encuentra a una distancia de a lo sumo un 15 % del óptimo en el peor de los casos, pero éste muy probablemente sea un valor atípico porque la cota está a una distancia del 1,83 % en promedio y del 0,41 % en el mejor de los casos.

Desempeño del verificador de no factibilidad

Sólo para 114 del total de ejecuciones del presente trabajo, el *solver* esclavo tuvo que iterar al menos una vez debido a la no factibilidad del subproblema, y 90 de ellas terminaron descubriéndose no factibles por algún algoritmo. La Tabla 5.12 muestra cómo aumenta el porcentaje de instancias no factibles cuando aumenta la cantidad de iteraciones (debido a la no factibilidad) del esclavo. Sólo una instancia terminó con solución factible a pesar de que el *solver* esclavo iteró más de cuatro veces. Este resultado apoya la decisión de

Rango de densidad	Cantidad de instancias			
	dif = 0	dif = 1	dif = 2	dif = 3
0 % to 10 %	41	0	0	0
10 % to 20 %	63	3	1	1
20 % to 30 %	33	1	2	0
30 % to 40 %	12	2	2	0
40 % to 50 %	17	3	0	1
50 % to 60 %	5	1	0	0
60 % to 70 %	0	1	0	0
70 % to 100 %	0	0	0	0

Tab. 5.11: Cantidad de instancias resueltas con optimalidad discriminadas por su densidad y la distancia –en cantidad de arcos– a la cota inferior. Aquí $dif = x^* - LB$.

declarar no factible una instancia con más de tres subproblemas no factibles, o de manera equivalente, cuando el factor de superposición toma un valor menor o igual al 70 %, *i.e.*, $overlap = 0.7$. Esa fue la configuración para la mayoría de los experimentos, fijando el *límite de no factibilidad* en 75 %.

Iteraciones por no factibilidad	Total de instancias	Cantidad de *no factibles*	Cantidad de *óptimas*	Cantidad de *desconocidas*
≥ 1	114	90	5	19
≥ 2	90	82	2	6
≥ 3	66	65	1	0
≥ 4	21	21	0	0
≥ 5	0	0	0	0

Tab. 5.12: Relación entre la no factibilidad del *solver* esclavo y primario.

Del total de 97 instancias determinadas como no factibles por algunos de los algoritmos, la heurística propuesta con cualquier configuración fue capaz de descubrir 79 etiquetándolas como *probablemente no factible*. Esto es interesante porque tenemos una cantidad total de 83 instancias etiquetadas con este rótulo. Una de las otras cuatro instancias fue resuelta de manera óptima sólo por FCP y para las otras tres instancias no tenemos resultados. Es importante señalar que el falso positivo estaba en una instancia con una densidad de 47,71 %, que está cerca del límite que somos capaces de resolver con los mejores algoritmos y usando un *límite de no factibilidad* muy estricto (*i.e.*, del 85 %). Este resultado se obtuvo en sólo cuatro de las 13 veces que la heurística propuesta ejecutó esta instancia, una con ME-BAC como *solver* esclavo y otra con Cplex sin *pre-solver* (las dos combinaciones de peor desempeño), siendo factible en las otras nueve. Además, la heurística propuesta encontró el resultado en el rango de 5 a 20 segundos cada vez, mientras que FCP tardó 81 segundos en tener la primera solución y más de 200 en alcanzar la óptima. Esto nos sugiere la idea de que el *límite de no factibilidad* podría ajustarse un poco más, quizás de acuerdo con la densidad, para evitar este tipo de valores atípicos.

Agregar una heurística primal que requiere demasiado tiempo generalmente trae problemas cuando la instancia no es factible. A pesar de los buenos resultados obtenidos en todos los demás aspectos, parecería ser conveniente intentar mejorar esta característica en el algoritmo propuesto. La Tabla 5.13 muestra la comparación del tiempo que necesitan

los algoritmos completos para determinar la no factibilidad de las 79 instancias que ambos fueron capaces de etiquetar de esa manera. Aquí, FCP necesitó una cuarta parte menos del tiempo requerido por FRSAP. El primero necesitó apenas un poco más de cuatro horas y el algoritmo propuesto en este trabajo requiere más de cinco horas. Observamos algo similar cuando comparamos el tiempo necesario para mostrar la no factibilidad sólo en las 65 instancias etiquetadas como *probablemente no factible* por la heurística. El tiempo aquí también es aproximadamente una cuarta parte más del que necesita FCP. El algoritmo propuesto necesitó 4.3 horas frente a las 3.57 requeridas por Cplex, que pudo resolver dos instancias más de las 60 determinadas por FRSAP, resultando en un coeficiente $\bar{\tau}$ de 333 contra los 259.5 de Cplex.

Algoritmo	Tiempo [hs]	Coef. $\bar{\tau}$	No Factible	Prob. No Factible
FCP	4.229	253.737	79	-
FRSAP	5.341	320.459	79	58

Tab. 5.13: Comparación de FRSAP vs FCP sobre el subconjunto de 79 instancias etiquetadas como *no factibles* por ambos algoritmos. Resultados tomados de las ejecuciones en el conjunto total de 549 instancias al fijar el límite de tiempo en 15 minutos.

Conclusiones

Dado que el *branch-and-cut* propuesto en el Capítulo 4 obtiene rápidamente cotas duales muy cercanas al óptimo y utiliza el resto del tiempo buscando factibilidad, hemos decidido desarrollar heurísticas primales con el fin de obtener soluciones factibles en una etapa temprana de la resolución. La heurística matemática desarrollada resuelve de forma iterativa un modelo ILP para una versión simplificada del RSA en la cual los caminos para gran parte de las demandas son fijados de antemano. De este modo el *solver* tiene la tarea de hallar la asignación de slots y los caminos para las demandas no fijadas. Dicha cantidad de demandas fijadas va decreciendo aumentando los grados de libertad con el objetivo de garantizar cada vez más la factibilidad del problema resultante pero con la consecuencia de complejizarlo en cada iteración. Asimismo, para evitar dedicar demasiado tiempo a instancias no factibles, se desarrolló un mecanismo para detectar este tipo de instancias con cierta probabilidad. La heurística implementada de este modo no sólo halla rápidamente una solución factible casi óptima, sino que es capaz de detectar instancias probablemente no factibles con gran precisión, a pesar de que los tiempos destinados a esta última parte del algoritmo siguen siendo más elevados que los requeridos por el *branch-and-cut* implementado por Cplex. Los experimentos en más de 550 instancias mostraron que los tiempos y la calidad de las soluciones mejoraron notablemente al agregar la heurística y sus variantes, resolviendo por optimalidad un cuarto más de instancias que la mejor configuración del *solver* Cplex, *i.e.*, utilizando *pre-solver*, heurísticas, paralelización y planos de corte. La diferencia del número de instancias resueltas sin garantía de optimalidad también estuvo en el mismo orden. A su vez, es interesante observar que la heurística propuesta presentó problemas de memoria en la resolución de la mitad de las instancias que Cplex. Por último, es notable que las heurísticas propuestas que utilizan el verificador de no factibilidad resuelven casi al instante las instancias con *slots-density* superior al 100 % –lo cual las hace directamente no factibles–, mientras que el resto de los

algoritmos dedican demasiado tiempo para resolverlas, dando *timeout* en la mayoría de ellas.

6
Conclusiones y trabajo futuro

El problema de ruteo y asignación de espectro surge como resultado de una de las soluciones más prometedoras para hacer frente a la enorme demanda de ancho de banda en las grandes redes de comunicación: la tecnología de *grilla flexible* (*flexgrid* del inglés) especificada en el estándar ITU-T G.694.1.

En esta tesis hemos presentado formalmente la versión estática de este problema y también hemos demostrado que el mismo pertenece a la clase $\mathcal{NP}$-difícil. Hemos propuesto diversos modelos de programación entera para la versión estática del RSA, hemos presentado y descrito una gran cantidad de desigualdades e igualdades válidas así como cortes optimales, y los hemos utilizado como planos de corte en un algoritmo *branch-and-cut*. Dicho algoritmo fue mejorado agregando heurísticas iniciales como *warm-start*, *i.e.*, las soluciones obtenidas con las heurísticas son pasadas al *solver* de programación entera para que no comience de cero la búsqueda del óptimo. En el presente capítulo resumimos las conclusiones a las que llegamos y algunas posibles líneas de trabajo futuro.

Las formulaciones Hemos propuesto doce modelos de programación entera para la versión estática del RSA, probando con diversas familias de variables y restricciones, obteniendo resultados mixtos. Para comparar estos modelos, los experimentos se realizaron utilizando OPL y el *solver* como una caja negra obteniendo así para cada modelo una idea aproximada del tamaño de instancia que es capaz de resolver sin técnicas de programación lineal entera más sofisticadas. Un resultado interesante es que, para la función objetivo seleccionada, todos los modelos propuestos superan a los extraídos de la literatura en todas las instancias utilizadas. Es razonable creer que esto es debido a la gran cantidad de variables y restricciones que estos modelos existentes utilizan. El modelo NLS, por ejemplo, presenta $\mathcal{O}(|D||E|+|D|\bar{s})$ variables y $\mathcal{O}(|D|^2|E|\bar{s}+|D||V|)$ restricciones, mientras que el modelo NL-CA emplea $\mathcal{O}(|D||E|\bar{s})$ variables en $\mathcal{O}(|D||E|\bar{s}|V|)$ restricciones. Sin embargo, ninguno de los modelos propuestos en el presente trabajo utiliza combinaciones tan grandes.

Estos resultados también sugieren que los modelos de la familia DR superan a las otras formulaciones propuestas, resolviendo casi el 90 % de las instancias utilizadas sin

problemas de memoria y obteniendo valores óptimos en las instancias más grandes. La primera conclusión que parece extraerse de este hecho es que los modelos generados por estas formulaciones son más pequeños que los generados por las demás, pero dado que los modelos DR utilizan tanto variables como restricciones que relacionan cada par de demandas existentes, mientras que el resto de los modelos suelen relacionar demandas con arcos y/o *slots*, los resultados pueden estar ligeramente sesgados por las instancias utilizadas. Proponemos como trabajo futuro verificar si este comportamiento se mantiene cuando el conjunto de demandas crece más rápido que el número de arcos y *slots*.

El algoritmo *branch-and-cut* y la heurística La segunda familia con mejor desempeño es la denominada DSL, en particular su formulación básica y, debido a su estructura y simetrías, hemos decidido continuar nuestro estudio sobre esta formulación. Hemos propuesto más de 65 familias de desigualdades e igualdades válidas y cortes optimales para esta formulación, y hemos demostrado la no implicación entre algunas de ellas. Luego hemos utilizado estas familias como planos de corte en un algoritmo de tipo *branch-and-cut*, el cual presenta diversas variantes dependiendo de los parámetros utilizados para el filtrado de los cortes y las distintas estrategias de selección de los mismos. Estas estrategias se compararon entre sí y la mejor de ellas resultó en una muy buena mejora con respecto a un algoritmo *branch-and-bound* y un algoritmo *branch-and-cut* con cortes genéricos. Con el procedimiento propuesto hemos mejorado notablemente los tiempos de ejecución y resuelto un tercio más de instancias que estos algoritmos.

A partir de los experimentos realizados con estas familias de desigualdades, al tomarlas aisladas, hemos observado que los cortes con mejor desempeño son aquellos que aprovechan las restricciones de contigüidad de los *slots*. Creemos que esto se debe principalmente al hecho de que las restricciones de flujo y no solapamiento son lo suficientemente ajustadas como para no dar demasiada libertad a una solución, recayendo en la propiedad de contigüidad la gran cantidad de combinaciones posibles para verificar. Por lo tanto, planos de corte que dejen afuera parte de estas combinaciones reducirían significativamente el espacio de búsqueda. Asimismo, debido a que varias de estas familias se deducen de un resultado que define una faceta en un problema muy pequeño, *i.e.*, el RSA con una sola demanda y un solo arco, es razonable esperar que, con algunas modificaciones a dichas familias, se llegue a desigualdades que definan facetas para el RSA general.

Dado que el *branch-and-cut* propuesto obtiene cotas duales muy cercanas al óptimo y utiliza el resto del tiempo buscando factibilidad, hemos decidido desarrollar heurísticas primales con el fin de obtener soluciones factibles en una etapa temprana de la resolución. La heurística matemática desarrollada resuelve de forma iterativa un modelo ILP y, no sólo halla rápidamente una solución factible casi óptima, sino que es capaz de detectar instancias probablemente no factibles con gran precisión. Los experimentos en más de 550 instancias mostraron que los tiempos y la calidad de las soluciones mejoraron notablemente al agregar dicha heurística y sus variantes, resolviendo por optimalidad un cuarto más de instancias que la mejor configuración del *solver* Cplex, *i.e.*, utilizando *pre-solver*, heurísticas, paralelización y planos de corte. La diferencia del número de instancias resueltas sin garantía de optimalidad también estuvo en el mismo orden. Asimismo, es interesante observar que la heurística propuesta presentó problemas de memoria en la resolución de la mitad de las instancias que Cplex.

El *benchmark* de instancias utilizadas Dado que a nuestro leal saber y entender

no existen instancias completas disponibles en la literatura, sino sólo algunas topologías y parámetros, y siendo una práctica muy común en el estudio del RSA y sus variaciones, el generar un *benchmark* para cada trabajo, hemos implementado un *script* generador de instancias basado en topologías y parámetros reales con el fin principal de realizar los experimentos pertinentes. Sin embargo, también creemos que sería bueno que estas instancias así como el *script* desarrollado, el cual está disponible en [16], sean el comienzo de un *benchmark* estandarizado para el RSA.

Para medir la dificultad de resolver una instancia dada hemos propuesto una nueva definición de densidad, que llamamos *slots-density*, la cual relaciona la cantidad total de *slots* requeridos por el conjunto de demandas con el total de *slots* disponibles en la red. Una conclusión interesante que surge del estudio de esta densidad es que en la mayoría de las instancias utilizadas en la literatura se observa una densidad menor al 20 % (muchas incluso menor al 10 %) y como el número de *slots* y las demandas generalmente se generan con valores tales que sea posible manejarlos por los modelos, éstas se vuelven muy fáciles de resolver (muchas de ellas hasta la optimalidad) para la heurística primal presentada en este trabajo.

Otro resultado interesante es que, dado que las instancias son generadas aleatoriamente por el *script* usando otra unidad de densidad, es decir, la denominada *demands-density*, éstas pueden tener una *slots-density* superior al 100 %, lo cual las hace directamente no factibles. Sin embargo, debido a que son tan densas, todos los algoritmos, tanto los propuestos –sin el *density-checker*– como los genéricos, requieren demasiado tiempo para resolverlas, devolviendo *time out* para la mayoría de ellas.

Finalmente, presentamos el *survey* más extenso hasta la fecha enumerando y clasificando los trabajos que tratan el RSA y sus variaciones más cercanas desde la óptica de la optimización combinatoria.

6.1. Trabajo futuro

El objetivo principal del presente trabajo fue analizar y explotar diversas propiedades, tanto intrínsecas al problema como a uno de los modelos propuestos, con el fin de sentar las bases para iniciar estudios poliedrales y ser capaces de resolver instancias reales en tiempos razonables. Si bien el estudio realizado cumplió con su principal objetivo, el mismo ha dejado un gran número de líneas de trabajo pendientes en cada una de las etapas. En esta sección intentamos resumirlas.

Estudiar las formulaciones Dada la naturaleza del problema y debido a las características de los modelos propuestos, es posible que muchas soluciones presuntamente diferentes no lo sean tanto, sino que una se pueda transformar en la otra mediante un renombre de variables. Esto se conoce como *simetría*. Por ejemplo, cualquier solución para el DSL-BF para una instancia para la cual los *slots* se numeraron en orden ascendente es una solución para la misma instancia numerando los *slots* en orden descendente, y viceversa. Lo mismo ocurre con una gran cantidad de puntos, enteros y fraccionarios, que no son solución. Estos puntos bien podrían unificarse para reducir el espacio de búsqueda. En lo que respecta al modelado, por lo tanto, una mejora podría ser generar formulaciones que eliminen las simetrías que presenta este problema; aunque esto probablemente resulte en modelos con familias exponenciales de variables que requieran técnicas de generación de columnas, o bien modelos con estructuras complicadas de analizar.

Para la comparación de las formulaciones se unificó la función objetivo seleccionando la que consideramos más simple. Sin embargo, dado que la función objetivo utilizada afecta el rendimiento en la práctica, se podrían realizar experimentos con diversas funciones objetivo. En particular, dado que una de las funciones objetivo más utilizadas para la etapa de planificación es la que busca minimizar el mayor *slot* asignado a las demandas, sería interesante comparar las formulaciones utilizando esta función y verificar si se mantienen o no los comportamientos. Quizás sea posible encontrar alguna relación en términos de desempeño entre el tipo de modelo y la función objetivo elegida.

Estudiar el poliedro asociado con la formulación Dada la complejidad del problema estudiado, y la escasez de bibliografía poliedral relacionada, en este trabajo hemos intentado dar unos primeros pasos con miras a comenzar un estudio poliedral. Ante la imposibilidad de proveer resultados de facetitud, se buscó relacionar entre sí la gran cantidad de familias propuestas, con el fin de determinar de algún modo razonable su eficacia. Un posible paso siguiente sería buscar generalizaciones que abarquen varias de estas familias.

Paralelamente hemos comenzado un estudio poliedral sobre un problema reducido, *i.e.*, asumiendo un único arco y una única demanda. Creemos que posiblemente convenga avanzar por ese camino e ir complejizando el problema cada vez más hasta llegar al RSA. Una vez caracterizada la dimensión se podría aspirar a demostrar resultados de facetitud tanto para las desigualdades propuestas como para las que surjan de todos estos estudios. Es esperable que las desigualdades que definan facetas resulten en mejores cortes para un algorimo de tipo *branch-and-cut*, por lo que eso implicaría también actualizar dicho algorimo.

Del estudio realizado sobre las topologías existentes hemos observado que casi todas son grafos planares o grafos tales que si se les extrae una o dos aristas son planares. Asimismo, como era de esperar, generalmente el tamaño de corte mínimo de estos grafos es relativamente alto, lo que los hace muy tolerantes a las fallas de la red. Un trabajo futuro interesante sería estudiar las particularidades de los cortes propuestos sobre familias de grafos bien definidas con características similares a las observadas.

Mejorar el algoritmo *branch-and-cut* Hemos propuesto algunas posibles modificaciones para algunos de los procedimientos presentados con el fin de separar las familias de desigualdades, generalmente mediante el uso de estructuras particulares. No implementamos algunas de estas mejoras debido al mal desempeño de los cortes asociados. Por tanto, en lo que respecta a los algoritmos de separación, dejamos como trabajo futuro estudiar en profundidad el motivo de estos malos comportamientos así como la implementación de las restantes mejoras. Dado que la mayoría de los cortes tales que al agregarlos no mejoran –incluso empeoran– el comportamiento del *branch-and-cut* pertenecen a familias exponenciales, quizás optimizar sus algoritmos de separación mejoraría su contribución.

También sería posible agrupar las instancias por sus características y utilizar estos resultados para mejorar el filtrado de cortes. Al estudiar con más detalle las instancias en las que los diferentes cortes se comportan mejor, puede ayudar a priorizar ciertas familias con –presumiblemente– un mejor desempeño. Quizás sea posible sacar conclusiones analizando si hay cortes que funcionan mejor en grafos más densos o en instancias con una mayor probabilidad de solapamiento. Hacer que la cantidad de cortes a agregar dependa tanto de la estructura relacionada con el corte como del nodo del árbol de búsqueda actual pareciera ser otro buen punto de partida.

Modificar la heurística Dado que el objetivo principal de la última parte de la tesis es estudiar si la adición de heurísticas iniciales al algoritmo tiene un impacto positivo en su rendimiento, se intentó implementar de forma sencilla tanto las heurísticas desarrolladas como sus ligeras variaciones. Esto conduce a una amplia variedad de posibles mejoras.

En primer lugar, hemos utilizado la misma formulación tanto para el *solver* esclavo como para el primario. Sin embargo, dado que el primero está mayoritariamente resolviendo el SA, quizás el *branch-and-cut* usado en ese *solver* podría modificarse agregando sólo los cortes más relacionados con este problema. Se podría incluso cambiar el modelo, quizás por uno de aquellos con dos variables diferentes para el ruteo y la asignación de espectro.

Si bien en la literatura no existen heurísticas matemáticas para el RSA o heurísticas utilizadas como *warm-start* de algoritmos exactos para este problema, sin embargo quizás algunas de las ideas propuestas en las heurísticas existentes podrían utilizarse para mejorar el desempeño de la presentada en este trabajo. Entre los enfoques utilizados, el que divide el RSA en los problemas de ruteo y de SA es el más estudiado. Dado que en cada iteración de la heurística mezclamos las demandas y, dado que en algunos pequeños experimentos que hemos realizado pudimos observar que ordenándolas según su volumen –de mayor a menor, *i.e.*, aplicando MSF [33]– los resultados empeoraban significativamente, quizás se podría utilizar la base de algunos principios de las otras heurísticas existentes, como FF [62], o VSA [3], para dar un orden más inteligente a dicho conjunto.

El parámetro que limita el tiempo de ejecución del *solver* secundario fue calibrado con relativamente pocos experimentos hasta lograr un valor que diera resultados razonables. Quizás amerite estudiar más profundamente la relación entre el tiempo consumido por el *solver* y ciertas propiedades de las instancias, como por ejemplo el tamaño o la densidad, a fin de ajustar mejor dicho parámetro. En este paso podrían incluso aplicarse técnicas de aprendizaje automático.

Debido a que hay muchas instancias intratables a causa del tamaño del modelo, la heurística podría modificarse para que, en lugar de establecer las rutas mínimas en el modelo original, intente generar un modelo reducido; quizás resolviendo primero el SA en las demandas fijas y luego eliminando variables y restricciones para tratar de acomodar el resto. Evidentemente esto permitiría tener soluciones factibles sin garantizar la optimalidad. Para esto último, sigue siendo mandatorio cargar el modelo completo en un paso posterior.

También podemos mejorar la heurística al encontrar, para cada demanda, la ruta más corta que haga que el solapamiento sea mínimo. Se puede hacer en un conjunto de *k-rutas-más-cortas* para cada demanda; y el solapamiento se puede tomar como el promedio o el máximo, por ejemplo. Resolver un modelo de ILP podría ser una buena opción para este paso.

Otros estudios y mejoras posibles Dado que al desactivar el *pre-solver* del *branch-and-cut* implementado por Cplex al usarlo como esclavo empeoran notablemente los tiempos y la calidad de las soluciones encontradas por las heurísticas, desarrollar un *pre-solver* dedicado podría resultar en una gran mejora. Se podría utilizar tanto para el *solver* esclavo como para el primario. Dada la pequeña cantidad de variables necesarias para determinar una ruta y un rango de *slots* para satisfacer cada demanda y la gran cantidad de variables que presenta la formulación, es razonable pensar que es posible encontrar diversas formas de reducir el modelo. Tener un conocimiento más profundo de las simetrías del problema y en particular del modelo estudiado podría traer muchos beneficios a este paso.

Para mejorar la medida de densidad utilizada en el presente trabajo sería interesante analizar los arcos donde hay solapamiento cuando cada demanda utiliza uno de sus caminos más cortos. Quizás mirando el máximo, el mínimo, la desviación estándar y otros parámetros, podría aparecer algún valor que se correlacione incluso mejor que la densidad de demandas que usamos aquí.

Asimismo, para conocer mejor la eficacia del *infeasibility check* de la heurística, sería interesante ejecutar en una computadora más potente las instancias marcadas como *probablemente no factible* que no hayan finalizado o bien por alcanzar el límite de tiempo o por problemas de memoria.

Bibliografía

[1] F. S. Abkenar, A. Ghaffarpour Rahbar y A. Ebrahimzadeh. «Best fit (BF): A new Spectrum Allocation mechanism in Elastic Optical Networks (EONs)». En: *2016 8th International Symposium on Telecommunications (IST)*. 2016, págs. 24-29.

[2] M. Aibin y K. Walkowiak. «Adaptive modulation and regenerator-aware dynamic routing algorithm in elastic optical networks». En: *2015 IEEE International Conference on Communications (ICC)*. 2015, págs. 5138-5143.

[3] M. Aibin y K. Walkowiak. «Simulated Annealing algorithm for optimization of elastic optical networks with unicast and anycast traffic». En: *16th International Conference on Transparent Optical Networks (ICTON)* (2014), págs. 1-4.

[4] S. Aleksić y A. Lovrić. «Power consumption of wired access network technologies». En: *2010 7th International Symposium on Communication Systems, Networks Digital Signal Processing (CSNDSP 2010)*. 2010, págs. 147-151.

[5] R. C. Almeida Jr. y col. «A Slot-priority Spectrum Assignment Algorithm for Elastic Optical Networks». En: *Journal of Microwaves, Optoelectronics and Electromagnetic Applications (JMOe)* 12 (ago. de 2013), págs. 181-192.

[6] E. Amaldi, C. Iuliano y R. Rizzi. «A better polynomial-time algorithm for finding minimum cycle bases in undirected graphs». En: *Graph and Optimization Meeting (GOM 2008)*. Association of European Operational Research Societies. Saint-Maximim La Sainte Baume, France, 2008.

[7] C. M. Araújo y col. «On solving the capacitated routing and spectrum allocation problem for flexgrid optical networks». En: *Computer Networks* 181 (2020), págs. 1-10. ISSN: 1389-1286.

[8] E. Archambault y col. «Routing and Spectrum Assignment in Elastic Filterless Optical Networks». En: *IEEE/ACM Transactions on Networking* 24.6 (dic. de 2016), págs. 3578-3592. ISSN: 1558-2566.

[9] K. D. R. Assis, A. Ferreira dos Santos y R. C. Almeida. «Optimization in spectrum-sliced optical networks». En: *Optical Metro Networks and Short-Haul Systems VI*. Ed. por Werner Weiershausen y col. Vol. 9008. International Society for Optics y Photonics. SPIE, 2014, págs. 102-109.

[10] K. D. R. Assis, A. Ferreira dos Santos e I. M. Queiroz. «Routing in EON networks under mixed static and dynamic traffic». En: *Optical Metro Networks and Short-Haul Systems IX*. Ed. por Atul K. Srivastava, Benjamin B. Dingel y Youichi Akasaka. Vol. 10129. International Society for Optics y Photonics. SPIE, 2017, págs. 44-49.

[11] J. Baliga y col. «Energy consumption in wired and wireless access networks». En: *IEEE Communications Magazine* 49.6 (2011), págs. 70-77.

[12] S. Behera y G. Das. «Dynamic Routing and Spectrum Allocation in Elastic Optical Networks with Minimal Disruption». En: *2020 National Conference on Communications (NCC)*. 2020, págs. 1-5.

[13] F. Berger, P. Gritzmann y S. De Vries. «Minimum cycle bases and their applications». En: *Algorithmics of large and complex networks*. Springer, 2009, págs. 34-49.

[14] F. Bertero, M. Bianchetti y J. Marenco. «Integer programming models for the routing and spectrum allocation problem». En: *TOP* 322.10 (jul. de 2018), págs. 891-921.

[15] D. Bertsimas y R. Weismantel. *Optimization over integers*. Belmont, MA: Dynamic Ideas, 2005.

[16] M. Bianchetti. *exactasmache/RSAinstances: RSA Instances*. Ver. v1.0. Oct. de 2020. URL: https://github.com/exactasmache/RSAinstances.

[17] M. Bianchetti y J. Marenco. «A Branch-and-cut algorithm for the Routing and Spectrum Allocation problem». En: *VIII Congreso de Matemática Aplicada, Computacional e Industrial. MACI 2021*. Centro de Posgrado Sergio Karakachoff, Universidad Nacional de La Plata. La Plata, Argentina, mayo de 2021.

[18] M. Bianchetti y J. Marenco. «A primal heuristic for the routing and spectrum allocation problem». En: *ALIO/EURO 2021-2022: Xth Joint ALIO/EURO International Conference on Applied Combinatorial Optimization*. Association of Latin-Iberoamerican Operational Research Societies (ALIO) y the Association of European Operational Research Societies (EURO). Viña del Mar, Chile, abr. de 2022.

[19] M. Bianchetti y J. Marenco. «Valid inequalities and a branch-and-cut algorithm for the routing and spectrum allocation problem». En: *Procedia Computer Science* 195 (2021). Proceedings of the XI Latin and American Algorithms, Graphs and Optimization Symposium., págs. 523-531. ISSN: 1877-0509.

[20] E. Birmelé y col. «Optimal Listing of Cycles and st-Paths in Undirected Graphs». En: *Proceedings of the 2013 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2013, págs. 1884-1896.

[21] A. Bley, O. Maurer e I. Ljubic. «Lagrangian decompositions for the two-level FTTx network design problem». En: *EURO Journal on Computational Optimization* 1.3 (2013), págs. 221-252. ISSN: 2192-4406.

[22] A. Cai y col. «Novel Node-Arc Model and Multiiteration Heuristics for Static Routing and Spectrum Assignment in Elastic Optical Networks». En: *Journal of Lightwave Technology* 31.21 (2013), págs. 3402-3413.

[23] J. H. L. Capucho y L. C. Resendo. «ILP model and effective genetic algorithm for routing and spectrum allocation in elastic optical networks». En: *2013 SBMO/IEEE MTT-S International Microwave Optoelectronics Conference (IMOC)*. 2013, págs. 1-5.

[24] A. Castro y col. «Dynamic routing and spectrum (re)allocation in future flexgrid optical networks». En: *Computer Networks* 56.12 (2012), págs. 2869-2883.

[25] A. Castro y col. «Path-based recovery in flexgrid optical networks». En: *2012 14th International Conference on Transparent Optical Networks (ICTON)*. 2012, págs. 1-4.

[26] B. C. Chatterjee, N. Sarma y E. Oki. «Routing and Spectrum Allocation in Elastic Optical Networks: A Tutorial». En: *IEEE Communications Surveys Tutorials* 17.3 (2015), págs. 1776-1800.

[27] X. Chen, Y. Zhong y A. Jukan. «Multipath routing in elastic optical networks with distance-adaptive modulation formats». En: *2013 IEEE International Conference on Communications (ICC)*. 2013, págs. 3915-3920.

[28] I. Chlamtac y A. Ganz. «Lightpath Communications: An Approach to High Bandwidth Optical WAN's». En: *IEEE Transactions on Communications* 40.7 (1992), págs. 1171-1182.

[29] K. Christodoulopoulos, I. Tomkos y E. Varvarigos. «Dynamic bandwidth allocation in flexible OFDM-based networks». En: *2011 Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference*. 2011, págs. 1-3.

[30] K. Christodoulopoulos, I. Tomkos y E. Varvarigos. «Spectrally/bitrate flexible optical network planning». En: *36th European Conference and Exhibition on Optical Communication*. 2010, págs. 1-3.

[31] K. Christodoulopoulos, I. Tomkos y E. A. Varvarigos. «Corrections to *Elastic Bandwidth Allocation in Flexible OFDM-Based Optical Networks*». En: *IEEE Journal of Lightwave Technology* 29.12 (2011), pág. 1899.

[32] K. Christodoulopoulos, I. Tomkos y E. A. Varvarigos. «Elastic bandwidth allocation in flexible OFDM-based optical networks». En: *IEEE Journal of Lightwave Technology* 29.9 (2011), págs. 1354-1366.

[33] K. Christodoulopoulos, I. Tomkos y E. A. Varvarigos. «Routing and Spectrum Allocation in OFDM-Based Optical Networks with Elastic Bandwidth Allocation». En: *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*. 2010, págs. 1-6.

[34] K. Christodoulopoulos y E. Varvarigos. «Static and dynamic spectrum allocation in flexi-grid optical networks». En: *2012 14th International Conference on Transparent Optical Networks (ICTON)*. 2012, págs. 1-5.

[35] R. Colares, H. Kerivin y A. Wagler. «An extended formulation for the Constraint Routing and Spectrum Assignment Problem in Elastic Optical Networks». En: *Proceedings of the Joint ALIO/EURO International Conference 2021-2022 on Applied Combinatorial Optimization*. Viña del Mar, Chile, 2022, págs. 5-10.

[36] H. Dao Thanh. «Contribution to Flexible Optical Network Design: Spectrum Assignment and Protection». Tesis doct. Télécom Bretagne ; Université de Bretagne Occidentale, mar. de 2014.

[37] M. N. Dharmaweera, R. Parthiban e Y. A. Şekercioğlu. «Toward a Power-Efficient Backbone Network: The State of Research». En: *IEEE Communications Surveys Tutorials* 17.1 (2015), págs. 198-227.

[38] L. P. Dias y col. «Channel-based RSA approach for virtualization and QoS-aware protection in optical networks». En: *ICC 2021-IEEE International Conference on Communications*. IEEE. 2021, págs. 1-6.

[39] M. A. Djojo y K. Karyono. «Computational load analysis of Dijkstra, A*, and Floyd-Warshall algorithms in mesh network». En: *2013 International Conference on Robotics, Biomimetics, Intelligent Computational Systems*. 2013, págs. 104-108.

[40] A. F. Dos Santos y col. «YBS heuristic for routing and spectrum allocation in flexible optical networks». En: *2017 IEEE 18th International Conference on High Performance Switching and Routing (HPSR)*. 2017, págs. 1-6.

[41] G. M. Durães y col. «The choice of the best among the shortest routes in transparent optical networks». En: *Computer Networks* 54.14 (2010), págs. 2400-2409. ISSN: 1389-1286.

[42] J. Enoch. «Nested Column Generation decomposition for solving the Routing and Spectrum Allocation problem in Elastic Optical Networks». En: *CoRR* abs/2001.00066 (2020). arXiv: 2001.00066.

[43] J. Enoch y B. Jaumard. «Towards Optimal and Scalable Solution for Routing and Spectrum Allocation». En: *Electron. Notes Discret. Math.* 64 (2018), págs. 335-344.

[44] RC Entringer y PJ Slater. «On the maximum number of cycles in a graph». En: *Ars Combin* 11 (1981), págs. 289-294.

[45] G. Feng, C. Douligeris y M. Klinkowski. «A heuristic for routing, modulation and spectrum allocation in spectrum sliced elastic optical path network». En: *2015 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*. 2015, págs. 111-115.

[46] S. Fujii y col. «On-demand spectrum and core allocation for multi-core fibers in elastic optical network». En: *2013 Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC)*. Mar. de 2013, págs. 1-3.

[47] S. Fujii y col. «On-Demand Spectrum and Core Allocation for Reducing Crosstalk in Multicore Fibers in Elastic Optical Networks». En: *J. Opt. Commun. Netw.* 6.12 (dic. de 2014), págs. 1059-1071.

[48] M. Garey y D. Johnson. *Computers and Intractability: A Guide to the Theory of NP- Completeness*. W. H. Freeman, 1979.

[49] O. Gerstel y col. «Elastic optical networking: a new dawn for the optical layer?» En: *IEEE Communications Magazine* 50.2 (feb. de 2012), págs. 12-20.

[50] L. Gong y col. «A Two-Population Based Evolutionary Approach for Optimizing Routing, Modulation and Spectrum Assignments (RMSA) in O-OFDM Networks». En: *IEEE Communications Letters* 16.9 (2012), págs. 1520-1523.

[51] R. Goścień y P. Lechowicz. «Column generation technique for optimization of survivable flex-grid SDM networks». En: *2017 9th International Workshop on Resilient Networks Design and Modeling (RNDM)*. 2017, págs. 1-7.

[52] M. Grötschel, L. Lovász y A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Algorithms and Combinatorics. Springer-Verlag, 1988.

[53] Y. Hadhbi, H. Kerivin y A. Wagler. «A novel integer linear programming model for routing and spectrum assignment in optical networks». En: *2019 Federated Conference on Computer Science and Information Systems (FedCSIS)*. 2019, págs. 127-134.

[54] D. T. Hai y K. M. Hoang. «An efficient genetic algorithm approach for solving routing and spectrum assignment problem». En: *2017 International Conference on Recent Advances in Signal Processing, Telecommunications Computing (SigTelCom)*. Los Angeles, CA, USA, 2017, págs. 187-192.

[55] I. Huang y B. Li. «A Genetic Algorithm Using Priority-Based Encoding for Routing and Spectrum Assignment in Elastic Optical Network». En: *2014 7th International Conference on Intelligent Computation Technology and Automation*. 2014, págs. 5-11.

[56] R. Hulsermann y col. «A set of typical transport network scenarios for network modelling». En: *ITG FACHBERICHT* 182 (2004), págs. 65-72.

[57] International Telecommunication Union - ITU-T. «Spectral grids for WDM applications: DWDM frequency grid». En: *Ser.* G.694.1 (feb. de 2012), págs. 1-16.

[58] B. Jaumard y M. Daryalal. «Scalable elastic optical path networking models». En: *2016 18th International Conference on Transparent Optical Networks (ICTON)*. 2016, págs. 1-4.

[59] P. N. Ji y col. «Optical layer traffic grooming in flexible optical WDM (FWDM) networks». En: *2011 37th European Conference and Exhibition on Optical Communication*. 2011, págs. 1-3.

[60] Q. Jin y col. «Study of dynamic Routing and Spectrum Assignment schemes in Bandwidth Flexible Optical networks». En: *2011 Asia Communications and Photonics Conference and Exhibition (ACP)*. 2011, págs. 1-6.

[61] M. Jinno y col. «Demonstration of novel spectrum-efficient elastic optical path network with per-channel variable capacity of 40 Gb/s to over 400 Gb/s». En: *2008 34th European Conference on Optical Communication*. 2008, págs. 1-2.

[62] M. Jinno y col. «Distance-adaptive spectrum resource allocation in spectrum-sliced elastic optical path network [Topics in Optical Communications]». En: *IEEE Communications Magazine* 48.8 (2010), págs. 138-145.

[63] M. Jinno y col. «Spectrum-efficient and scalable elastic optical path network: architecture, benefits, and enabling technologies». En: *IEEE Communications Magazine* 47.11 (2009), págs. 66-73.

[64] D. B. Johnson. «Finding All the Elementary Circuits of a Directed Graph». En: *SIAM Journal on Computing* 4.1 (1975), págs. 77-84.

[65] H. Kasahara y S. Narita. «PracticalMultiprocessor Scheduling Algorithmsfor EfficientParallel Processing». En: *IEEE Transactions on Computers* 33.11 (1984), págs. 1023-1029.

[66] H. Kerivin, D. Nace y T. Pham. «Design of Capacitated Survivable Networks With a Single Facility». En: *IEEE/ACM Transactions on Networking* 13.2 (2005), págs. 248-261.

[67] H. Kerivin y A. Wagler. «On Superperfection of Edge Intersection Graphs of Paths». En: *Graphs and Combinatorial Optimization: from Theory to Applications: CTW2020 Proceedings*. Ed. por Claudio Gentile, Giuseppe Stecca y Paolo Ventura. Cham: Springer International Publishing, 2021, págs. 79-91. ISBN: 978-3-030-63072-0.

[68] K Klinkowski, P. Lechowicz y K. Walkowiak. «Survey of resource allocation schemes and algorithms in spectrally-spatially flexible optical networking». En: *Optical Switching and Networking* 27 (2018), págs. 58-78. ISSN: 1573-4277.

[69] M. Klinkowski. «A Genetic Algorithm for Solving RSA Problem in Elastic Optical Networks with Dedicated Path Protection». En: *International Joint Conference CISIS'12-ICEUTE'12-SOCO'12 Special Sessions*. Ed. por A. Herrero y col. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, págs. 167-176.

[70] M. Klinkowski y D. Careglio. «A Routing and Spectrum Assignment Problem in Optical OFDM Networks». En: *1st European Teletraffic Seminar (ETS)* (2011).

[71] M. Klinkowski y Walkowiak K. «A Simulated Annealing Heuristic for a Branch and Price-Based Routing and Spectrum Allocation Algorithm in Elastic Optical Networks». En: *Intelligent Data Engineering and Automated Learning - IDEAL 2015. Lecture Notes in Computer Science. Springer, Cham.* 9375 (2015).

[72] M. Klinkowski y K. Walkowiak. «Routing and spectrum assignment in spectrum sliced elastic optical path network». En: *IEEE Communications Letters* 15.8 (2011), págs. 884-886.

[73] M. Klinkowski y col. «Solving large instances of the RSA problem in flexgrid elastic optical networks». En: *IEEE/OSA Journal of Optical Communications and Networking* 8.5 (2016), págs. 320-330.

[74] M. Klinkowski y col. «Spectrum allocation problem in elastic optical networks - a branch-and-price approach». En: *2015 17th International Conference on Transparent Optical Networks (ICTON)*. Jul. de 2015, págs. 1-5.

[75] M. Klinkowski y col. «Valid inequalities for the routing and spectrum allocation problem in elastic optical networks». En: *2014 16th International Conference on Transparent Optical Networks (ICTON)*. 2014, págs. 1-5.

[76] S. Knight y col. «The Internet Topology Zoo». En: *IEEE Journal on Selected Areas in Communications* 29.9 (oct. de 2011), págs. 1765-1775. ISSN: 1558-0008.

[77] R. Leepila, E. Oki y N. Kishi. «A power-efficient design scheme for survivable networks with partial bandwidth path protection». En: *IEICE Communications Express* 3.1 (2014), págs. 1-6.

[78] F. Lezama y col. «Solving routing and spectrum allocation problems in flexgrid optical networks using pre-computing strategies». En: *Photonic Network Communications* 41.1 (2021), págs. 17-35.

[79] L. Li y H. Li. «Performance analysis of novel routing and spectrum allocation algorithm in elastic optical networks». En: *Optik* 212 (2020), pág. 164688. ISSN: 0030-4026.

[80] X. Li y col. «Farsighted Spectrum Resource Assignment Method for Advance Reservation Requests in Elastic Optical Networks». En: *IEEE Access* 7 (nov. de 2019), págs. 167836-167846.

[81] Y. Li, N. Hua y X. Zheng. «A capacity analysis for space division multiplexing optical networks with MIMO equalization». En: *2017 Optical Fiber Communications Conference and Exhibition (OFC)*. Optical Society of America, mar. de 2017, págs. 1-3.

[82] Y. Li, N. Hua y X. Zheng. «Routing, wavelength and core allocation planning for multi-core fiber networks with MIMO-based crosstalk suppression». En: *2015 Opto-Electronics and Communications Conference (OECC)*. Jun. de 2015, págs. 1-3.

[83] Y. Li y col. «Shared backup path protection in multi-core fiber networks with MIMO-based crosstalk suppression». En: *2016 Optical Fiber Communications Conference and Exhibition (OFC)*. 2016, págs. 1-3.

[84] V. Lohani, A. Sharma e Y. N. Singh. *Dynamic Routing and Spectrum Assignment based on the Availability of Consecutive Sub-channels in Flexi-grid Optical Networks*. 2021. arXiv: 2105.07560 [cs.NI].

[85] J. López y col. «Traffic and power-aware protection scheme in Elastic Optical Networks». En: *2012 15th International Telecommunications Network Strategy and Planning Symposium (NETWORKS)*. 2012, págs. 1-6.

[86] J. Marenco y A. Wagler. «Chromatic scheduling polytopes coming from the bandwidth allocation problem in point-to-multipoint radio access systems». En: *Annals of Operations Research* 150.1 (2007), págs. 159-175.

[87] J. Marenco y A. Wagler. «Cycle-based facets of chromatic scheduling polytopes». En: *Discrete Optimization* 6.1 (2009), págs. 51-63.

[88] J. Marenco y A. Wagler. «Facets of chromatic scheduling polytopes based on covering cliques». En: *Discrete Optimization* 6.1 (2009), págs. 64-78.

[89] J. Marenco y A. Wagler. «On the combinatorial structure of chromatic scheduling polytopes». En: *Discrete Applied Mathematics* 154.13 (2005), págs. 1865-1876.

[90] F. Margot. «Symmetry in Integer Linear Programming». En: *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Ed. por M. Jünger y col. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, págs. 647-686.

[91] G. Z. Marković. «Routing and spectrum allocation in elastic optical networks using bee colony optimization». En: *Photonic Network Communications* 34.3 (2017), págs. 356-374.

[92] E. Q. V. Martins y M. M. B. Pascoal. «A new implementation of Yen's ranking loopless paths algorithm». En: *Quarterly Journal of the Belgian, French and Italian Operations Research Societies* 1.2 (2003), págs. 121-133.

[93] N. S. Monteiro y col. «Provision of adaptive guard band in elastic optical networks». En: *Journal of Internet Services and Applications* 11.1 (2020), pág. 5.

[94] A. Muhammad y col. «Filterless Networks Based on Optical White Boxes and SDM». En: *ECOC 2016; 42nd European Conference on Optical Communication*. 2016, págs. 1-3.

[95] A. Muhammad y col. «Flexible and synthetic SDM networks with multi-core-fibers implemented by programmable ROADMs». En: *2014 The European Conference on Optical Communication (ECOC)*. 2014, págs. 1-3.

[96] A. Muhammad y col. «Routing, spectrum and core allocation in flexgrid SDM networks with multi-core fibers». En: *2014 International Conference on Optical Network Design and Modeling*. 2014, págs. 192-197.

[97] K. K. Munasinghe y col. «Joint Minimization of Spectrum and Power in Impairment-Aware Elastic Optical Networks». En: *IEEE Access* 9 (2021), págs. 43349-43363.

[98] F. Musumeci y col. «Energy efficiency in reliable optical core networks». En: *2015 IEEE Online Conference on Green Communications (OnlineGreenComm)*. 2015, págs. 1-6.

[99] G. Nemhauser y L. Wolsey. *Integer Programming and Combinatorial Optimization*. Wiley Series in Discrete Mathematics and Optimization. Wiley-Interscience, 1998.

[100] I. Olszewski. «Routing and Spectrum Assignment in Spectrum Flexible Transparent Optical Networks». En: *Image Processing and Communications Challenges 5*. Ed. por Ryszard S. Choras. Heidelberg: Springer International Publishing, 2014, págs. 407-417.

[101] S. Paira y col. «A novel fragmentation-aware and energy-efficient multipath routing and spectrum allocation for prioritized traffic in protected EONs». En: *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. 2020, págs. 1-6.

[102] A. N. Patel y col. «A naturally-inspired algorithm for Routing, Wavelength assignment, and Spectrum Allocation in flexible grid WDM networks». En: *2012 IEEE Globecom Workshops*. 2012, págs. 340-345.

[103] A. N. Patel y col. «Dynamic routing, wavelength assignment, and spectrum allocation in transparent flexible optical WDM networks». En: *Optical Metro Networks and Short-Haul Systems III*. Ed. por Werner Weiershausen y col. Vol. 7959. International Society for Optics y Photonics. SPIE, 2011, págs. 170-177.

[104] A. N. Patel y col. «Routing, wavelength assignment, and spectrum allocation algorithms in transparent flexible optical WDM networks». English. En: *Optical Switching and Networking* 9.3 (2012), págs. 191-204.

[105] A. N. Patel y col. «Routing, Wavelength Assignment, and Spectrum Allocation in Transparent Flexible Optical WDM (FWDM) Networks». En: *Integrated Photonics Research, Silicon and Nanophotonics and Photonics in Switching*. Optical Society of America, 2010.

[106] A. N. Patel y col. «Survivable transparent Flexible optical WDM (FWDM) networks». En: *2011 Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference*. 2011, págs. 1-3.

[107] B. Patel y col. «On Efficient Candidate Path Selection for Dynamic Routing in Elastic Optical Networks». En: *2020 11th IEEE Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*. 2020, págs. 0889-0894.

[108] A. Paul. «An optimal and a heuristic approach to solve the route and spectrum allocation problem in OFDM networks». Windsor, ON, Canada: School of Computer Science, University of Windsor, 2014.

[109] A. Paul y col. «A Fast Approach to Solve the Route and Spectrum Allocation Problem in OFDM Networks». En: *Proceedings of the 2015 International Conference on Distributed Computing and Networking*. ICDCN '15. Goa, India: Association for Computing Machinery, 2015.

[110] A. Paul y col. «An optimal and a heuristic algorithm for solving the route and spectrum allocation problem in OFDM networks». En: *Photonic Network Communications* 30.2 (2015), págs. 280-289.

[111] J. Perelló y col. «Assessment of Flex-grid/SDM backbone networks under inter-core XT-limited transmission reach». En: *2015 International Conference on Photonics in Switching (PS)*. 2015, págs. 190-192.

[112] J. Perelló y col. «Flex-grid/SDM backbone network design with inter-core XT-limited transmission reach». En: *IEEE/OSA Journal of Optical Communications and Networking* 8.8 (2016), págs. 540-552.

[113] C. D. Pérez López y col. «Semi-dynamic Routing and Spectrum Assignment with variable bandwidth in Elastic Optical Networks. Bee-inspired Algorithms approach». En: *2021 XLVII Latin American Computing Conference (CLEI)*. 2021, págs. 1-10.

[114] A. Perko. «Implementation of algorithms for K shortest loopless paths». En: *Networks* 16.2 (1986), págs. 149-160.

[115] G. Retvari, J. J. Biro y T. Cinkler. «On Shortest Path Representation». En: *IEEE/ACM Transactions on Networking* 15.6 (2007), págs. 1293-1306.

[116] J. M. Rivas-Moscoso y col. «Cost Benefit Quantification of SDM Network Implementations based on Spatially Integrated Network Elements». En: *ECOC 2016; 42nd European Conference on Optical Communication*. 2016, págs. 1-3.

[117] C. Rottondi y col. «Optimal Resource Allocation in Distance-Adaptive Few-Modes Backbone Networks with Flexible Grid». En: *Asia Communications and Photonics Conference 2015*. Optical Society of America, 2015, AS4H.2.

[118] C. Rottondi y col. «Routing, modulation format, baud rate and spectrum allocation in optical metro rings with flexible grid and few-mode transmission». En: *Journal of Lightwave Technology* 35.1 (2016), págs. 61-70.

[119] L. Ruan y N. Xiao. «Survivable multipath routing and spectrum allocation in OFDM-based flexible optical networks». En: *IEEE/OSA Journal of Optical Communications and Networking* 5.3 (2013), págs. 172-182.

[120] L. Ruan e Y. Zheng. «Dynamic survivable multipath routing and spectrum allocation in OFDM-based flexible optical networks». En: *IEEE/OSA Journal of Optical Communications and Networking* 6.1 (2014), págs. 77-85.

[121] M. Ruiz y col. «A column generation approach for large-scale RSA-based network planning». En: *2013 15th International Conference on Transparent Optical Networks (ICTON)*. 2013, págs. 1-4.

[122] M. Ruiz y col. «Column generation algorithm for RSA problems in flexgrid optical networks». En: *Photonic Network Communications* 26.2 (dic. de 2013), págs. 53-64. ISSN: 1572-8188.

[123] R. Rumipamba-Zambrano y col. «Resource Allocation in WDM vs. Flex-Grid Networks: Use Case in CEDIA Optical Backbone Network». En: *Information and Communication Technologies of Ecuador (TIC.EC)*. Advances in Intelligent Systems and Computing. 2019, págs. 18-33.

[124] A. F. Santos, R. C. Almeida y K. D. R. Assis. «Yen-BSR: A New Approach for the Choice of Routes in WDM Networks». En: *Journal of Optical Communications* 35.4 (2014), págs. 293-296.

[125] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.

[126] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.

[127] X. Shao y col. «Shared-path protection in OFDM-based optical networks with elastic bandwidth allocation». En: *OFC/NFOEC*. Mar. de 2012, págs. 1-3.

[128] B. Shariati y col. «Evaluation of the impact of different SDM switching strategies in a network planning scenario». En: *2016 Optical Fiber Communications Conference and Exhibition (OFC)*. 2016, págs. 1-3.

[129] B. Shariati y col. «Evaluation of the impact of spatial and spectral granularities on the performance of spatial superchannel switching schemes». En: *2016 18th International Conference on Transparent Optical Networks (ICTON)*. 2016, págs. 1-4.

[130] G. Shen, H. Guo y S.K. Bose. «Survivable elastic optical networks: survey and perspectives». En: *Photonic Network Communications* 31 (2013), págs. 71-87.

[131] S. Shirazipourazad, Z. Derakhshandeh y A. Sen. «Analysis of on-line routing and spectrum allocation in spectrum-sliced optical networks». En: *2013 IEEE International Conference on Communications (ICC)*. 2013, págs. 3899-3903.

[132] S. Shirazipourazad y col. «On routing and spectrum allocation in spectrum-sliced optical networks». En: *Proceedings IEEE INFOCOM* (abr. de 2013), págs. 385-389.

[133] Jon Sjogren. «Cycles and spanning trees». En: *Mathematical and Computer Modelling - MATH COMPUT MODELLING* 15 (dic. de 1991), págs. 87-102.

[134] J. Sócrates-Dantas y col. «Challenges and requirements of a control plane for elastic optical networks». En: *Computer Networks* 72 (2014), págs. 156-171.

[135] S. Sugihara y col. «Routing and spectrum allocation method for immediate reservation and advance reservation requests in elastic optical networks». En: *2015 International Conference on Photonics in Switching (PS)*. 2015, págs. 178-180.

[136] I. Szcześniak, A. Jajszczyk y B. Woźna-Szcześniak. «Generic Dijkstra for optical networks». En: *Journal of Optical Communications and Networking* 11.11 (sep. de 2019), pág. 568. ISSN: 1943-0639.

[137] I. Szcześniak y B. Woźna-Szcześniak. «Adapted and constrained Dijkstra for elastic optical networks». En: *2016 International Conference on Optical Network Design and Modeling (ONDM)*. 2016, págs. 1-6.

[138] S. Talebi y col. «Spectrum assignment in optical networks: A multiprocessor scheduling perspective». En: *IEEE/OSA Journal of Optical Communications and Networking* 6.8 (2014), págs. 754-763.

[139] S. Talebi y col. «Spectrum assignment in rings with shortest-path routing: Complexity and approximation algorithms». En: *2015 International Conference on Computing, Networking and Communications (ICNC)*. 2015, págs. 642-647.

[140] S. Talebi y col. «Spectrum management techniques for elastic optical nerworks: A survey». En: *Optical Switching and Networking* 13 (2014), págs. 34-48.

[141] R. Tarjan. «Enumeration of the Elementary Circuits of a Directed Graph». En: *SIAM Journal on Computing* 2.3 (1973), págs. 211-216.

[142] M. Tornatore y col. «On the complexity of routing and spectrum assignment in flexible-grid ring networks [Invited]». En: *Journal of Optical Communications and Networking* 7.2 (2015), págs. 256-267.

[143] R. S. Tucker y col. «Energy consumption in IP networks». En: *2008 34th European Conference on Optical Communication*. 2008, pág. 1.

[144] E. A. Varvarigos y K. Christodoulopoulos. «Algorithmic Aspects in Planning Fixed and Flexible Optical Networks With Emphasis on Linear Optimization and Heuristic Techniques». En: *Journal of lightwave technology* 32.4 (feb. de 2014), págs. 681-693.

[145] E. A. Varvarigos y K. Christodoulopoulos. «Algorithmic challenges in flexible optical networks». En: *2014 International Conference on Computing, Networking and Communications (ICNC)*. 2014, págs. 236-241.

[146] L. Velasco y col. «Modeling the routing and spectrum allocation problem for flexgrid optical networks». En: *Photonic Network Communications* 24 (2012), págs. 177-186.

[147] Luis Velasco y col. «Solving Routing and Spectrum Allocation Related Optimization Problems: From Off-Line to In-Operation Flexgrid Network Planning». En: *Journal of Lightwave Technology* 32.16 (ago. de 2014), págs. 2780-2795. ISSN: 1558-2213.

[148] K. Walkowiak, R. Goścień y M. Klinkowski. «On Minimization of the Spectrum Usage in Elastic Optical Networks with Joint Unicast and Anycast Traffic». En: *Asia Communications and Photonics Conference 2013*. Optical Society of America, 2013, AF4G.1.

[149] K. Walkowiak y M. Klinkowski. «Joint anycast and unicast routing for elastic optical networks: Modeling and optimization». En: *2013 IEEE International Conference on Communications (ICC)*. Jun. de 2013, págs. 3909-3914.

[150] K. Walkowiak y col. «ILP modeling of flexgrid SDM optical networks». En: *2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks)*. 2016, págs. 121-126.

[151] K. Walkowiak y col. «Optical networks for cost-efficient and scalable provisioning of big data traffic». En: *International Journal of Parallel, Emergent and Distributed Systems* 30.1 (2015), págs. 15-28.

[152] K. Walkowiak y col. «Routing and spectrum allocation algorithms for elastic optical networks with dedicated path protection». En: *Optical Switching and Networking* 13 (2014), págs. 63-75.

[153] Krzysztof Walkowiak. *Modeling and optimization of cloud-ready and content-oriented networks*. Vol. 56. Springer, 2016.

[154] X. Wan, N. Hua y X. Zheng. «Dynamic routing and spectrum assignment in spectrum-flexible transparent optical networks». En: *IEEE/OSA Journal of Optical Communications and Networking* 4.8 (2012), págs. 603-613.

[155] X. Wan y col. «Dynamic routing and spectrum assignment in flexible optical path networks». En: *2011 Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference*. 2011, págs. 1-3.

[156] N. Wang y col. «Scheduling Large Data Flows in Elastic Optical Inter-Datacenter Networks». En: *2015 IEEE Global Communications Conference (GLOBECOM)*. 2015, págs. 1-6.

[157] Y. Wang, X. Cao y Q. Hu. «Routing and Spectrum Allocation in Spectrum-Sliced Elastic Optical Path Networks». En: *2011 IEEE International Conference on Communications (ICC)*. 2011, págs. 1-5.

[158] Y. Wang, X. Cao e Y. Pan. «A study of the routing and spectrum allocation in spectrum-sliced Elastic Optical Path networks». En: *2011 Proceedings IEEE INFOCOM* (abr. de 2011), págs. 1503-1511.

[159] Y. Wang y col. «Towards elastic and fine-granular bandwidth allocation in spectrum-sliced optical networks». En: *IEEE Journal of Optical Communications and Networking* 4.11 (2012), págs. 906-917.

[160] G. Wellbrock y T. J. Xia. «The road to 100g deployment [Commentary]». En: *IEEE Communications Magazine* 48.3 (mar. de 2010), S14-S18.

[161] H. Wu y col. «Spectrum Management in Elastic Optical Networks: Perspectives of Topology, Traffic and Routing». En: *2019 IFIP Networking Conference (IFIP Networking)*. 2019, págs. 1-9.

[162] Q. Wu, J. Wang y M. Shigeno. «A novel channel-based model for the problem of routing, space, and spectrum assignment». En: *Optical Switching and Networking* 43 (2022), pág. 100636. ISSN: 1573-4277.

[163] H. Xuan y col. «Grey Wolf Algorithm and Multi-Objective Model for the Manycast RSA Problem in EONs». En: *Information* 10.12 (2019), págs. 1-16.

[164] H. Xuan y col. «New optimization model for routing and spectrum assignment with nodes insecurity». En: *Optics Communications* 389 (2017), págs. 42-50. ISSN: 0030-4018.

[165] H. Xuan y col. «Routing, spectrum and core assignment for multi-domain elastic optical networks with multi-core fibers». En: *Optical Fiber Technology* 59 (2020), págs. 1-10. ISSN: 1068-5200.

[166] L. Yang y col. «Polynomial-time adaptive routing algorithm based on spectrum scan in dynamic flexible optical networks». En: *China Communications* 10.4 (2013), págs. 49-58.

[167] M. Yang y col. «Hierarchical Routing and Resource Assignment in Spatial Channel Networks (SCNs): Oriented Toward the Massive SDM Era». En: *Journal of Lightwave Technology* 39.5 (mar. de 2021), págs. 1255-1270. ISSN: 1558-2213.

[168] J. Y. Yen. En: *Management Science* 17.11 (1971), págs. 712-716.

[169] J. Y. Yen. «An Algorithm for finding shortest routes from all source nodes to a given destination in general networks.» En: *Quarterly of Applied Mathematics* 27.4 (1970), págs. 526-530.

[170] Y. Yin y col. «Spectral and spatial 2D fragmentation-aware routing and spectrum assignment algorithms in elastic optical networks [invited]». En: *IEEE/OSA Journal of Optical Communications and Networking* 5.10 (2013), A100-A106.

[171] G. Zhang y col. «A Survey on OFDM-Based Elastic Core Optical Networking». En: *IEEE Communications Surveys Tutorials* 15.1 (2013), págs. 65-87.

[172] J. Zhao, H. Wymeersch y E. Agrell. «Nonlinear Impairment-Aware Static Resource Allocation in Elastic Optical Networks». En: *J. Lightwave Technol.* 33.22 (nov. de 2015), págs. 4554-4564.

[173] X. Zhou y col. «Dynamic RMSA in elastic optical networks with an adaptive genetic algorithm». En: *2012 IEEE Global Communications Conference (GLOBECOM)*. 2012, págs. 2912-2917.

[174] M. Zhu y col. «Energy-Aware Virtual Optical Network Embedding in Sliceable-Transponder-Enabled Elastic Optical Networks». En: *IEEE Access* 7 (ene. de 2019), págs. 41897-41912.

[175] R. Zhu y col. «Dynamic time and spectrum fragmentation-aware service provisioning in elastic optical networks with multi-path routing». En: *Optical Fiber Technology* 32 (2016), págs. 13-22. ISSN: 1068-5200.

[176] M. Zötkiewicz y col. «Optimization models for flexgrid elastic optical networks». En: *2013 15th International Conference on Transparent Optical Networks (ICTON)* (2013), págs. 1-4.

7

Apéndice: Clasificación del estado del arte

Según nuestro leal saber y entender, hasta la fecha existen pocos *surveys* relacionados con el RSA, y ninguno desde la perspectiva de la optimización combinatoria. Un *survey* sobre tecnologías de redes y transmisión óptica de alta velocidad basadas en la multiplexación por división de frecuencia ortogonal (OFDM) –con un enfoque específico en la tecnología OFDM– fue presentado por G. Zhang et al. en febrero de 2012 (y publicado en 2013) [171]. Otro estudio sobre la investigación realizada para definir arquitecturas de planos de control de redes ópticas elásticas se presentó en 2014 [134] por Sócrates et al. En 2015 [26], Chatterjee et al. escribieron un *survey* acerca del RSA desde una perspectiva telecomunicacional. En enero de 2018 [68], junto con la definición de los posibles escenarios de SDM, Klinkowski et al. estudiaron los algoritmos propuestos para la asignación de recursos y los clasificaron de acuerdo a diversos criterios, enfocándose mayormente en el RSSA. Recientemente, en 2020 [7], Araújo et al. presentaron un *survey* reducido sobre el RSA clasificando alrededor de 36 trabajos según el método de optimización propuesto, *i.e.*, si formulan un modelo ILP o utilizan heurísticas, y luego subclasificaron los 27 pertenecientes a la primera categoría, según el tipo de modelo.

En este apéndice pretendemos completar y continuar estos dos últimos *surveys*, mediante la recopilación de más de 110 publicaciones, no sólo con el fin de brindar una revisión general, sino haciendo foco principalmente en los artículos más íntimamente relacionados con nuestro estudio. Consideramos una variedad de criterios de clasificación, separando en particular los artículos que proponen formulaciones ILP y estudiando sus funciones objetivo, sus variables y sus técnicas de resolución, *i.e.*, *branch-and-cuts* con desigualdades válidas *custom*, generación de columnas, heurísticas primales; también recopilamos una gran cantidad de artículos que hacen uso de caminos mínimos, estrategias de ordenamiento del conjunto de demandas y aquellos que proponen alguna cota. También intentamos estudiar las instancias utilizadas, con sus tamaños y topologías. En los casos en los cuales tuvimos dudas sobre cómo clasificar un trabajo con respecto a alguno de los criterios, por ejemplo, si la información requerida no se proporciona en el artículo o no está claramente establecida, entonces dicho trabajo no se incluyó en la clasificación específica.

Problema		Dinámica	Estática
C-RSA	Constrained RSA		[7][53][137]
IA-RMSA	Impairment-Aware RMSA	[34][172]	[34][97]
M-RSA	Manycast RSA	[163]	
RMBSA	Routing, Modulation format, Baud-rate, Mode and Spectrum Allocation		[118][142]
RMSA	Routing, Modulation and Spectrum Allocation	[24][29][62][79] [100][107][136] [145][154][173] [175]	[31][32][36][45][80][142][144] [175]
RMSA-DP	RMSA with Dedicated Path protection		[36]
RMSA-G	RMSA with Grooming		[142]
RMSA-IN	RMSA with Insecure Nodes		[164]
RMSA-R	RMSA with Regeneration		[142]
RMSA-RG	RMSA with Regeneration and Grooming		[142]
RMSA-SP	RMSA with Shared Path protection		[36]
RMSCA	Routing, Modulation format, Spectrum, and Core Allocation		[95][112]
RMSCA-MD	RMSCA in Multi Domain networks		[165]
RMSCA-PF	Programmable F-RMSCA		[94]
RMSCA-F	Filterless RMSCA		[8]
RMSSA	Routing, Modulation format, Spatial resources and Spectrum Allocation		[150]
RMSSA-DP	RMSSA with Dedicated Path protection		[51]
RSA	Routing and Spectrum Assigment	[9][12][23][40] [41][55][60][84] [131][136][147] [155]	[14][19][22][30][33][36][42][43] [54][58][67][70][71][72] [73][74][75][78][91][108][121] [142][146][147][157][158][159] [161][170]
RSA-DP	RSA with Dedicated Path protection		[36]
RSA-DP-SC	RSA-DP using the Same Channel		[69]
RSA-G	RSA with Grooming		[142]
RSA/JAU	Joint Anycast and Unicast RSA		[3][148][149]
RSA-MP	RSA with Multi Path protection	[101]	
RSA-R	RSA with Regeneration		[142]
RSA-RG	RSA with Regeneration and Grooming		[142]
RSA-SP	RSA with Shared Path protection	[62]	[36]
RSCA	Routing, Spectrum and Core Allocation		[47][96]
RSCSA	Routing, Spatial Channel, and Spectrum Allocation		[167]
RSSA	Routing, Spatial resources and Spectrum Allocation		[68][162]
RWA	Routing and Wavelength Allocation	[136]	[102][144]
RWA-SP	RWA with Shared Path protection		[83]
RWCA	Routing, wavelength, and core allocation		[82]
RWSA	Routing, Wavelength Allocation, and Spectrum Allocation	[103]	[102][104][105]
SA	Spectrum Allocation		[3][139]
SCRSA	Spectrum Constrained RSA		[132]

Tab. 7.1: Referencias discriminadas por los problemas y variaciones estudiadas.

El listado de publicaciones más relevantes ordenadas cronológicamente con un breve resumen de cada una (destacando los datos más relevantes para nuestro trabajo) se puede consultar en el Apéndice 8.

Aunque el objetivo de este trabajo es comprender más profundamente el RSA en su versión estática en una red con fibras simples unidireccionales asimétricas (*i.e.*, interpretable como un grafo dirigido) y asumiendo la misma modulación para todas las demandas, hemos recopilado información sobre más de 35 variaciones del problema, que van desde el RWA hasta versiones más específicas, como el *programmable filterless RMSCA*. La Tabla 7.1 clasifica una gran cantidad de referencias según la versión estudiada de cada problema. Aparte del tráfico dinámico y estático, en la literatura se encuentra una tercera posibilidad denominada tráfico *semi-estático*, la cual es estudiada por los trabajos [1] y [10]. El primero analiza dicha variación sobre el RSA y el SA, mientras que el segundo sólo sobre el RSA. La Tabla 7.2 clasifica los trabajos separando aquellos en donde explícitamente se aclara si el grafo es interpretado como dirigido o no dirigido, y los que asumen bandas de guarda fijas.

Detalles del problema	Referencias
Bandas de guarda fijas	[7][8][23][29][30][31][32][33][34][43][55][58][62][70][78][84][100][107] [108][118][131][142][146][148][149][150][157][158][159][161][163]
Grafo dirigido	[14][19][30][31][32][33][36][70][72][96][107][108][137][144][157][161]
Grafo no dirigido	[22][42][53][58][67][73][82][83][163][165][172]

Tab. 7.2: Referencias discriminadas según algunos detalles del problema estudiado.

Espectro máximo	Número de *slots* requeridos en la red para satisfacer todas las demandas (también denominado *nivel de congestión* en [36]).
Intervalo de espectro mínimo	Diferencia entre el mayor y el menor *slot* utilizado en toda la red.
Espectro total	Número total de *slots* asignados en todos los enlaces de la red.
Espectro medio	Promedio del espectro utilizado por enlace por fibra.
Tráfico aceptado	Máximo número de demandas, de un conjunto dado, que pueden ser satisfechas por la red.
Costo de la red	Cantidad de transceptores de canales espaciales/espectrales, fibras utilizadas, nodos ópticos *express* o *add/drop*, amplificadores SDM o módulos de *switching*; potencia consumida, entre otros.
Longitud de las rutas	Suma de todas las longitudes de los caminos. Podría verse como un caso particular de costo de la red.
Tasa de bloqueo	Hace referencia a la *probabilidad de bloqueo de una conexión*, definida como la relación entre el número de rechazos y la cantidad total de solicitudes de conexión de la red, o a la *probabilidad de bloqueo de ancho de banda*, la cual típicamente expresa una relación entre la cantidad rechazada y el volumen de demanda aceptado (*i.e.*, definida en términos del ancho de banda o la tasa de bits).
Fragmentación	Cuando las conexiones se establecen dinámicamente, el espectro puede fragmentarse y obstaculizar el establecimiento de conexiones futuras. Este objetivo es difícil de medir y la mayoría de las veces se termina evaluando la variación de la tasa de bloqueo.

Tab. 7.3: Diferentes objetivos que se pueden seleccionar al resolver el RSA o sus variaciones.

La Tabla 7.4 separa los algoritmos propuestos de acuerdo al objetivo que buscan minimizar o maximizar. Aquellos que pudimos encontrar son explicados en la Tabla 7.3. Consideramos que minimizar el espectro máximo utilizado es un caso particular de minimizar el intervalo entre el máximo y el mínimo *slot*, *i.e.*, el intervalo de espectro mínimo, por lo que los agrupamos juntos. A pesar de que en la Tabla 7.4 no los clasificamos de este modo, cada trabajo que utiliza un conjunto precalculado de caminos mínimos, en algún sentido tiene el minimizar las longitudes de sus caminos como uno de sus objetivos.

Objetivo	Referencias
Intervalo de espectro mínimo	[3][22][23][29][30][31][32][33][34][36][41][45][51][53][54] [67][68][69][70][71][72][73][74][78][94][96][97][102][105][104] [108][112][131][132][144][149][150][154][155][157][158][159] [161][163][165][172][173]
Longitud de las rutas	[14][19][53][79][91][100][107][136][146][176]
Espectro total	[8][36][43][55][81][82][83][91][117][118][139][142][144][157][159] [163]
Espectro medio	[3][107][128][129][148]
Tráfico aceptado	[7][36][42][58][75][83][81][82][111][121]
Costo de la red	[53][79][94][95][97][101][111][116][117][118][142][144][147][163]
Tasa de bloqueo	[5][9][10][24][34][41][47][55][58][60][75][79][80][101][103][107] [121][145][146][147][154][155][170][173]
Fragmentación	[1][24][46][47][79][84][101][170][175]

Tab. 7.4: Referencias discriminadas de acuerdo a sus objetivos.

Detalles del enfoque	Referencias
MINLP	[100][161]
Modelos ILP exactos	[1][7][8][12][14][19][22][31][32][35][36][38][42][43][51][53][58][68] [70][71][73][74][75][79][82][94][95][96][97][105][104][108][112][118] [121][142][144][146][147][150][157][158][159][162][163][164][165] [167][172]
Heurísticas basadas en ILP	[7][23][24][30][31][32][33][34][36][43][58][70][72][79][108][146][147] [148][149][164]
Heurísticas que no usan ILP	[3][8][9][10][12][22][23][32][36][38][40][45][47][51][54][55][62] [69][70][72][78][80][84][91][96][97][101][102][103][104][107][112] [131][132][136][147][148][155][157][158][159][163][164][165][167] [170][172][173][175]
Calculan camino mínimo	[1][3][7][8][9][10][12][22][23][24][29][30][31][32][33][34][36][40][41] [43][45][47][54][58][60][62][69][70][71][72][78][79][80][95][96][97] [100][101][102][104][107][108][112][131][132][136][137][139][144][146] [147][148][149][154][155][157][158][159][161][164][165][170][173] [175]
Descomponen R + (S)SA	[1][3][7][9][10][12][22][24][29][30][31][32][33][34][36][45][47][55] [62][67][70][72][78][80][96][102][107][112][131][132][136][137][144] [147][149][154][155][158][159][164][170][175]
Presentan una cota inferior	[8][67][69][71][73][74][102][104][139][146][157][158][159][161]

Tab. 7.5: Referencias discriminadas de acuerdo a los detalles del enfoque utilizado.

La Tabla 7.5 discrimina las publicaciones según el enfoque utilizado para resolver el problema, es decir, si presentan formulaciones de programación lineal entera o no lineal, y si esas formulaciones se utilizan para resolver el problema completo hasta el óptimo o si son sólo parte de un enfoque heurístico; también separa los casos en los que las heurísticas

presentadas no hacen uso de modelos ILP, las cuales generalmente se basan en algoritmos genéticos. Tanto para los algoritmos exactos como para los heurísticos, mostramos aquellas publicaciones en las que el enfoque intenta de alguna manera descomponer el problema en una fase de ruteo y una fase de asignación de espectro (y espacio) y slots. La mayoría de estas publicaciones utilizan algún algoritmo para calcular la ruta más corta para cada demanda. Finalmente, agrupamos las formulaciones que presentan alguna cota inferior para la solución del problema estudiado. Cuando una formulación de ILP propuesta que es capaz de devolver la solución óptima al problema utiliza sólo un subconjunto de rutas, canales, *lightpaths* o configuraciones precalculadas, el trabajo pertenece a ambos grupos.

Finalmente, estudiamos con más detalle los modelos de programación entera propuestos y los clasificamos en cuatro grupos de acuerdo con su enfoque de modelado. Estos grupos, que dependen de la forma en la cual manejan el ruteo y la asignación de canales, son una combinación de los enfoques explicados en la Tabla 7.6.

Enlace-ruta	Relaciona las demandas con los caminos o *lightpaths*, por lo tanto precalculándolos todos (cuando el óptimo es deseable) o sólo un subconjunto de ellos (cuando un resultado subóptimo alcanza).
Nodo-enlace	También conocido como nodo-arco, recurre a variables que relacionan enlaces y/o nodos con demandas para forzar las restricciones de flujo.
Basado en canales	Hace uso de conjuntos de canales espectrales precalculados, donde cada canal se identifica por un subconjunto de *slots* contiguos, para satisfacer una demanda particular. En este caso, la formulación de ILP utiliza un conjunto de variables que representan la selección de cierto canal para cada demanda.
Basado en *slots*	Modela la asignación de espectro explícitamente asignando el *slot* inicial para cada demanda evitando colisiones entre las demandas que compiten por los recursos utilizando restricciones adecuadas. Para una discusión más detallada del modelado de espectro en redes de grillas flexibles ver [153].

Tab. 7.6: Clasificación de los enfoques de modelado (en formulaciines ILP).

Enfoque de modelado	Basado en canales	Basado en *slots*
Enlace-ruta	[36][38][43][42][51][58][71][73] [74][75][68][112][118][121][142] [146][147][148][149][150]	[8][12][23][30][32][33][34][31][68] [70][72][94][95][96][108][142][144] [145][146][164][165]
Nodo-enlace	[7][68][81][82][146][147]	[14][19][22][35][53][83][104][105] [144][145][157][158][159][163][172]

Tab. 7.7: Referencias a los modelos ILP separados de acuerdo al enfoque de modelado.

También agrupamos las referencias sobre formulaciones de ILP según algunas particularidades en su formulación y resolución. Es decir, si unen múltiples objetivos en una sóla función, si proponen desigualdades válidas, estrategias de *pre-solving*, heurísticas primales, o si utilizan técnicas de generación de columnas. Esta clasificación se resume en la

Tabla 7.8.

Detalles del modelo ILP	Referencias
Objetivo múltiple	[36][75][82][83][79][94][97][104][105][118][121][142][144][163]
Pre-solver	[35]
Heurísticas primales	[36]
Desigualdades válidas	[19][53][73][74][75]
Generación de columnas	[42][43][51][58][71][73][74][75][121][122][147]

Tab. 7.8: Referencias a las formulaciones ILP separadas según sus particularidades.

y_{de}	el enlace e es asignado a la demanda d	[14][22][35][36][53][68][97] [108][146][172][176]
l_d	el primer *slot* utilizado por la demanda d	[14][22][30][31][32][33][34] [68][95][96][97][108][142] [144][146][165][172]
r_d	último *slot* usado por la demanda d	[14][22]
$p_{dd'}$	$r_d < l_{d'}$	[14]
$n_{dd'}$	los caminos asignados a las demandas d y d' comparten al menos un enlace y $r_d < l_{d'}$	[14][97][172]
l_{ds}	s es el primer *slot* usado por la demanda d	[14]
r_{ds}	s es el último *slot* usado por la demanda d	[14][35][53]
x_{ds}	d usa el *slot* s	[14][83][104][105]
u_{des}	d usa s en el enlace e	[8][14][19][35][53][82][83] [104][105][163][165]
l_{des}	d usa s en e, y s es su primer *slot*	[8][14][58][163]
a_{ds}	d usa *slots* mayores que s	[14]
b_{ds}	d usa *slots* menores que s	[14]

Tab. 7.9: Referencias a familias de variables usadas por los modelos ILP propuestos en el presente trabajo.

También hemos analizado todas las variables halladas en la bibliografía. La Tabla 7.9 muestra los trabajos que utilizan las mismas variables pertenecientes a los modelos propuestos en el presente trabajo, mientras que la Tabla 7.10 contiene todas las demás variables encontradas en la literatura. Para algunas de ellas, omitimos los subíndices relacionados a problemas específicos, como núcleos, modos, nivel de modulación, entre otros. Asimismo, hemos unificado tanto los conceptos de *slot* y *wavelength* como los de arco, arista y enlace, enunciando todas las variables para *slots* y *enlaces*. Junto a los ya mencionados conceptos de *slot*, *camino* y *lightpath*, algunos modelos utilizan el concepto de *canal*, el cual es simplemente un conjunto de *slots* adyacentes, y el de *spectral-spatial channel* (SSCh), que abarca los conceptos tales como núcleos y/o modos. Como se mencionó anteriormente, al presentar las variables, estos últimos son tratados simplemente como canales. En 2016 [58], aparece el concepto de *configuración*, definido como un conjunto de *lightpaths* que comienzan en el mismo *slot*, a pesar de que en el modelo presentado en 2018 [43], Enoch y Jaumard utilizan *lightpaths* nuevamente. En 2020 [42], Enoch aclara (o agrega a la definición) que los *lightpaths* de una configuración deben ser disjuntos en enlaces.

a_d	la demanda d es rechazada	[42][75][83][121][146][147]
b_p	la ruta p es seleccionada	[30][31][32][33][34][82][146]
f_l	el *lightpath* l es utilizado	[43][79]
z_c	la configuración c es utilizada	[42][58]
f_{pd}	la ruta p es asignada a la demanda d	[8][68][71][73][74][94][95] [96][97][108][118][164][165] [176]
g_{ps}	el *slot* s es asignado a la ruta p	[70][72][144][146]
h_{ps}	el *slot* s es asignado a la ruta p como el primer slot de alguna demanda	[70][72]
k_{cd}	el canal c es asignado a la demanda d	[36][68][142]
m_{cp}	el canal c es asignado a la ruta p	[118][146]
$o_{dd'}$	$l_d < l_{d'}$	[30][31][32][33][34][94][95] [96][142][146][164][165]
q_{de}	primer *slot* utilizado por d en el enlace e	[164]
$t_{dd'}$	$r_d < r_{d'}$	[22]
v_{es}	s es el primer *slot* usado en el enlace e	[12]
u_{es}	el *slot* s es utilizado en el enlace e	[12][23][51][68][70][71][72] [73][74][112][148][149][150]
$w_{dd'}$	las demandas d y d' comparten algún enlace	[83][97][108][172]
z_{ld}	el *lightpath* l es asignado a la demanda d	[51][71][73][74][75][112][121]
$z_{dd'}$	las demandas d y d' comparten algún enlace y $l_d < l_{d'}$	[68][108]
a_{pds}	el *slot* s es asignado a la ruta p como el primer *slot* de la demanda d	[23]
g_{cpd}	el canal c es asignado a la ruta p para satisfacer la demanda d	[68][147][148][149][150]
h_{cde}	el canal c es asignado a la demanda d en el enlace e	[7][68][146][147]
k_{ces}	el *slot* s es usado en el arco e dentro del canal c	[118]
$m_{dd'e}$	las demandas d y d' comparten el enlace e	[108]
$n_{dd'e}$	$l_d < l_{d'}$ en el enlace e	[94]
$o_{dd'e}$	la/s demanda/s d y/o d' utiliza/n el enlace e	[68]
q_{pds}	la ruta p es asignada a la demanda d y s es el primer slot	[12]
r_{des}	d usa s en e, y s es su último *slot*	[35]
a_{ijes}	existe un *lightpath* entre los nodos i y j utilizando el *slot* s en el enlace e	[144]
b_{pdes}	el *slot* s en el enlace e de la ruta p es utilizado por la demanda d	[164]
a_{ijdes}	existe un *lightpath* entre los nodos i y j utilizando el *slot* s en el enlace e para satisfacer la demanda d	[142][157][158][159]

Tab. 7.10: Referencias a las familias de variables utilizadas por los modelos ILP hallados en la literatura.

La mayoría de los trabajos experimentan sobre instancias que utilizan las topologías *n6s9* y NSF, con 6 nodos y 9 enlaces bidireccionales y con 14 nodos y 21 enlaces bidireccionales, respectivamente. Sin embargo, hay muchas otras topologías presentes en diversos trabajos previos. La Tabla 7.11 clasifica las publicaciones de acuerdo a las topologías utilizadas para las pruebas, y en el Appendix 9 mostramos veinte de ellas utilizadas en el presente trabajo junto con el script desarrollado para generar las instancias a partir de ellas.

Topología	Referencias
Anillos	[67][118][131][132][139][142][146][158][161]
Árboles	[67]
5n-14m-Spain	[35]
6n-14m	[36]
6n-18m-n6s9	[12][19][22][31][32][29][36][53][54][70][96][95][104][112][158][159][162][165][172]
8n-24m	[36]
8n-32m	[36]
10n-24m-BRASIL	[146]
10n36m	[36]
10n-42m-SmallNet	[23]
10n-44m-SmallNet	[19][22][53]
11n-28m-Abilene	[7][9]
11n-36-EON	[112]
11n-46m-NJ-LATA	[161]
11n-52m-COST239	[14][19][22][36][80]
12n-40m-BRAZILIAN	[9][74][71][73]
12n-30m-ABILENE	[146]
12n-32-JPN	[47]
14n-42m-NSF	[5][9][10][12][14][19][22][23][53][55][60][70][72][78][80][79][82][83][84][91][104][132][131][155][158][159][157][161][165][164][163][170][175]
14n-46m-GDT	[5][14][19][31][32][29][34][51][97][108][112][146][172]
15n-46m-NSF	[3][19][45][69][149][148]
15n-54m-CHNNET	[164]
16n-46m-EURO	[3][19][149][148]
17n-50m-German	[35]
19n-76m-EON19	[9][10][14][19][40][78][91]
20n-62m-ARPANet	[19][164]
20n-78m-EON20	[19]
21n-62m-DT	[14][24]
21n-70m-Spain	[14][19][24][42][43][53][58][75][121][146]
21n-72m-Italian	[19]
21n-78m-UKNet	[14][19]
22n-70m-BT	[14][19][24][73][147]
24n-86m-UBN24	[1][3][19][43][42][55][72][70][69][79][84][104][149][148][155][163][170][175]
26n-84m-US26	[107]
28n-68m-EON	[19]
28n-82m-EURO28	[3][19][73][91][107][131][148][149]
28n-88m-US-Backbone	[47][132]
30n-112m-Spain	[19][147]
40n-114m-USA	[78][91]
43n-176m-Euro	[19]

Tab. 7.11: Referencias discriminadas según las topologías utilizadas en los experimentos.

8

Apéndice: Cronología del estado del arte

En este apéndice resumimos, en orden cronológico, más de 120 publicaciones, que representan gran parte del trabajo total relacionado con el RSA, destacando las particularidades más relevantes para nuestro estudio, así como la variación del problema abordado, el enfoque utilizado, las particularidades y complejidades de los algoritmos desarrollados, los detalles sobre los experimentos realizados, mencionando principalmente el tipo y tamaño de las instancias, entre otros. Hemos navegado hasta los orígenes de algunos de los algoritmos más utilizados, como los diseñados para calcular las rutas mínimas de manera eficiente. La Tabla 8.1 resume las siglas y abreviaturas utilizadas a lo largo del apéndice.

8.1. Trabajos relacionados

En 1970 [169], Yen presentó un algoritmo para hallar un camino mínimo desde cada uno de los nodos del grafo a un nodo de destino particular. Este algoritmo es presentado para redes generales *N-nodes* (permitiendo distancias negativas). Si no existe ningún ciclo negativo, el procedimiento requiere $\frac{1}{2}M(N-1)(N-2)$, $1 < M \leq N-1$, sumas y comparaciones. La existencia de un ciclo negativo –siempre que exista alguno– es detectada después de $\frac{1}{2}M(N-1)(N-2)$ sumas y comparaciones. En ese mismo año [168], el mismo autor presentó otro algoritmo para hallar los k caminos mínimos desde un nodo a otro en la red, utilizando el procedimiento presentado en [169]. La cota superior computacional del nuevo algoritmo crece sólo linealmente con el valor de k.

En 1975 [64], Johnson propuso el mejor algoritmo conocido hasta la fecha para hallar todos los *circuitos elementales* (actualmente llamados *ciclos simples*) en un grafo dirigido $G = (V, E)$ en tiempo acotado por $\mathcal{O}((|V|+|E|)(c+1))$ y espacio acotado por $\mathcal{O}(|V|+|E|)$, donde c es la cantidad de circuitos elementales en el grafo. El algoritmo se asemeja a los procedimientos presentados por Tiernan y Tarjan en 1973 [141], pero el propuesto por Johnson es más rápido dado que considera cada arista como máximo dos veces entre un circuito y el siguiente en la secuencia de salida.

En 1981 [44], Entringer y Slater mostraron que el número de ciclos $\Psi(G)$ en un grafo $G = (V, E)$ con $|V|$ vértices y $|E|$ aristas está acotado superiormente por 2^n e inferiormente

por 2^{n-1}, siendo $n = |E| - |V| + 1$. El trabajo muestra también que la cota inferior es suficientemente ajustada, definiendo un grafo que la satisface por igualdad para cada valor de $n \geq 3$. Por último, los autores conjeturaron que, bajo ciertos supuestos, es posible ajustar aún más la cota superior.

En diciembre de 1991 [133], Sjogren demostró que el número de árboles generadores etiquetados de un grafo conexo es igual al determinante de la matriz de intersección para una base integral de sus ciclos, es decir, si V es el conjunto de vértices del grafo, entonces el número de árboles generadores distintos puede llegar a ser $|V|^{|V|-2}$.

En 2003 [92], Martins et al. presentaron una nueva implementación del algoritmo de Yen para clasificar los k caminos mínimos sin ciclos entre un par de nodos en una red [168]. Tanto el algoritmo original como esta nueva implementación presentan complejidad computacional $\mathcal{O}(Kn(m + n \log n))$, sin embargo, el procedimiento propuesto en este artículo supera en la práctica a la implementación de Perko [114] y a otra más sencilla.

En 2004 [56], Hülsermann et al. presentaron tres escenarios para el modelado de redes de transporte típicamente utilizadas en ese momento. Dichas redes son *German network*, con 17 nodos y 26 enlaces, *European network*, con 28 nodos y 41 enlaces, y *US network*, con 14 nodos y 21 enlaces. El artículo también presenta algunos parámetros topológicos, como los grados de los nodos, la longitud máxima, mínima y promedio de los enlaces, y el diámetro de red en kilómetros y *hops*, entre otros. Los autores también calcularon matrices de tráfico que hoy en día quedaron claramente desactualizadas.

8.2. Nacimiento de la EON y el RSA

En 2008 [61] y 2009 [63], Jinno et al. presentaron una *elastic optical network* (EON) de espectro eficiente basada en la tecnología de *orthogonal frequency-division multiplexing* (OFDM), para superar las limitaciones de las redes ópticas tradicionales. OFDM asigna los datos a varios canales subportadores de baja velocidad de datos. Como el espectro de los canales subportadores adyacentes se modula ortogonalmente, éstos pueden superponerse entre sí, lo que aumenta la eficiencia espectral de transmisión. Además, la OFDM óptica puede proporcionar capacidad de granularidad fina a las conexiones mediante la asignación elástica de subportadoras de baja velocidad. Ésta es la tecnología que da origen al RSA.

8.3. 2010

En enero de 2010 [41], Durães et al. introdujeron el problema de la mejor elección entre M combinaciones de caminos mínimos para el provisionamiento dinámico de *lightpaths* en redes ópticas. El rendimiento del algoritmo *best among the shortest routes* (BSR) se comparó en términos de probabilidad de bloqueo y utilización de la red con el algoritmo de camino mínimo de Dijkstra y otros algoritmos propuestos en la literatura. Para todos los escenarios estudiados, BSR logró un rendimiento superior.

En julio de 2010 [105], Patel et al. demostraron que era posible mejorar la *wavelength-division multiplexing* WDM en una red flexible que presentara la arquitectura de red óptica flexible WDM. Este artículo enuncia por primera vez el *routing, wavelength allocation, and spectrum allocation problem* (RWSA), donde para una topología de red dada y un conjunto de *wavelengths* ofrecidos, el desafío es cómo establecer *lightpaths* transparentes extremo a extremo para un conjunto dado de demandas, de manera tal de minimizar el máximo

espectro requerido para soportar el tráfico. En este trabajo también se demuestra que el RWSA pertenece a la clase $\mathcal{NP}$-difícil.

En agosto de 2010 [62], M. Jinno et al. presentaron un esquema de asignación de espectro adaptable a la distancia, el cual toma un formato de modulación de alto nivel para trayectos de larga distancia, y un formato de modulación de bajo nivel para trayectos más cortos. El artículo también presenta un algoritmo para resolver la versión dinámica del *routing and spectrum allocation problem* (RSA) donde la ruta y los *slots* contiguos se calculan de manera heurística con base en un algoritmo de ruteo, *i.e.*, *fixed-alternated*, y un algoritmo de asignación de *slots*, *i.e.*, *first-fit* (FF), bajo la restricción de continuidad del espectro.

En septiembre [30] y en diciembre de 2010 [33, 30], Christodoulopoulos et al. abordaron la versión estática del RSA para el esquema presentado por Jinno et al. [61, 63, 62], proponiendo varios algoritmos para resolverlo. Para esta versión del problema se tiene en cuenta la banda de guarda. El artículo presenta un algoritmo heurístico basado en ILP que intenta minimizar el espectro precalculando las rutas utilizadas para satisfacer las demandas. También se propone un método de descomposición que divide al RSA en dos subproblemas, a saber, por un lado el ruteo y por otro la asignación de espectro (R + SA), los cuales resuelve secuencialmente. También se presenta un algoritmo heurístico que atiende conexiones una por una y utiliza ese método para resolver el problema de planificación, satisfaciendo secuencialmente todas las demandas. Se planearon dos políticas de pedidos para alimentar el algoritmo secuencial, proponiendo una nueva denominada como *most subcarriers first* (MSF).

8.4. 2011

En enero de 2011 [103], Patel et al. introdujeron la versión dinámica del problema RWSA por vez primera, y propusieron un algoritmo goloso que intenta minimizar la probabilidad de bloqueo y el rendimiento de la red.

En febrero de 2011 [70], Klinkowski y Careglio, presentaron un modelo ILP basado en rutas precalculadas para el RSA (tres caminos mínimos para cada par de nodos). Los autores explicaron por qué es posible asignar el volumen a *slots* y por qué puede éste ser constante a lo largo de toda la ruta: «en miras a la simplicidad, consideramos que el volumen de una demanda es independiente de la longitud del camino asignado. Como se discutió en [62], la dependencia en la longitud de la ruta comienza a jugar un papel cuando $|p| \geq 10$». Las topologías utilizadas para los experimentos fueron *6n-18m*, *14n-43m-NSF* y *24n-86m-UBN24*, con $\bar{s} \leq 170$, $|D| \leq 552$, y $v(d) \in [1, 5]$ para cada $d \in D$.

En marzo de 2011 [155], Wan et al. presentaron tres algoritmos heurísticos para resolver la versión dinámica del RSA. El primero de ellos (KSP) precalcula los primeros k-caminos mínimos usando el algoritmo de Yen [168] y luego resuelve (suponemos que usando FF) el *spectrum allocation problem* (SA). La segunda heurística (MSP) es una modificación del algoritmo de camino mínimo de Dijkstra. Para el tercero, el artículo propone construir un árbol de vectores de ruta con restricciones de espectro para buscar el camino óptimo global. Los experimentos se realizaron sobre las topologías *14n-42m-NSF* y *24n-86m-UBN24*, con valores para $\bar{s}$ inferiores a 400 *slots*, y el volumen de cada demanda de a lo sumo diez *slots*.

En abril de 2011 [158], Wang et al. demostraron la pertenencia del RSA a la clase $\mathcal{NP}$-difícil y formularon un modelo ILP para minimizar –de manera óptima– el número máximo de subportadoras necesarias en cualquier enlace de la red. El artículo también

analiza los límites superior e inferior para el número de subportadoras en una red de topología general o específica, y propone dos algoritmos heurísticos para reducir el número requerido de subportadoras en una red SLICE, a saber, el algoritmo *balanced load spectrum allocation* (BLSA) y el algoritmo *shortest path with maximum spectrum reuse* (SPSR). Estas heurísticas, con pequeñas diferencias, calculan previamente los k-caminos mínimos para cada par de nodos utilizando el algoritmo presentado por Yen en [168]. Luego, para cada demanda, los procedimientos seleccionan la ruta que minimiza el espectro utilizado en los arcos, partiendo de la demanda con mayor volumen. Finalmente, estas heurísticas asignan el espectro usando un algoritmo goloso (*first-fit*). En el artículo se afirma que para redes en anillo con varias demandas de tráfico uniformes y tamaños de banda de guarda, el modelo ILP puede alcanzar la cota inferior producida por el método *cut-set* (CS); y también que los algoritmos BLSA y SPSR producen resultados cercanos a la solución ILP óptima para demandas de tráfico uniformes. Los experimentos se llevaron a cabo en topologías pequeñas con cuatro, cinco y seis nodos, y en una mediana de 14 nodos, utilizando diez demandas con volúmenes de hasta tres *slots* cada una.

En mayo de 2011 [32, 31], Christodoulopoulos et al. estudiaron el *routing, modulation level and spectrum allocation problem* (RMSA o RMLSA) en un grafo dirigido, teniendo en cuenta las bandas de guarda –que fueron fijadas en dos *slots*– y el nivel de modulación, que es interpretado como una función que relaciona la longitud de la ruta asignada a una demanda con el número de *slots* necesarios para satisfacer el ancho de banda requerido. Además, en lugar de establecer el valor para $\bar{s}$, se buscó minimizarlo, por lo que se seleccionó un valor grande para evitar la no factibilidad. En el artículo se prueba que este problema pertenece a la clase $\mathcal{NP}$-difícil, y también se presenta un algoritmo ILP para minimizar el espectro. Con este fin, se calculan previamente k rutas (tomando $k = 3$) para cada demanda utilizando una variación del algoritmo de camino mínimo (en cada paso, se selecciona un camino mínimo, y los costos de los arcos que éste utiliza se duplican para ser evitados por los caminos hallados en los pasos siguientes). El artículo también presenta un método de descomposición que divide el RMSA en dos subproblemas secuenciados, a saber, primero el ruteo y nivel de modulación y luego la asignación de espectro (RML + SA). También propone un algoritmo heurístico para resolver el RSA estático satisfaciendo las demandas una por una. La heurística secuencial propuesta, combinada con una disciplina de pedido adecuada, fue capaz ofrecer soluciones cercanas a las óptimas en tiempos de ejecución razonables. El artículo también demuestra que las redes basadas en OFDM tienen beneficios de espectro sustanciales sobre las redes WDM de red fija clásicas: "el alto número de subportadoras OFDM (del orden de varios cientos) limita la aplicabilidad de los algoritmos RWA tradicionales". También se presenta la problemática denominada *bandwidth fragmentation*. Para los experimentos, los algoritmos se ejecutaron sobre dos topologías: la *6n-18m* y la *14n-46m-GDT*, utilizando demandas con volúmenes entre 1 y 4 ó entre 1 y 30. El tiempo de ejecución fue fijado en dos horas, y un sólo nivel de modulación, *i.e.*, simplificando hacia el RSA. La instancia pequeña fue resuelta en segundos, mientras que para la más grande, a pesar de que los algoritmos fueron capaces de resolver el problema de ruteo de forma óptima, no alcanzaron el óptimo para el SA en casi ningún caso. El mismo año [29], estos autores presentaron un algoritmo heurístico para resolver la versión dinámica del RMSA con bandas de guarda, el cual también precalcula un conjunto de k-caminos mínimos para cada demanda.

En junio de 2011 [72], Klinkowski y Walkowiak señalaron la ineficiencia del algoritmo de asignación de frecuencia *first-fit* discutido por Jinno et al. en [62] en 2010. Además

de proponer un algoritmo ILP, los autores también presentaron un algoritmo heurístico, llamado *adaptive frequency assignment, collision avoidance*, el cual selecciona de forma adaptativa la secuencia de demandas procesadas para minimizar el espectro utilizado. Las comparaciones fueron realizadas con dos algoritmos de referencia, a saber, FA-FF y MSF; el último presentado por Christodoulopoulos et al. en [33].

Wang et al. en junio de 2011 [157], presentaron nuevamente el modelo ILP propuesto en [158] con diferentes objetivos de optimización y examinaron enfoques para encontrar la cota inferior y superior para el número de subportadoras. También estudiaron, bajo diferentes objetivos de optimización, los dos algoritmos heurísticos presentados en el mencionado trabajo anterior, a saber, BLSA y SPSR.

Internet Topology Zoo [76] es un servidor de datos de red creado por Knight et al. en octubre de 2011, a partir de la información que los operadores de red hicieron pública. Hasta donde sabemos, es la mayor y más precisa colección de topologías de red disponible, e incluye metadatos que no podrían haberse medido. Presenta 261 topologías con fechas de 1969 a 2011.

En noviembre de 2011 [60], Jin et al. propusieron cuatro esquemas para el RSA dinámico en redes ópticas de ancho de banda flexible, incluyendo los esquemas *routing based* (RB) RSA, *signaling-based* (SB) RSA, *hybrid* RSA, y *k-path signaling-based* RSA; comparándolos por la probabilidad de bloqueo (BP) de la red, la cual consta de dos tipos básicos: 1) bloqueo debido a una capacidad de red insuficiente, que se refleja en la *forward blocking probability* (BBP); y 2) bloqueo debido a información desactualizada, que se refleja en la *backward blocking probability* (BBP). Para los experimentos se utilizó la topología *14n-42m-NSF*, fijando el ancho de banda total en $4THz$, y los volúmenes de las demandas en el rango $[10, 100GHz]$. Asumiendo $12.5Ghz$ por *slot*, tenemos $\bar{s} = 320$, y $v(d) \in [1, 10]$.

8.5. 2012

En enero de 2012 [104], Patel et al. continuaron el trabajo iniciado en 2010 [105] sobre el RWSA con el objetivo de maximizar la eficiencia espectral. Este artículo prueba la $\mathcal{NP}$-completitud del problema, presenta una formulación ILP, y propone tres heurísticas eficientes, *i.e.*, de tiempo polinomial; a saber, *greedy-routing, wavelength assignment, and spectrum allocation* (Greedy-RWSA), *k-alternate paths routing, wavelength assignment, and spectrum allocation* (KPaths-RWSA), y *shortest path routing, wavelength assignment, and spectrum allocation* (SP-RWSA). Los autores también analizaron la cota inferior del espectro requerido dada una topología y un conjunto de demandas.

En febrero de 2012 [49], Jinno et al. describieron los *drivers*, los bloques de construcción, la arquitectura y las tecnologías habilitadoras para el paradigma EON, así como los primeros esfuerzos de estandarización.

Un *survey* sobre tecnologías de redes y transmisión óptica de alta velocidad basadas en *orthogonal frequency-division multiplexing* (OFDM) –con un enfoque específico en la tecnología OFDM– fue presentado por Zhang et al. en febrero de 2012 (y publicado en 2013) [171].

En julio de 2012 [20] (y publicado en 2013), Birmelé et al. presentaron el primer procedimiento óptimo para listar todos los ciclos en un grafo no dirigido $G = (E, V)$, mejorando el tiempo del algoritmo de Johnson [64] por un factor que puede ser $\mathcal{O}(|V|^2)$. Específicamente, si $C(G)$ denota el conjunto de todos los ciclos, para un ciclo $c \in C(G)$, llamemos $|c|$ al número de aristas en c. Entonces, el algoritmo propuesto requiere $\mathcal{O}(m +$

$\sum_{c \in C(G)} |c|$) y es asintóticamente óptimo: de hecho, se requiere $\Omega(m)$ para leer G como entrada, y $\Omega(\sum_{c \in C(G)} |c|)$ para listar la salida.

En julio de 2012 [34], Christodoulopoulos y Varvarigos consideraron el RMSA tanto en su versión estática como dinámica. Ampliando el trabajo [32], los autores presentaron dos algoritmos de planificación heurísticos para el *impairment-aware RMSA* (IA-RMSA), uno de ellos basado en la formulación anterior de ILP, y políticas de asignación de espectro dinámico para satisfacer un tráfico variable en el tiempo. En el problema IA-RMSA, una vez que se elige el formato de modulación y el espectro, el transpondedor flexible se puede sintonizar para transmitir una cantidad de Gbps a través de una ruta de una distancia particular utilizando una banda de guarda de g *slots* respecto de las conexiones adyacentes para que la transmisión presente una calidad aceptable.

En agosto de 2012 [154], Wan et al. presentaron un método de descomposición para resolver el RMSA dinámico en tres pasos. En el primer paso, se realiza la selección del formato de señal. En el segundo paso, se resuelve el RSA sin tener en cuenta las distancias de transmisión. La solución de este problema se basa en determinar el camino mínimo con la asignación de espectro adecuada para la demanda entrante. El tercer paso incluye verificar la distancia de transmisión.

En agosto de 2012 [24], Castro et al. propusieron un algoritmo heurístico para resolver la versión dinámica del RMSA. El objetivo buscado en el trabajo es minimizar la probabilidad de bloqueo. Para conseguirlo, las rutas candidatas se calculan previamente con un algoritmo de camino mínimo adaptado del algoritmo de Dijkstra, y la asignación de *slots* se resuelve en una segunda fase. El artículo también estudia la cantidad de *slots* óptima en función del tráfico a ser atendido, y propone un algoritmo basado en ILP para reasignar algunas conexiones ópticas ya establecidas con el fin de dejar espacio en el espectro para las nuevas. Los experimentos fueron llevados a cabo sobre las topologías *21n-70m-SpanishTelefonica*, *22n-70m-British-telecom*, y *21n-62m-DT*. El espectro fue fijado en $800GHz$ utilizando diferentes anchos de banda para los *slots*, e.g. 50, 25, 12.5, y $6.25GHz$, mientras que las bandas de guarda no fueron tenidas en cuenta.

Klinkowski, en septiembre de 2012 (publicado en 2013) [69], presentó un algoritmo genético (GA) para resolver la versión estática del *RSA with dedicated path protection and same channel allocation RSA/DPP/SC*, *i.e.*, un *lightpath* primario y otro de *backup* para cada demanda utilizando el mismo intervalo de *slots*. En este trabajo se precomputa un conjunto de rutas candidatas utilizando un algoritmo de camino mínimo, mirando a la suma total sobre el camino primario y el *backup*, y la función objetivo busca minimizar el máximo *slot* necesario, Φ. Para los experimentos se utilizaron cuatro topologías, siendo *24n-86m-UBN24* la mayor. Los parametros utilizados son: $\bar{s} = 1500$ (buscando que $\Phi \leq \bar{s}$) y $|D| \in \{2, 3, 5, 10, 30\}$ con $v(d) \in \{2, 4, 6, \ldots, 16\}$ para toda demanda $d \in D$. La comparación fue realizada contra un algoritmo basado en grafos auxiliares propuesto en [106] para *dedicated path protection*, y un algoritmo RSA basado en ordenamiento de demandas en función del número de *slots* requerido [32]. Para instancias pequeñas, la comparación fue realizada con una formulación ILP basada en asignación de canales presentada en [146]. El artículo también propone una cota inferior *(LB)* para Φ, obtenida resolviendo un *multicommodity routing problem*, presentado en la Sección III-A de [32].

En noviembre de 2012 [159], Wang et al. completaron el trabajo comenzado en 2011 [158]. Los autores definieron formalmente el RSA teniendo en cuenta bandas de guarda fijas y mostraron la pertenencia de dicho problema a la clase $\mathcal{NP}$-difícil. En el trabajo se presentan dos modelos de IP para lograr diferentes objetivos de optimización: minimizar

el máximo intervalo de espectro utilizado, y minimizar el espectro general. Uno de estos modelos presenta una única familia exponencial de variables binarias que establece para cada demanda d, *slot* s y par de nodos i, o, si hay una ruta de i a o usando s para satisfacer d. También se analizaron cotas superior e inferior para el espectro utilizado, y nuevamente se propusieron las dos heurísticas SPSR y BLSA, logrando resolver el RSA en redes de gran escala.

En diciembre de 2012 [102], Patel et al. continuaron el trabajo empezado en 2010 [105] y seguido en enero de 2012 [104] acerca del RWSA, presentando un algoritmo metaheuristico para resolver dicho problema dividiéndolo en dos fases. La fase de ruteo es resuelta precomputando un subconjunto de k-caminos mínimos para cada par de nodos, los cuales son utilizados para la generación de población del algoritmo genético. El objetivo buscado es maximizar la eficiencia del espectro. Para evaluar la calidad de la solución, los autores enuncian una cota inferior basada en un concepto de teoría de cortes.

En diciembre de 2012 [173], Zhou et al. desarrollaron un algoritmo genético adaptativo para resolver el RMSA dinámico. Para casos con bajo tráfico, cuando no hay bloqueo, el algoritmo minimiza el mayor número de *slots* requeridos en cualquier enlace de la red; caso contrario, minimiza la probabilidad de bloqueo. El rendimiento de dicho algoritmo fue evaluado en las topologías *14n-42m-NSF* y *28n-88m-US-Backbone*.

En diciembre de 2012 [146], Velasco et al. propusieron dos modelos ILP para resolver la versión estática del RSA con bandas de guarda fijas, utilizando un conjunto de k-caminos mínimos precomputados (seleccionando $k \in \{1, 2, 3, 4, 5, 10, 15, 20\}$) para cada par de nodos, y, por primera vez, también un conjunto de canales precomputados. El objetivo a minimizar es el ancho de banda no satisfecho, *i.e.*, la suma de los anchos de banda de las demandas rechazadas. El artículo también propone un modelo ILP para resolver el mismo problema hasta el óptimo, minimizando la cantidad de enlaces utilizados. Para acelerar los tiempos de resolución, el trabajo propone una relajación del modelo de tipo enlace-ruta con una cota inferior a la función objetivo, la cual viene dada por la resolución del RSA relajando las restricciones de contigüidad y continuidad. Para los experimentos se utilizó $\bar{s} \in [30, 60, 64, 250]$ y $|D| = 36$ con $v(d) \in [1, 4]$ sobre las topologías *21n-70m-SpanishTelefonica*, *14n-46m-GDT*, *12n15m-ABILENE*, *10n12m-BRASIL*, y un anillo de nueve nodos.

8.6. 2013

En marzo de 2013 [46], Fujii et al. propusieron una asignación de espectro bajo demanda para EON con fibras de múltiples núcleos, introdujeron un índice simple de *crosstalk* y evaluaron el rendimiento básico mediante la probabilidad de bloqueo y el solapamiento de espectro (el cual causa *crosstalk* intra núcleo).

En abril de 2013 [132], Shirazipourazad et al. demostraron que el RSA es $\mathcal{NP}$-difícil incluso cuando la topología es una cadena o un anillo, y propusieron algoritmos aproximados cuando la topología de red es un árbol binario o un anillo. En el documento se demuestra que el problema del RSA que no tiene en cuenta las bandas de guarda es fácilmente transformable en el RSA que sí las tiene en cuenta, por lo que una solución óptima del primer problema puede convertirse polinomialmente en una solución del segundo. La transformación consiste simplemente en agregar el ancho de la guarda al volumen de las demandas. En el artículo también se propone una heurística, que busca reducir el intervalo de espectro utilizado, y en la cual se realiza un proceso iterativo tanto para precomputar

los caminos mínimos para cada demanda como para asignar el espectro. En cada iteración, se recorre la lista de demandas que aún no tienen *lightpath* asignado, ordenadas de mayor a menor respecto de sus volúmenes, y para cada una de ellas se busca en el grafo el camino más corto disponible que une el origen con el destino. Sin embargo, dicho procedimiento no permite utilizar arcos ya utilizados por otra demanda en la misma iteración. Luego, cuando ya no es posible asignar ninguna demanda más, se realiza la asignación de *slots* y se repite el proceso con el resto de las demandas, pero nuevamente con todos los arcos del grafo disponibles. Este proceso se repite hasta que todas las demandas tienen asignado un *lightpath*. Los experimentos fueron llevados a cabo sobre las topologías *14n-42m-NSF* y *28n-88m-US-Backbone*.

En junio de 2013 [131], Shirazipourazad et al. demuestran nuevamente que la versión estática del RSA es $\mathcal{NP}$-difícil incluso para anillos. En el trabajo se propone un algoritmo para este tipo de topologías y se provee una heurística para redes arbitrarias para la versión dinámica del RSA sin tener en cuenta las bandas de guarda. Para ello se presenta nuevamente la transformación de un problema al otro, como en [132]. El objetivo del algoritmo es minimizar el intervalo de espectro utilizado. Luego de resolver el ruteo, el SA es reducido a calcular el *interval chromatic number* (ICN) del grafo de intersección de los caminos seleccionados. Para los experimentos se utilizaron las topologías *14n-42m-NSF* y *28n-82m-EURO28*.

En junio de 2013 [121], Ruiz et al. propusieron un modelo multiobjetivo y técnicas de generación de columnas para resolver el RSA minimizando el número de demandas rechazadas y la cantidad de ancho de banda no servido. El modelo utiliza dos conjuntos exponenciales de variables binarias para relacionar demandas con *lightpaths* y con canales, y otro conjunto lineal que indica si una demanda es satisfecha o rechazada. En este trabajo se lograron resolver instancias con hasta 96 *slots*, y un conjunto de demandas distribuidas uniformemente sobre un total de 180 pares de nodos tomados de la red *21n-70m-SpanishTelefonica*.

En junio de 2013 [149], Walkowiak y Klinkowski formularon la versión estática del *joint anycast and unicast routing and spectrum allocation problem* (RSA/JAU) y propusieron un modelo ILP para resolverlo. Esta formulación utiliza conjuntos precomputados de caminos y canales para cada demanda. El trabajo propone también algoritmos heurísticos dedicados. Una de las conclusiones principales del artículo es que el ruteo *anycast* aporta importantes ventajas en cuanto a la utilización del espectro sobre EON.

En agosto de 2013 [23], Capucho et al. propusieron dos heurísticas: un modelo ILP basado en la configuración enlace-ruta y un algoritmo genético, para resolver la versión dinámica del RSA con bandas de guarda y asumiendo un formato de modulación $QPSK$ con subportadoras de $12.5GHz$. La formulación ILP, que utiliza un conjunto precalculado de k-caminos mínimos e intenta minimizar el mayor *slot* ocupado, se utiliza únicamente como un *benchmark* para calibrar la metaheurística sobre una topología pequeña de 6 nodos. Los experimentos finales fueron llevados a cabo con una topología de 10 nodos (creemos que *10n-42m-SmallNet*) y la topología *14n-42m-NSF* con $v(d) \in \{2, 4, 6, 8\}$ *slots* para cada demanda d.

En septiembre de 2013 [100] (publicado en 2014), Olszewski presentó un modelo de *nonlinear integer programming* (NILP) para resolver la versión dinámica del RSA. Esta formulación utiliza una variable para cada *lightpath*. La función objetivo utilizada busca minimizar la longitud de los caminos seleccionados. Para esta versión del RSA se tuvo en cuenta la velocidad de transmisión, la distancia, y las bandas de guarda. El artículo

también propone dos heurísticas basadas en modificaciones al algoritmo de Dijkstra y las compara contra la heurística presentada en [154].

En septiembre de 2013 [22], Cai et al. presentaron un modelo ILP para una simplificación del RSA asumiendo que para cada par origen-destino (i, j) el tráfico de la demanda de i a j es igual al requerido de j a i, y también que los nodos pertenecientes a la ruta que va de i a j son los mismos que los de la ruta que une j con i, *i.e.*, ambas rutas atraviesan exactamente los mismos nodos pero en sentido opuesto. Motivo por el cual se utiliza un grafo no dirigido para representar la topología. El modelo recurre a una variable binaria que relaciona cada par origen destino (i, j) con cada enlace, y otro que relaciona dicho par (i, j) con cada vértice. También tiene dos familias de variables enteras que indican para cada demanda el primer y último *slot* utilizados por ella. Por último, una variable binaria indica si el mayor *slot* asignado a una demanda es mayor que el mayor *slot* asignado a otra. El trabajo presenta también una heurística que resuelve el problema dividiéndolo de la forma acostumbrada, *i.e.*, R+SA, y utilizando caminos mínimos para la primera fase. Para los experimentos con el modelo exacto, se utilizaron las topologías *6n-18m* y *10n-44m-SmallNet* comparando con las formulaciones presentadas en [159], [32] y [146] con pequeñas modificaciones, y unificando la función objetivo. Las heurísticas fueron comparadas además en las topologías *11n-52m-Pan-European-COST239* y *14n-42m-NSF*.

Dos algoritmos para resolver el RSA dinámico, a saber, *fragmentation-aware RSA* y *fragmentation-aware RSA with congestion avoidance*, fueron propuestos por Yin et al. en octubre de 2013 [170], buscando aliviar la fragmentación espectral en el proceso de provisión de *lightpaths*. Estos procedimientos utilizan conjuntos precomputados de caminos mínimos entre todos los pares de vértices. Los experimentos fueron llevados a cabo sobre las topologías *14n-42m-NSF* y *24n-86m-UBN24*, con $\bar{s} = 400$, los volúmenes de las demandas en el rango de $[1, 10]$ *slots*, y comparando con los algoritmos *shortest path routing and first-fit* (SP-FF) y *k-shortest path routing and first-fit* (KSP-FF).

En noviembre de 2013 [148], Walkowiak et al. propusieron un modelo ILP y un algoritmo de tipo *tabu search* (TS) para el RSA con tráfico tanto *unicast* como *anycast* en EONs, minimizando el promedio del espectro utilizado en cada enlace. Para el modelo ILP se precomputan conjuntos de caminos y canales.

En diciembre de 2013 [5], Almeida et al. propusieron un algoritmo para el SA basado en el ordenamiento de espectro FF [62], donde a los *slots* se les da prioridad con respecto al número de formas en que se podrían asignar en el futuro a las demandas con ancho de banda variable. Los experimentos fueron llevados a cabo sobre las topologías *14n-42m-NSF* y *14n-46m-GDT*, con $\bar{s} = 128$ y $v(d) \in [1, 32]$ para toda demanda $d \in D$.

En diciembre de 2013 [9], Assis et al. presentaron un algoritmo heurístico (un nuevo BSR [41]) para resolver el RSA dinámico. Para este algoritmo se precomputan los k-caminos mínimos para cada demanda y luego se selecciona uno para cada una entre éstos, buscando balancear la distribución de enlaces utilizados, con el objetivo de minimizar la probabilidad de bloqueo. Los experimentos fueron realizados en las topologías *14n-42m-NSF*, *19n-76m-EON19*, *12n-20m-BRAZILIAN*, y *11n-28m-Abilene*, con hasta 256 *slots* disponibles por arco, y volúmenes de demanda de hasta 16 *slots*.

8.7. 2014

En enero de 2014 [77], Leepila et al. propusieron un esquema de basado en ILP para el diseño de redes eficientes –en cuanto al consumo de potencia– con protección de ruta

con ancho de banda parcial. El trabajo considera, no sólo la capacidad consumida, sino también el número de enlaces y nodos usados en la red.

En febrero de 2014 [144] y [145], Varvarigos y Christodoulopoulos describieron diversas técnicas utilizadas para resolver las versiones estáticas de los distintos problemas de optimización que surgen al trabajar con redes. Algunas de las técnicas explicadas son ILP, LP, metaheurísticas, heurísticas, algoritmos golosos, entre otras. El trabajo también presenta un modelo ILP para el RWA que utiliza una única familia de variables binarias que relaciona cada par de nodos con cada enlace y cada *wavelength*. Luego, se presenta una heurística basada en LP con una única familia de variables que relaciona cada una de las rutas precomputadas con cada *wavelenght*, y también una heurística basada en ILP para el RMSA con el objeto de minimizar el mayor *slot* asignado y el costo de los transpondedores utilizados. En el artículo [145], los autores mencionan que este último algoritmo fue adoptado con el fin de resolver indirectamente la versión dinámica del RSA minimizando la probabilidad de bloqueo.

En mayo de 2014 [96], Muhammad et al. formularon la versión estática del *routing, spectrum and core allocation problem* (RSCA) presentando un modelo de ILP. El objeto buscado en este trabajo es minimizar el máximo número de *slots* requeridos en cualquier núcleo de la *multi-core fiber* (MCF) de una red *space division multiplexing* (SDM) con grilla flexible. La topología de la red es interpretada como un grafo dirigido y, para este problema, no se tienen en cuenta las bandas de guarda. Para su resolución se precomputan todas las rutas para cada demanda utilizando el algoritmo de Yen [168]. Para las instancias más grandes, se propone una heurística denominada *shortest path with cumulative spectrum availability* (SPSA), la cual recibe un conjunto precalculado de los k-caminos mínimos (utilizaron 3 en los experimentos) para cada demanda, e intentan satisfacer secuencialmente cada demanda recorriéndolas en orden descendente según los requerimientos de espectro. Los algoritmos fueron testeados sobre la topología más simple *6n-18m*.

En julio de 2014 [3], Albin y Walkowiak presentaron una heurística denominada *VSA* para iterar sobre una lista de demandas con el fin de resolver la fase SA del RSA. La propuesta es intercambiar demandas de a pares, calculando un valor ω con ello. Dicho proceso se realiza una cierta cantidad de veces, quedándose al final con el orden que obtuvo el mayor ω. Para medir el rendimiento, se resuelven dos modelos de ILP presentados en [149] y [148].

En julio de 2014 [75], Klinkowski et al. mejoraron la formulación presentada por Ruiz et al. en 2013 [121] mediante la adición de una familia de cortes personalizados: las desigualdades *clique*, las cuales aseguran que a lo sumo puede ser utilizado uno de un conjunto de k *lightpaths* que se solapan. Sin embargo, los resultados obtenidos fueron levemente mejores que los presentados en el ya mencionado trabajo. Los experimentos fueron llevados a cabo en una topología de 21 nodos, con $\bar{s}$ tomando valores de hasta 100 *slots*, y conjuntos de 160 demandas.

En agosto de 2014 [147], Velasco et al. propusieron una formulación ILP basada en nodo-enlace y otra en enlace-ruta, ambas utilizando conjuntos de canales precomputados, también propusieron un algoritmo de generación de columnas y algunas metaheurísticas, con el fin de resolver la versión estática del RSA. También se presentan dos métodos heurísticos para la versión dinámica del problema. Para precomputar los caminos, se propone un algoritmo heurístico en un paso, el cual es una versión restringida del algoritmo de k-caminos mínimos presentado por Yen. Esta variación agrega una poda para evitar caminos que no son capaces de soportar una demanda.

En septiembre de 2014 [95], Muhammad et al. presentaron una formulación ILP para resolver la versión estática del *routing, modulation format, spectrum, and core allocation problem* (RMSCA). Para cada demanda se calculan todos los caminos posibles, pero, para tener en cuenta el *inter-core crosstalk*, el modelo recibe para cada demanda solamente un subconjunto de dichos caminos, para los cuales los valores de *crosstalk* están por debajo de cierto límite dado. Sin embargo, en el trabajo se asegura que la solución hallada por el modelo es óptima. La formulación, adaptada de [96], tiene una familia de variables binarias que relacionan cada demanda con cada ruta candidata, y una familia que compara el primer *slot* para cada par de demandas en cada núcleo y arco. Este algoritmo fue ejecutado en instancias muy pequeñas con la topología *6n-18m*, $\bar{s} = 50$ y volúmenes en el rango $[2, 8]$.

En 2014 [36], Hai en su tesis de doctorado presentó diversos algoritmos para resolver una gran variedad de versiones del RSA y del RMSA. Para el RSA con objetivo único, se busca minimizar el espectro utilizado. Para conseguirlo, el trabajo propone un algoritmo genético y una heurística que utiliza una formulación ILP basada en enlace-ruta. Este último utiliza un conjunto de canales y k caminos precomputados (utilizando para estos últimos el algoritmo de Yen [168]). El resultado del GA (*i.e.*, los *lightpaths* y el valor objetivo) es utilizado como un *warm-start* para el modelo ILP. También se considera el RSA multiobjectivo, buscando minimizar tanto el espectro utilizado por enlace como el nivel de congestión. Para este fin, el autor implementa otra formulación basada en ruta-enlace que minimiza un objetivo único, el cual sincretiza los multiobjetivos en un solo valor. También se implementa un GA multiobjetivo. El artículo presenta además algoritmos heurísticos basados en formulaciones ILP con ciclos y rutas precomputadas para el RMSA con *dedicated path protection* (DPP) y con *shared path protection* (SP) en redes con y sin capacidades (*i.e.*, rechazando o no demandas). Los experimentos fueron llevados a cabo utilizando diversas topologías, *6n-18m*, *8n32m*, y *11n-52m-Pan-European-COST239*, entre ellas, y el máximo $\bar{s}$ utilizado fue 200.

Un *survey* enfocado en los esfuerzos de investigación aplicados a la definición de arquitecturas de planos de control de redes ópticas elásticas fue presentado, en octubre de 2014 [134], por Sócrates et al.

En octubre de 2014, Paul en su tesis de doctorado [108], presentó un modelo exacto para la versión estática del RSA basado en una formulación MILP, y lo comparó con la formulación utilizada en [33]. En este trabajo se utiliza el conjunto de todos los caminos (simples) posibles para cada demanda, se interpreta la topología como un grafo dirigido y se tienen en cuenta las bandas de guarda. El modelo utiliza una variable binaria que asocia un camino con una demanda, otra familia de variables también binarias que para cada arco y cada par de demandas determinan si los caminos asignados a éstas comparten el arco, y una tercera variable binaria que dice si las rutas de dos demandas comparten al menos un arco. También se utiliza una variable representando el primer *slot* asignado a una demanda. El objetivo buscado es minimizar el máximo espectro utilizado. Para los experimentos se utilizó la topología *14n-46m-GDT*, junto con un conjunto generado aleatoriamente de topologías de hasta 15 nodos con $\delta(v) \in \{2, 3\}$ para cada vértice v, utilizando conjuntos de 8 a 40 demandas. En el trabajo se propone una simplificación precomputando un subconjunto de los k-caminos mínimos para cada demanda utilizando el algoritmo de Yen presentado en [169] y adaptando un poco las restricciones. Finalmente, en el artículo se concluye que no fue posible manejar propiamente instancias de tamaño real con ninguna de estas dos variantes. El documento [110] publicado en 2015 resume la tesis, y el artículo [109] resume sólo la heurística.

En octubre de 2014 [55], Huang et al. presentó una heurística basada en GA para resolver el RSA dinámico minimizando la cantidad de demandas rechazadas y la suma total de *slots* utilizados. El rendimiento fue comparado sobre las topologías *14n-42m-NSF* y *24n-86m-UBN24*, con $\bar{s} = 360$ y volúmenes en el rango $[2, 12]$, contra el algoritmo genético propuesto para el RSA dinámico en [60], y el algoritmo *spectrum scan routing* presentado en [166].

En diciembre de 2014 [47], Fujii et al. estudiaron el problema del *crosstalk* en MCF en cuanto al espectro y la asignación de núcleos, desde la perspectiva de la red. El artículo propone un método de asignación de núcleo y espectro bajo demanda que reduce tanto la diafonía como la fragmentación en redes ópticas elásticas con MCF. Este algoritmo utiliza caminos mínimos calculados previamente con el algoritmo de Dijkstra. Para los experimentos se asume una fibra de $4THz$ con subportadoras de $12.5Ghz$ y, por lo tanto, $\bar{s} = 320$.

8.8. 2015

En febrero de 2015 [142], Tornatore et al. propusieron cuatro formualciones ILP multi-objetivo para diferentes variaciones del RSA en anillos: RSA, *RSA with grooming* (RSA-G), *RSA with regeneration* (RSA-R); y las dos juntas (RSA-RG). Estas mismas configuraciones fueron aplicadas también para los problemas RMSA, RBSA y RMBSA sobre topologías en anillo y asumiendo bandas de guarda fijas.

En mayo de 2015 [26], Chatterjee et al. escribió un *survey* acerca del RSA desde sus orígenes con una perspectiva telecomunicacional. El artículo comienza presentando los conceptos básicos, la arquitectura y el principio de operación de EON; explica los *bandwidth-variable transponders* (BVT) y los *sliceable bandwidth-variable transponders* (SBVT), discute las diferentes arquitecturas de nodos: *broadcast* y *select*, ruteo de espectro, *switch* y *select* con funcionalidad dinámica, y arquitectura bajo demanda, junto con sus funcionalidades. Luego discute también las diferencias entre el RSA y el RWA, y estudia diversos enfoques de ruteo y políticas de asignación de espectro. El trabajo asimismo explica fragmentación, modulación, calidad de transmisión, *traffic grooming*, *survivability*, ahorro de energía, y costo de red; clasifica los enfoques para evitar la fragmentación; menciona trabajos acerca del RSA que consideran técnicas de modulación para adaptarse según la distancia; discute y compara el *traffic grooming* en redes ópticas basadas en WDM, con BVTs y con SBVTs; aborda la protección y restauración; analiza la reducción de costos de redes mediante el uso de SBVT. Finalmente, los autores exploraron las demostraciones experimentales que se han publicado en 2015 para confirmar la funcionalidad de EON.

En febrero de 2015 [139], Talebi et al. estudiaron el problema del SA en redes anillo con ruteo de camino mínimo (o, más generalmente, ruteo fijo). El trabajo demuestra que, si se satisfacen todas las demandas utilizando el camino más corto, este problema puede resolverse en tiempo polinomial en pequeños anillos de tres y cuatro nodos. También se desarrollan algoritmos de aproximación de *constant-ratio* para anillos grandes de hasta 16 nodos.

En junio [74] y octubre de 2015 [71], Klinkowski et al. presentaron una formulación ILP basada en un modelo previo [146] para la versión estática del RSA, con el objetivo de minimizar el espectro total usado. Como el modelo utiliza una familia exponencial de variables que relacionan los *lightpaths* precomputados con las demandas, los autores proponen un enfoque *branch-and-cut-and-price* para resolverlo, agregando una familia de

planos de corte para mejorar la cota inferior, y un algoritmo goloso y una heurística *simulated annealing* para mejorar la cota superior. El orden en el cual los algoritmos iteran sobre la lista de demandas se basa en la heurística VSA, presentada en 2014 [3]. Los experimentos fueron realizados seleccionando $|D| \in \{20, 30, \ldots, 60\}$, con volúmenes en el rango de $[1, 32]$ *slots*, precomputando hasta diez caminos por cada demanda, así como todos los *lightpaths* posibles para cada una de ellas.

En julio de 2015 [82], Li et al. estudiaron la versión estática del *routing and wavelength core assigment problem* (RWCA) para redes MCF con *MIMO-based crosstalk suppression* y presentaron un modelo de ILP basado en nodo-enlace utilizando canales precomputados para resolverlo. El objetivo buscado fue maximizar el tráfico aceptado y optimizar la utilización de recursos.

En noviembre de 2015 [172], Zhao et al. propusieron una formulación ILP y algunas heurísticas para resolver el RMSA para tráfico estático, teniendo en cuenta tanto las degradaciones lineales como las no lineales, con el objetivo de minimizar el máximo *slot* asignado. Dado que el modelo de ILP presentado no escala, los autores también presentaron dos heurísticas, una de ellas que precalcula los k-caminos mínimos para cada demanda.

En diciembre de 2015 [45], Feng et al. presentaron una heurística para resolver el RMSA definido sobre un digrafo, buscando minimizar el mayor *slot* asignado a las demandas. Para los primeros experimentos se utilizó la topología *15n-46m-NSF*, asumiendo *subportadoras* de $12.5Ghz$, y con $|D| \in [20, 1000]$, y $v(d) \in [10Gb/s, 100Gb/s]$ para cada $d \in D$, utilizando los niveles de modulación BPSK, QPSK, 8-QAM, 16-QAM y 32-QAM. La distancia de transmisión del formato BPSK se asumió de 10.000 km, y este valor se redujo a la mitad por cada aumento del formato de modulación en un bit. Estos parámetros de instancias fueron utilizados para comparar tanto contra el algoritmo genético presentado en [50] como con el algoritmo basado en generación de columnas sobre un modelo ILP presentado en [122]. La comparación contra los algoritmos propuestos en [158], [72] y [132] fue realizada con una topología extraída de la red óptica de nivel 3 de USA (60 nodos, 142 enlaces) y la *CenturyLink Fiber Network* (102 nodos, 284 enlaces), con $\bar{s}$ y volúmenes de hasta 1000 *slots*.

8.9. 2016

En marzo de 2016 [8], Archambault et al. presentaron una formulación ILP para el RMSCA estático en redes ópticas *elastic filterless*, las cuales utilizan una arquitectura *passive broadcast-and-select* para ofrecer agilidad a la red. El modelo tiene una familia de variables que relaciona rutas con demandas, aunque también tiene variables que relacionan arcos con *slots* y demandas. Debido a la complejidad del problema, los autores propusieron asimismo dos heurísticas basadas en enfoques golosos y un algoritmo genético para obtener soluciones sub-optimas en redes más grandes. Para los experimentos, no está claro si todos los caminos candidatos fueron precomputados o solamente unos pocos de ellos. Las topologías de las instancias resueltas tanto hasta el óptimo como heurísticamente no presentan más de siete nodos. También es propuesta una cota inferior, calculada como el espectro requerido por el arco más utilizado cuando los caminos están fijos.

En marzo de 2016 [83], Li et al. propusieron un modelo ILP multiobjetivo para resolver el *RWA with shared backup path protection* (RWA-SBPP) en redes MCF considerando *inter-core crosstalk* y *MIMO-based crosstalk suppression*. La topología es interpretada como un grafo no dirigido y los objetivos buscados incluyen maximizar el número de

tráfico aceptado y minimizar el total de los recursos de *wavelength* utilizados tanto en las rutas principales como en las de recuperación.

En mayo de 2016 [137], Szcześniak et al. presentaron un algoritmo para resolver el *constrained RSA problem* (C-RSA) (de forma óptima cuando es una sola demanda) en un multigrafo dirigido, interpretando las longitudes de los arcos como sus costos. El algoritmo es una adaptación –y restricción– del algoritmo de caminos mínimos de Dijkstra teniendo en cuenta las restricciones de contigüidad y continuidad del espectro, y un límite en la longitud de las rutas. La comparación del desempeño fue realizada contra la resolución del ruteo con caminos mínimos disjuntos en arcos, y contra el algoritmo de k-caminos mínimos de Yen, sobre 50 *grafos Gabriel* de 100 nodos generados aleatoriamente, donde cada arco tiene 400 *slots*. Un grafo $G = (V, E)$ es llamado grafo Gabriel si todo par de puntos distintos $p, q \in V$ son adyacentes cuando el disco cerrado que contiene a p y a q con el vector pq como diámetro no contiene ningún otro punto además de ellos.

En junio de 2016 [73], Walkowiak et al. nuevamente utilizaron el VSA [3] con el fin de resolver el RSA con el modelo ILP presentado en [74]. En este trabajo los autores agregaron unas pocas desigualdades válidas más para mejorar el *branch-and-cut-and-price* y fueron capaces de resolver instancias con $|D|$ hasta 200, en topologías de 22 nodos, y $|D| = 150$ en una topología de 28 nodos.

La descomposición por *configuraciones* fue presentada en julio de 2016 [58] por Jaumard y Daryalal, utilizando generación de columnas y combinando dos algoritmos para resolver el problema de *pricing*, uno de los cuales es una formulación ILP basada en enlace-ruta restringida a un subconjunto de caminos precomputados y a un subconjunto de configuraciones factibles. El otro algoritmo recurre a una formulación ILP exacta basada en nodo-enlace y utilizada al final de la generación de columnas para garantizar la optimalidad del problema de *pricing*. Sin embargo, según los mismos autores, esta solución tiene dos falencias. El cálculo previo para el problema de *pricing* basado en rutas es complejo, y el problema de *pricing* basado en enlaces es ineficiente. Con este enfoque, en este trabajo fueron capaces de resolver heurísticamente (dado que se precomputa sólo un subconjunto de los caminos posibles) instancias de 180 demandas, con volúmenes de hasta 16 *slots* cada una, sobre la topología *21n-70m-SpanishTelefonica*, con $\bar{s} = 330$ *slots*.

En agosto de 2016 [112], Perelló et al. presentaron una formulación ILP para la versión estática del RMSCA haciendo uso de estimadores para el alcance de la transmisión. El modelo, entre otras, recurre a una familia exponencial de variables binarias que relaciona demandas con *lightpaths* y otra que establece si una demanda utiliza un *slot* particular en un arco. En este trabajo, todos los caminos posibles son precalculados para cada demanda, por lo que únicamente fueron capaces de resolver instancias muy pequeñas (6 nodos, 320 *slots*) hasta el óptimo. El artículo también presenta una heurística basada en *simulated annealing* capaz de resolver instancias más grandes, utilizando por ejemplo la topología *14n-46m-DT*.

En septiembre de 2016 [94], Muhammad et al. presentaron una formulación ILP multiobjectivo para el RMSCA sobre redes *programmable filterless SDM* (PF-SDM) buscando minimizar la cantidad de núcleos y el espectro utilizado. El trabajo también propone otra formulación relacionada con la asignación de núcleos y nivel de modulación. El modelo para el RMSCA recurre a una familia de variables binarias que relacionan rutas con demandas, y a una familia que compara el primer *slot* utilizado por cada par de demandas. No nos queda claro si estas formulaciones utilizan todas las rutas posibles o sólo un subconjunto de ellas pero, finalmente, las comparaciones se realizan con la heurística presentada en [8].

En septiembre de 2016 [150], Walkowiak et al. presentaron un modelo ILP para el *routing, modulation format, spatial resources and spectrum allocation problem* (RMSSA) y mostraron pequeñas modificaciones según si el espacio y el espectro son flexibles o no. Los *slots* se asumen de $6.25GHz$, se precomputa un conjunto de rutas candidatas para cada demanda, y se utiliza el enfoque basado en canales, *i.e.*, se precomputan todos los intervalos posibles de *slots* que satisfacen cada demanda en cada ruta (dependiendo del formato de modulación). El objetivo buscado es minimizar el máximo espectro, sin embargo el artículo no presenta ningún experimento.

En septiembre de 2016 [175], Zhu et al. explicaron los conceptos relacionados con los recursos: *immediate reservation* (IR) y *advance reservation* (AR), los cuales derivan en las versiones dinámica y estática del RMSA, respectivamente. Luego presentaron un algoritmo *multi-path fragmentation-aware* (MPFA) para ambas versiones de dicho problema. Cuando no hay disponible suficientes recursos de espectro para una demanda entrante, en el trabajo se propone dividir los pedidos y transferir dichos subpedidos por diversos caminos utilizando *sliceable bandwidth variable transponders*. También propone una medición bidimensional de la ocurrencia de fragmentación tanto en el dominio del espectro como del tiempo.

En septiembre de 2016 [1], Abkenar et al. presentaron un algoritmo heurístico, denominado *best fit*, para resolver el SA para el RSA semiestático, *i.e.*, Las condiciones del tráfico cambian en tiempos más largos que en la versión dinámica del problema. Este artículo no pone el foco en el ruteo, y utiliza el algoritmo Floyd-Warshal [39] para hallar las rutas candidatas. Para cada demanda d, esta heurística analiza bloques libres de *slots* intentando asignarle a d el menor de ellos que alcance para satisfacerla. Para resolver el problema combinatorio resultante se utiliza un modelo ILP. Los experimentos fueron llevados a cabo con la topología *24n-86m-UBN24*, con $\bar{s} = 1000$, y los volúmenes distribuidos uniformemente en el rango $[20, 40]$, con una banda de guarda fijada en diez *slots*. La comparación de rendimiento fue hecha contra los algoritmos *first-fit* [62] y *random-fit*.

En noviembre de 2016 [118], Rottondi et al. formularon, utilizando un modelo ILP multiobjetivo, el RMBSA para dos políticas diferentes de *switching*: *spatially flexible* y *spatially and spectrally flexible*. El objetivo es hallar la menor ocupación de espectro y el mínimo costo de los transceptores instalados. El modelo presentado asume un conjunto precomputado de rutas (que aquí denominan *lightpaths*) y canales para cada demanda (dado que el estudio es realizado únicamente para anillos, los caminos posibles son sólo dos por demanda), tiene en cuenta las bandas de guarda, fijadas en un *slot*, y tiene, entre otras, una variable que relaciona *lightpaths* con demandas. Con esta configuración, los autores fueron capaces de resolver hasta el óptimo, topologías de ocho nodos con 80 *slots* por arco.

8.10. 2017

En enero de 2017 [10], Assis et al. utilizaron tres algoritmos para resolver la versión semiestática del RSA: BSR [41], SPSR [158], y BLSA con protección [146]. La política de asignación de espectro utilizada para la fase dinámica es FF [62]. Los experimentos fueron realizados utilizando las topologías *14n-42m-NSF* y *19n-76m-EON19*, con 60 *slots* por arco y requiriendo de uno a ocho *slots* cada demanda.

En enero de 2017 [54], Hai et al. presentaron un algoritmo genético para resolver el RSA heurísticamente utilizando el procedimiento de caminos mínimos de Yen [168], bus-

cando minimizar el espectro total usado. Para los experimentos, se crearon dos pequeñas topologías, de seis y ocho nodos, con 18 y 32 arcos, respectivamente, fijando $\bar{s} = 50$, y los volúmenes de las demandas en el rango $[1, 5]$. Las comparaciones fueron realizadas con la heurística MSF presentada en [33] y una formulación ILP no descrita.

En abril de 2017 [164], Xuan et al. formularon, utilizando ILP, una versión particular del RMSA estático teniendo en cuenta un conjunto de nodos inseguros por conexión. Para resolver el problema, en el artículo se propone una implementación de un *framework* con una heuristica y un algoritmo genético, resolviendo primero la asignación de *slots* y luego la fase de ruteo. Los experimentos fueron llevados a cabo utilizando las topologías *14n-42m-NSF*, *15n-54m-CHNNET* y *20n-62m-ARPANet*, con cuatro conjuntos diferentes de instancias seleccionando la cantidad de demandas para cada una de ellas dentro del conjunto $\{250, 500, 750, 1000\}$, mientras que el volumen de cada demanda fue seleccionado en el rango $[1, 10]$.

En junio de 2017 [40], Dos Santos et al. presentaron una nueva heurística para resolver el RSA dinámico. El procedimiento, denominado *Yen-BSR-SLICE* (YBS), se basa en el algoritmo de k-caminos mínimos presentado por Yen [168] y los algoritmos presentados en [41] y [124]. El rendimiento fue comparado sobre la topología *19n-76m-EON19*, con un total de 256 *slots* disponibles por enlace, y cada demanda requiriendo de 1 a 16 *slots*.

En julio de 2017 [91], Marković aplicó el enfoque metaheurístico *bee colony optimization* (BCO) para resolver el RSA estático, buscando minimizar tanto la utilización del espectro como el promedio de las longitudes de las rutas asignadas a las demandas. Los experimentos fueron realizados sobre las topologías *14n-42m-NSF*, *19n-76m-EON19*, *28n-82m-EURO28*, y la *40n-114m-USA*.

En septiembre de 2017 [51], Goścień y Piotr presentaron un modelo de ILP basado en enlace-ruta y un algoritmo de generación de columnas para resolver la versión estática del *RMSSA with dedicated path protection* (RMSSA-DP), asumiendo que ambos caminos son disjuntos en enlaces. A pesar de que el modelo es capaz de devolver el óptimo, los autores proveen un conjunto de *lightpaths* precomputados (dos por cada demanda). Los experimentos utilizan la topología *14n-46m-GDT*.

8.11. 2018

En enero de 2018 [68], junto con la definición de los posibles escenarios SDM, Klinkowski et al. hicieron un recuento de los algoritmos de asignación de recursos que se hallan en la literatura y los clasificaron de acuerdo a diversos criterios. El artículo también presenta tres modelos ILP para resolver la versión estática del RSSA, interpretando la topología como un digrafo y minimizando la suma total de *slots* requeridos para satisfacer todas las demandas.

En febrero de 2018 [43], Enoch y Jaumard propusieron una formulación ILP basada en una descomposición de *lightpaths*, con el fin de resolver la versión estática del RSA, aplicando técnicas de generación de columnas, asumiendo un grafo no dirigido y bandas de guarda de un *slot*. Sin embargo, los autores cuentan que este modelo no funciona bien cuando las instancias del problema crecen. El modelo ILP final del proceso de generación de columnas contiene una cantidad muy grande de columnas, por lo que puede llevar mucho tiempo resolverlo. Para aliviar este problema, el artículo propone remover las columnas no básicas luego de cada optimización para el problema maestro restringido. Los experimentos fueron llevados a cabo con los mismos parámetros que los utilizados en [58], agregando la

topología *24n-86m-UBN24*, con 400 *slots* y 276 demandas con volúmenes de hasta 16 *slots* cada una.

En julio de 2018 [14], como un primer resultado de esta tesis, Bertero et al. presentamos doce formulaciones ILP compactas y naturales para resolver la versión estática del RSA comparándolas con la adaptación de los modelos presentados por Velasco et al. en 2012 [146] y 2013 [176]. Los experimentos fueron llevados a cabo sobre ocho topologías reales distintas, siendo *21n-78m-UKNet* una de las más grandes, con $|D| \in \{10, 20, 40\}$, $\bar{s} \leq 110$ y $v(d) \in [1, \bar{s}]$ para toda demanda $d \in D$.

En diciembre de 2018 [161], Wu et al. se enfocaron en la versión estática del RSA y estudiaron el impacto de la topología de red, de la distribución del tráfico y del esquema de ruteo sobre el espectro utilizado. En este trabajo el RSA fue estudiado sobre un grafo dirigido, teniendo en cuenta las bandas de guarda y buscando minimizar el máximo espectro utilizado. Como un primer resultado, el estudio muestra que el uso óptimo del espectro está positivamente correlacionado con la probabilidad de intersección de la ruta de dos demandas cualesquiera, a la vez que proporciona los límites superior e inferior del uso óptimo del espectro mediante el análisis del número cromático del grafo de conflicto de las rutas utilizadas. Asimismo, se proporciona un enfoque analítico sobre cómo conectar el número cromático del grafo de conflicto con la probabilidad de intersección, y se verifican estos resultados mediante experimentos mediante la implementación de un modelo de programación cuadrática, el cual utiliza un conjunto precomputado de k-caminos mínimos (la cantidad seleccionada para los experimentos fue de 2) para cada par de nodos. Las pruebas fueron realizadas sobre tres topologías: un anillo simple de 12 nodos, *14n-42m-NSF*, y *11n-46m-NJ-LATA*.

8.12. 2019

En septiembre de 2019 [53], un modelo ILP basado en enlace-nodo y un algoritmo *branch-and-cut* son presentados por Hadhbi et al. para una variación del RSA, considerando un límite para las longitudes de los caminos e interpretando la topología como un grafo no dirigido, *i.e.*, asumiendo trafico simétrico bidireccional, tanto en la ruta como en los intervalos asignados. El modelo propuesto utiliza tres familias de variables previamente propuestas en [14], pero de un modo diferente agregando las restricciones de distancia mencionadas. Las desigualdades propuestas como planos de corte pertenecen a dos familias exponenciales basadas únicamente en consideraciones de flujo, *i.e.*, mirando sólo a los nodos y arcos en lo que respecta al ruteo; éstas son *path-continuity* y *cycle-elimination*. Los experimentos fueron llevados a cabo en instancias medianas sobre las topologías *n6s9*, *10n-44m-SmallNet*, *14n-42m-NSF*, y *21n-70m-SpanishTelefonica*, con hasta 285 *slots* por enlace en la menor, y hasta 64 en la última. Algunos conjuntos de hasta 500 demandas, con volúmenes de hasta cuatro *slots*, fueron utilizadas para las topologías más pequeñas, y sólo 15 demandas, requiriendo también de uno a cuatro *slots* cada una, para las mayores.

En septiembre de 2019 [136], Szcześniak et al. presentaron una modificación al algoritmo de camino mínimo de Dijkstra, que permite resolver de manera eficiente las versiones dinámicas del RWA, del RSA y del RMSA optimizando cada conexión. Los experimentos fueron realizados sobre 100 grafos Gabriel generados aleatoriamente con 25 y 75 nodos, y con enlaces en el rango $[119, 145]$. Tres valores para $\bar{s}$ fueron utilizado de acuerdo al ancho de banda seleccionado para los *slots*, *i.e.*, 160, 320, y 640, correspondientes a $25Ghz$, $12.5GHz$ y $6.25Ghz$, respectivamente. El volumen de cada demanda fue seleccionado en

el rango $[1, 10]$. El artículo no presenta análisis de complejidad ni de corrección, pero la extensión de los experimentos, al comparar contra procedimientos basados en fuerza bruta y grafos filtrados, pareciera corroborar la corrección y la eficiencia del algoritmo propuesto.

En noviembre de 2019 [80], Xuhong et al. estudiaron y subclasificaron la *advance reservation* (AR) y propusieron un algoritmo heurístico que precomputa los k-caminos mínimos utilizando el procedimiento de Yen [168] para resolver la versión estática del RMSA. Los experimentos fueron llevados a cabo sobre dos topologías: *14n-42m-NSF* (*National Science Foundation Network*), y *11n-52m-Pan-European-COST239*, con 320 *slots* por arco, y volúmenes dentro del conjunto $\{2, 4, 6, \ldots, 20\}$. Los formatos de modulación empleados fueron BPSK, QPSK, 8-QAM y 16-QAM, mientras que los *benchmarks* utilizados fueron aquellos presentados en [156], [135], y [175].

En diciembre de 2019 [163], Xuan et al. estudiaron la versión dinámica del *manycast RSA* (M-RSA) en grafos no dirigidos. Este artículo presenta un modelo de optimización multiobjetivo, el cual utiliza dos variables y una gran variedad de restricciones adaptadas de [14], y busca minimizar el consumo de potencia de la red, el total del espectro ocupado, y el mayor índice de los *sltos* utilizados. Dichos objetivos son condensados en uno solo, utilizando la estrategia de suma ponderada. El artículo también propone un algoritmo *grey wolf optimization* (IGWO) y un algoritmo *differential evolution* (DE) para el mismo problema. En este trabajo se tienen en cuenta la cantidad de amplificadores ópticos de cada enlace, la cantidad de conexiones cruzadas ópticas en cada nodo, y el consumo de potencia. Los experimentos son realizados utilizando las topologías *14n-42m-NSF* y *24n-86m-UBN24*, con $\bar{s} = 1000$ y $v(d) \in [1, 10]$ para cada demanda d.

8.13. 2020

En enero de 2020 [42], Enoch presentó una formulación ILP utilizando la descomposición por *configuraciones* para modelar la versión estática del RSA sobre grafos no dirigidos buscando maximizar el tráfico aceptado. Este artículo propone un algoritmo de generación de columnas anidado, *i.e.*, resolviendo el problema mediante un algoritmo *branch-and-cut-and-price* donde, a su vez, el problema de *pricing* es resuelto aplicando técnicas de generación de columnas. Los experimentos fueron llevados a cabo sobre las topologías *21n-70m-SpanishTelefonica* y *24n-86m-UBN24*.

En febrero de 2020 [12], Behera y Das estudiaron las problemáticas de fragmentación y bloqueo para el RSA dinámico, abordando también la restricción de reasignación, y proponiendo un modelo MILP, el cual funciona sólo en pequeños escenarios, y una heurística para las instancias reales. Ambos algoritmos utilizan k-caminos mínimos precomputados entre cada par de nodos, por lo tanto las soluciones halladas por ellos no tienen garantía de optimalidad. Los experimentos fueron llevados a cabo sobre las topologías *6n-18m-n6s9* y *14n-42m-NSF*, con $\bar{s} = 20$ y $\bar{s} = 300$, respectivamente, y fijando $k = 3$.

En junio de 2020 [79], Li y Hong-jie presentaron un modelo ILP para resolver el RMSA con un *trade-off* entre el consumo de recursos (R) y el intervalo con límite (I) (*Tradeoff-RnI*). La formulación multiobjetivo utiliza una combinación lineal para condensar ambos objetivos, precomputa los k-caminos mínimos entre cada par de nodos, y recurre a dos familias de variables enteras para indicar los *slots* extremos del rango asignado a cada demanda. Los experimentos, utilizando $k = 4$ sobre las topologías *24n-86m-UBN24* y *14n-42m-NSF*, con $\bar{s} = 240$ y volúmenes de hasta 16 *slots* por demanda, mostraron que el algoritmo propuesto tiene una tasa de bloqueo de conexiones menor que los procedimientos

first-last fit y *block-assignment*.

En julio de 2020 [101], Paira et al. propusieron una heurística basada en multicaminos para resolver el RSA dinámico, con los siguientes objetivos: minimizar el consumo total de potencia reduciendo el uso de varios elementos de la red, proporcionar capacidad de supervivencia de EON contra fallas de un solo enlace utilizando un esquema de protección multitrayecto, mejorar la tasa de ocupación del espectro de la red, y minimizar la tasa de bloqueo del ancho de banda sin comprometer las conexiones de alta prioridad. Los experimentos se llevaron a cabo en las topologías *11n-52m-Pan-European-COST239* y *15n-46m-NSF*.

En octubre de 2020 [165], Xuan et al. estudiaron una versión muy particular del RMSCA estático en una EON multidominio (RMSCA-MD), *i.e.*, cada nodo de la topología se denomina *dominio* porque está compuesto de muchos subnodos. Este problema se resuelve de manera óptima en instancias pequeñas utilizando una formulación ILP basada en enlace-ruta, y también heurísticamente con un GA, con *tailor-made crossover*, mutación y operadores de búsqueda local, cuando las instancias son más grandes. Ambos algoritmos intentan minimizar el máximo índice de los *slots* utilizados. En los experimentos con la heurística se utilizaron hasta diez subnodos sobre la topología *14n-42m-NSF*, lo cual da un tamaño total de 140 nodos.

En octubre de 2020 [107], Patel et al. presentaron un algoritmo heurístico para resolver la versión estática del RMSA. El algoritmo denominado *adaptive dynamic routing* (ADRA) se basa en el procedimiento *adaptive modulation and regenerator-aware* (AMRA) [2] y utiliza un conjunto de k-caminos mínimos entre cada par de nodos. El rendimiento fue medido contra el algoritmo *shortest path first*, presentado en [115], y el mencionado AMRA, fijando $k = 10$ y utilizando las topologías *28n-82m-EURO28* y *26n-84m-US26*, con $\bar{s} = 320$.

En noviembre de 2020 [7], Araújo et al. presentaron un *survey* acerca del RSA, clasificando alrededor de 40 trabajos de acuerdo al método de optimización utilizado, a la función objetivo y al enfoque de modelado. A continuación los autores resuelven el problema C-RSA, desarrollando una formulación *multi-commodity flow* basada en ILP y un procedimiento de equilibrio de carga que genera las rutas para la formulación de ILP presentada en [146]. Para los experimentos se utilizaron 405 instancias con $\bar{s} \leq 40$ y $|D| \leq 100$ sobre nueve topologías reales de 11 a 125 nodos, extraídas de [76](*i.e.*, *Abilene*, *RedIris*, *euNetworks*, *BSO Net.Solutions*, *Ipe*, *GARR*, *GEANT*, *Bell South*, y *ION*).

En noviembre de 2020 (publicado en febrero de 2021) [78], Lezama et al. presentaron tres algoritmos metaheurísticos para resolver el RSA estático, asumiendo bandas de guarda fijas, y buscando minimizar la utilización del espectro y el promedio de las longitudes de los caminos. Estos algoritmos –basados en optimización por colonias y evolución diferencial– precalculan los k-caminos mínimos y utilizan MSF como estrategia de precómputo. Los algoritmos fueron evaluados en un *benchmark* con las redes ópticas *14n-42m-NSF*, *19n-76m-EON19*, y *40n-116m-USA*, tomando $\bar{s} \leq 200$ *slots* y $v(d) \in [1, 4]$ para cada demanda $d \in D$.

8.14. 2021

En marzo de 2021 [97], Munasinghe et al. presentaron una formulación ILP multiobjectivo para resolver el IA-RMSA estático sobre redes pequeñas, y una heurística para resolver instancias mayores. El objetivo buscado es reducir el consumo de potencia y optimizar la

eficiencia del uso del espectro, minimizando la cantidad de *slots* utilizados y apagando una gran cantidad de enlaces con poco uso. Los algoritmos propuestos precomputan k-caminos mínimos para cada demanda, dividen los arcos del digrafo en *spans* agregando amplificadores entre ellos, utilizan diversos niveles de modulación, y, en lugar de usar bandas de guarda, calculan un *signal-to-noise ratio* (SNR) para cada conexión. Este valor condensa las *power spectral densities* (PSDs) de la señal, el ruido *amplified spontaneous emission* (ASE), y el ruido proveniente de *nonlinear impairments* (NLI). La formulación ILP propuesta, al tener en cuenta tantos parámetros es muy compleja. La misma recurre a una variable que asigna un camino a cada demanda, y otra variable que relaciona arcos con demandas para forzar las restricciones de flujo. Los experimentos realizados con el modelo ILP fueron llevados a cabo sobre redes aleatorias de 6 nodos, mientras que los realizados con la heurística, sobre la topología *14n-46m-GDT*; ambos con $\bar{s}$ hasta 320 *slots*.

En marzo de 2021 [167], Yang et al. definieron el *routing, spatial channel, and spectrum allocation problem* (RSCSA) para las *spatial channel networks* (SCN), proponiendo un modelo ILP y un algoritmo heurístico para resolverlo.

En mayo de 2021 [19], Bianchetti y Marenco como parte de la presente tesis, implementamos un algoritmo *branch-and-cut* para resolver el RSA estático. En el artículo se presentan 66 familias de desigualdades e igualdades válidas y cortes optimales para el modelo ILP, denominado DSL-BF, propuesto en [14], y se demuestra la (no) implicación entre estas familias (al menos de a pares). Asimismo, se comparan cinco estrategias para la selección de planos de corte, tanto entre ellas como contra los algoritmos genéricos *branch-and-bound* y *branch-and-cut* ofrecidos por el *solver* Cplex, en más de cien instancias generadas con el *script* disponible en [16] utilizando 20 topologías reales diferentes, con $\bar{s}$ hasta 200 y el volumen de cada demanda en el rango $[1, 124]$.

En mayo de 2021 [84], Lohani et al. presentaron tres algoritmos heurísticos para resolver el RSA dinámico buscando mitigar la problematica de fragmentación. Estos algoritmos utilizan conjuntos precomputados de k-caminos mínimos para las demandas. Los experimentos fueron realizados utilizando las topologías *14n-42m-NSF* y *24n-86m-UBN24* con un ancho de banda de $4THz$ y subportadoras de $12.5GHz$, dando un total de 320 *slots* por fibra. El volumen de cada demanda fue seleccionado en el rango $[1, 16]$, la banda de guarda fue fijada en un solo *slot*, y se consideraron diez rutas por demanda, *i.e.*, $k = 10$.

En agosto de 2021 [38], Dias et al. examinaron la importancia del diseño de la capacidad de supervivencia de la red contra fallas de un solo enlace, utilizando *dedicated path protection* y esquemas de compresión del ancho de banda. Este trabajo propone una formulación ILP y un GA para resolver los diversos tipos de protección, considerando un enfoque de espectro basado en canales, y buscando minimizar el máximo espectro necesario.

En agosto de 2021 [162] (publicado en febrero de 2022), Wu et al. propusieron una formulación ILP basada en canales y dos modelos ILP para el RSSA teniendo en consideración el cambio de carril espacial.

En octubre de 2021 [113], Pérez López et al. presentaron dos metaheurísticas para resolver las versiones estática y semiestática del RSA con ancho de banda variable. Las heurísticas, basadas en el comportamiento de las abejas, precomputan los primeros k-caminos mínimos entre los nodos y usan FF para asignar el espectro a cada demanda. Los experimentos fueron llevados a cabo utilizando las topologías *Atlanta* (15 nodos, 22 enlaces), *France* (25 nodos, 45 enlaces), y *Pioro* (40 nodos, 100 enlaces), fijando $\bar{s} = 200$, el volumen para cada demanda en el rango $[1, 10]$, y la cantidad de caminos $k = 5$. Para el problema estático se utilizó $|D| = 300$.

8.15. 2022

En Abril de 2022 [35], Colares et al. presentaron una formulación compacta para el C-RSA, *i.e.*, el RSA pero donde cada ruta asignada debe satisfacer una restricción de longitud. El modelo ILP es del tipo enlace-nodo y está principalmente basado en la formulación presentada por Hadhbi et al. en [53], el cual utililza las tres familias de variables propuestas por Bertero et al. en [14] agregando las restricciones de distancia mencionadas, *i.e.*, y_{de} para relacionar demandas con enlaces, x_{ds} para decir si una demanda utiliza o no un *slot*, y u_{des} para relacionar una demanda con un enlace y un *slot*. La diferencia radica en que en lugar de interpretar la topología como un grafo no dirigido *i.e.*, asumiendo tráfico simétrico bidireccional, tanto en la ruta como en los intervalos asignados, aquí generan un grafo dirigido a partir del original. Asimismo reescriben las tres familias de variables mencionadas con una nueva familia que es similar a la l_{des} propuesta en [14], pero relacionando cada enlace y demanda con el último *slot* utilizado por ésta en lugar de con el primero. Esta variable es la denominada r_{des} en el Apéndice 7. El modelo original se modifica también eliminando algunas variables ahora innecesarias y reemplazando restricciones con las clásicas desigualdades de conservación de flujo disgregadas por slot, entre otras. Asimismo se presenta una familia de desigualdades válidas *disaggregated length inequalities*. Para los experimentos se utilizan hasta 60 demandas en las topologías *5n-14m-Spain* con $\bar{s} = 30$, *6n-18m-n6s9* con $\bar{s} = 120$, y *17n-50m-German* con $\bar{s} = 140$. En la serie de ejecuciones, donde el tiempo límite para cada ejecución fue fijado en $2hs$, la nueva formulación resulta hasta 4742 veces más rápida que la anterior para una de las cuatro funciones objetivo probadas.

GA	Genetic Algorithm.
C-RSA	Constrained RSA.
EON	Elastic Optical Network.
FS	Frecuency Slot.
IA-RMSA	Impairment-Aware RMSA (Variable guard-bands).
ILP	Integer Linear Programming.
LP	Linear Programming.
MCF	Multi-Core Fiber.
MILP	Mixed Integer Linear Programming.
MINLP	Mixed Integer Non-Linear Programming.
MMF	Multi-mode fiber.
M-RSA	Manycast RSA.
OC	Optical chanel.
OFDM	Orthogonal Frequency-Division Multiplexing.
PF-SDM	Programmable Filterless SDM.
RMBSA	Routing, Modulation format, Baud-rate, Mode y Spectrum Allocation.
RMSA	Routing, Modulation y Spectrum Allocation.
RMSA-DP	RMSA con Dedicated Path protection.
RMSA-G	RMSA con Grooming.
RMSA-IN	RMSA con Insecure Nodes.
RMSA-R	RMSA con Regeneration.
RMSA-G	RMSA con Grooming.
RMSA-RG	RMSA con Regeneration y Grooming.
RMSA-SP	RMSA con Shared Path protection.
RMSCA	Routing, Modulation format, Spectrum, y Core Allocation.
RMSCA-MD	RMSCA in Multi Domain networks.
RMSCA-PF	Programmable F-RMSCA.
RMSCA-F	Filterless RMSCA.
RMSSA	Routing, Modulation format, Spatial resources y Spectrum Allocation.
RMSSA-DP	RMSSA con Dedicated Path protection.
RSA	Routing y Spectrum Assigment.
RSA-DP	RSA con Dedicated Path protection.
RSA-DP-SC	RSA-DP using the Same Channel.
RSA-G	RSA con Grooming.
RSA/JAU	Joint Anycast y Unicast RSA.
RSA-MP	RSA con Multi Path protection.
RSA-R	RSA con Regeneration.
RSA-RG	RSA con Regeneration y Grooming.
RSA-SP	RSA con Shared Path protection.
RSCA	Routing, Spectrum y Core Allocation.
RSCSA	Routing, Spatial Channel, y Spectrum Allocation.
RSSA	Routing, Spatial resources y Spectrum Allocation.
RWA	Routing y Wavelength Allocation.
RWA-SP	RWA con Shared Path protection.
RWCA	Routing, wavelength, y core allocation.
RWSA	Routing, Wavelength Allocation, y Spectrum Allocation.
SA	Spectrum Allocation.
SCh	Super-channel.
SCRSA	Spectrum Constrained RSA.
SDM	Space division multiplexing.
SMF	Single-mode fiber.
SSA	Spatial mode, y spectrum allocation.
WDM	Wavelength division multiplexing.

Tab. 8.1: Glosario.

9

Apéndice: Generador de instancias

En este apéndice explicamos el *script* desarrollado con el fin de generar las instancias utilizadas para los experimentos de la presente tesis. El código de dicho generador se encuentra disponible en [16].

9.1. Datos reales

Dado que el RSA es un problema que aparece en las redes de fibra óptica, seleccionamos 20 topologías utilizadas en la literatura para resolver diversos problemas relacionados con las telecomunicaciones, siendo la mayoría de ellas redes de fibra óptica reales. Sus parámetros están resumidos en la Tabla 9.1, y sus grafos asociados se muestran en la Figura 9.1. Una particularidad de este tipo de redes es que la mayoría de ellas son representables mediante grafos planares fuertemente conexos. Este hecho cobra sentido si se piensa que dicha estructura es más tolerante a fallas.

Para obtener una instancia del RSA, además de la topología, es necesario definir la cantidad máxima de *slots* disponibles por arco y el conjunto de demandas a satisfacer; este último dado como la cantidad de *slots* requerida por cada una de estas demandas, junto con sus nodos de origen y destino. De la literatura hemos obtenido algunos datos reales utilizados en los trabajos que se estudia el RSA o alguna de sus variantes:

- El ancho de banda de cada *slot*, en la mayoría de los casos, es de $12.5GHz$ de acuerdo con la recomendación de la UTI [57]. Sin embargo, ésta podría ser un poco menor (algunas veces de $6.5GHz$) o incluso mayor ($25Ghz$), aunque no demasiado, dado que para el RWA con WDM, el ancho de bánda mínimo de un *wavelength* es de $50GHz$, y el principal objetivo del RSA es mejorar dicha granularidad.
- El ancho de banda total de una fibra óptica utilizada es en promedio de $4800GHz$, (a pesar de que el máximo ancho de banda teórico está alrededor de los $231THz$).
- Podemos tener hasta 3200 *slots* en un espectro de $5THz$ utilizando *slots* de $6.25GHz$ [172].

- En los experimentos de los trabajos analizados, la capacidad $\bar{s}$ de los enlaces normalmente se fija en el rango $[170, 320]$.
- El número de demandas $|D|$ se selecciona comúnmente dentro del intervalo $[10, 100]$, excepto para algunos casos aislados que utilizan 552 [70], o incluso 1000 [45]. Sin embargo estos valores tan grandes suelen ser utilizados para experimentar con heurísticas y nunca con algoritmos exactos.
- El volumen requerido por cada demanda suele seleccionarse dentro del rango $[1, 20]$, aunque también puede encontrarse dependiente del valor asignado a $\bar{s}$.

Para más información acerca de las instancias utilizadas en la literatura, se puede consultar el *survey* presentado en el Apéndice 8.

Topología	Nodos	Arcos	Δ	$\text{mín}_{v \in V}\, \delta(v)$	$\text{máx}_{v \in V}\, \delta(v)$
6n-9m-n6s9	6	18	0.60	4	8
10n-42m-SmallNet	10	42	0.47	4	12
10n-44m-SmallNet	10	44	0.49	6	12
11n-52m-COST239	11	52	0.47	8	12
14n-42m-NSF	14	42	0.23	4	8
14n-46m-DT	14	46	0.25	4	12
15n-46m-NSF	15	46	0.22	4	8
16n-46m-EURO	16	46	0.19	4	8
19n-76m-EON19	19	76	0.22	4	12
20n-62m-ARPANet	20	62	0.16	6	8
20n-78m-EON20	20	78	0.21	4	14
21n-70m-Spain	21	70	0.17	4	8
21n-72m-Italian	21	72	0.17	4	12
21n-78m-UKNet	21	78	0.19	4	14
22n-70m-BT	22	70	0.15	6	8
24n-86m-UBN24	24	86	0.16	4	10
28n-68m-EON	28	68	0.09	4	8
28n-82m-EURO28	28	82	0.11	4	10
30n-112m-Spain	30	112	0.13	6	10
43n-176m-Euro	43	176	0.10	4	12

Tab. 9.1: Lista de las topologías utilizadas con el número de nodos, arcos, grado máximo y mínimo, y *Arcs-density* $\Delta(G)$ (definida en (**??**)).

9.2. Generación de las instancias

Como se menciona en la Sección 4.3.1, el generador utiliza dos definiciones distintas del término *density*, a saber,

1. **Arcs-density**: cantidad de enlaces de la topología sobre el total presentado por un grafo dirigido completo.
2. **Demands-Density**: Relación entre el máximo volumen requerido por las demandas y la capacidad de los arcos.

Para caluclar la *arcs-density* para cada topología, interpretamos la misma como un grafo dirigido $G = (V, E)$, que es la forma natural [14, 33, 36, 72] a pesar de que hay

varios trabajos [22, 42, 53, 73, 83, 165, 172] que asumen demandas simétricas utilizando el mismo espectro, por lo tanto simplificando la cantidad de enlaces a la mitad utilizando grafos no dirigidos.

En orden a generar cada instancia, el *script*, además del $\bar{s}$ y la topología $G = (V, E)$, recibe un parámetro real $p \in (0, 1]$ utilizado para calcular

$$maxSD = p \times \bar{s},$$

de modo tal que cada demanda utiliza un número aleatorio en el rango $[\frac{1}{2}maxSD,\ maxSD]$, y un factor opcional F para limitar el número de demandas, es decir,

$$|D| = \frac{2 \times F \times (|V| - 1) \times \Delta(G) \times \bar{s}}{maxSD},$$

con $\Delta(G)$ la *arc-density* del grafo G y $F = 1$ por defecto.

El origen y el destino de cada demanda se seleccionan uniformemente sobre los pares de nodos del grafo. Permitimos múltiples demandas para el mismo par origen-destino dependiendo de un parámetro pasado por el usuario.

9.3. El *script*

En esta sección presentamos las características del programa desarrollado así como el modo de ejecutarlo, sus *scripts*, directorios, parámetros y sus archivos de entrada y salida.

El *script instances_generator.py* lee las topologías almacenadas en el directorio *topologies/* y genera un conjunto de archivos de instancia para el RSA para cada topología basado en los datos mencionados anteriormente.

El formato de los datos es comentado en el encabezado de cada archivo. El separador utilizado es la tabulación, y las líneas que comienzan con el caracter # son interpretadas como comentarios.

El archivo de la topología es precedido por el número de nodos y el número de enlaces, seguido de la lista de estos últimos uno por línea, *i.e.*,

```
# Comment
|N|     |M|
<node i>    <node j>
<node k>    <node l>
...
```

La mayoría de las instancias pertenecen a redes con capacidad y hemos podido obtener esa información. En esos casos, el peso de cada enlace es agregado al definirlo, *i.e.*,

```
<node i>    <node j>    <weight ij>
```

A pesar de que el problema se define sobre grafos dirigidos, dado el modo en el que las redes son instaladas, asumimos que todos los enlaces tienen ambas direcciones.

El archivo de instancia también comienza con un encabezado que explica brevemente el formato. Allí se muestra la versión de este programa y la semilla utilizada. La cantidad de *slots* disponibles para cada arco y la cantidad de demandas solicitadas se muestran a continuación, seguidas de la lista de demandas.

```
# Comment
S     |D|
<src d1>    <dst d1>      <n of slots required by d1>
<src d2>    <dst d2>      <n of slots required by d2>
...
```

9.3.1. Uso

Los requerimientos se guardan en *requirements.txt*, y pueden ser instalados por medio del comando:

```
pip install -r requirements.txt
```

El siguiente comando ejecuta el *script* generando las instancias:

```
python instances_generator.py
```

Por defecto, las instancias serán ubicadas en una carpeta denominada *instances* en el directorio del *script*, en subcarpetas cuyos nombres dependen del máximo porcentaje de *slots* utilizado por las demandas y la topología. Cada uno de estos archivos con su topología asociada es la entrada del RSA.

El siguiente comando muestra como configurar los parámetros:

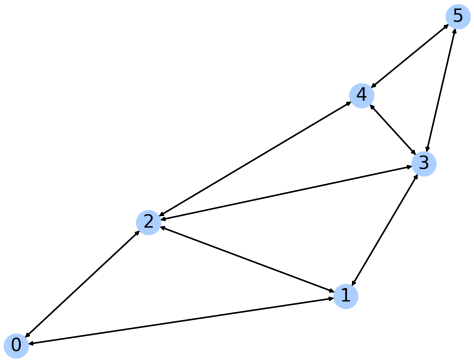
```
python instances_generator.py -h

optional arguments:
-h, --help            show this help message and exit
-mdir MDIR            The main directory or path. If no tdir or idir
                      parameters are used, mdir must contain the
                      'topologies' and/or 'instances' folder. The
                      default value is the location of this script
-tdir TDIR            The topologies directory or path.
-idir IDIR            The directory or path for the created instances.
-s SEED, --seed SEED  The random seed. Default is 1988.
-S SLOTS [SLOTS ...], --slots SLOTS [SLOTS ...]
                      List of amounts of available slots.
-p PERCENTS [PERCENTS ...], --percents PERCENTS [PERCENTS ...]
                      List of maximum percentage of total available
                      slots that a demand can use. Must be in (0, 1].
-d DENSITY, --density DENSITY
                      Density factor. The maximum amount of demands
                      is multiplied by this factor. Default is 1.0.
-m, --multiple        If set, multiple demands between a source and a
                      destination are allowed.
```

Por ejemplo, la siguiente línea generará instancias para $\bar{s} \in \{10, 25\}$ y $p \in \{10\,\%, 20\,\%, 30\,\%, 50\,\%\}$. Éstas serán guardadas en *./instances/*.

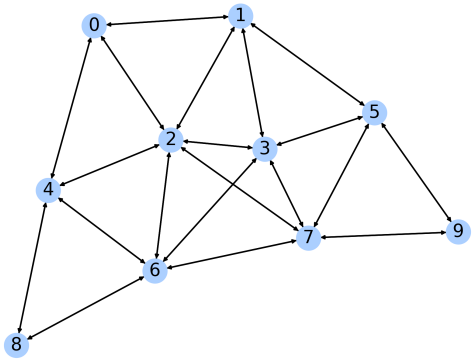
```
python instances_generator.py -S 10 25 -p .1 .3 .5 .2
```

Este comando producirá instancias con $\bar{s} \in \{10, 15, 20, 30, 40, 60, 80, 100, 150, 200, 300, 400, 600, 800, 1000\}$ y $p \in \{10\,\%, 20\,\%, 30\,\%, \ldots, 90\,\%\}$ en el directorio */home/instances*, permitiendo múltiples demandas entre cada par *(src, dst)*.

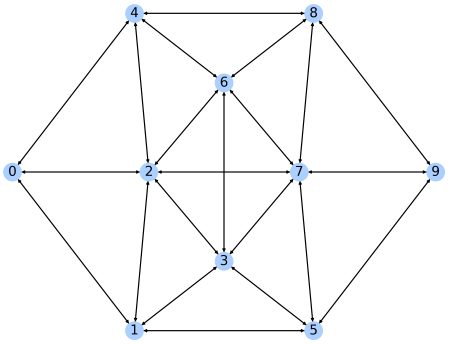
```
python instances_generator.py -idir /home/instances -m
```



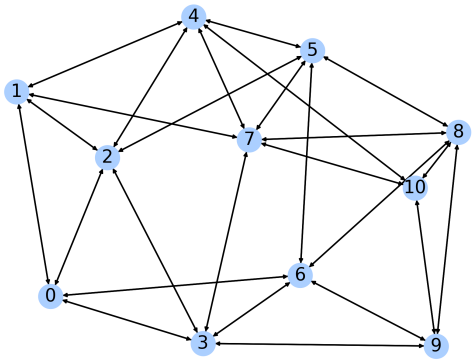
(a) 6n-18m-n6s9



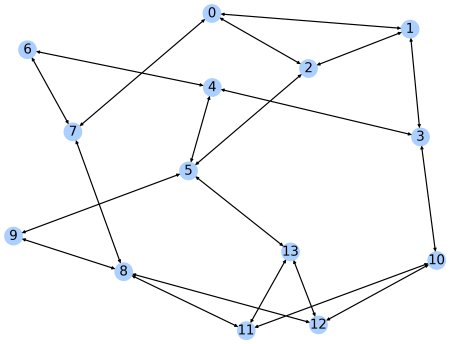
(b) 10n-42m-SmallNet



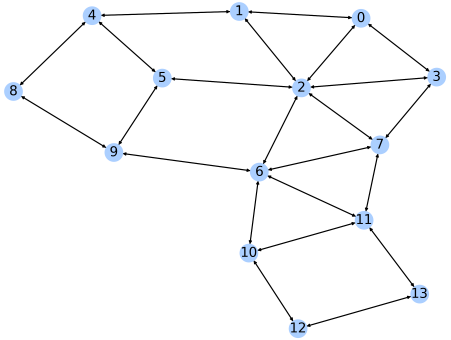
(c) 10n-44m-SmallNet



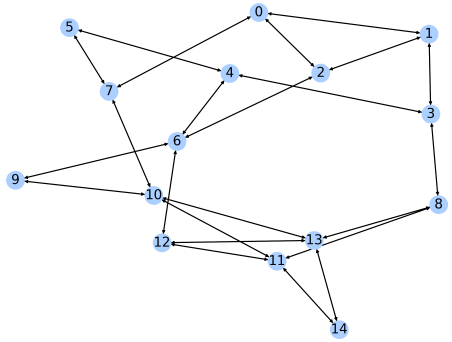
(d) 11n-52m-Pan-European-COST239 ó 11n-52m-COST239



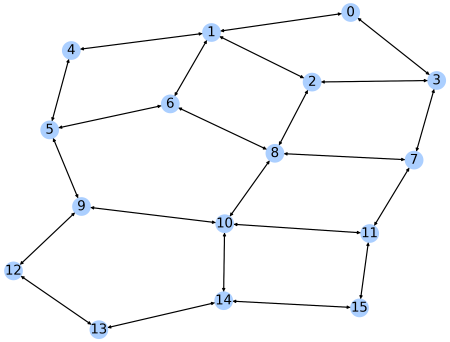
(e) 14n-42m-NSF



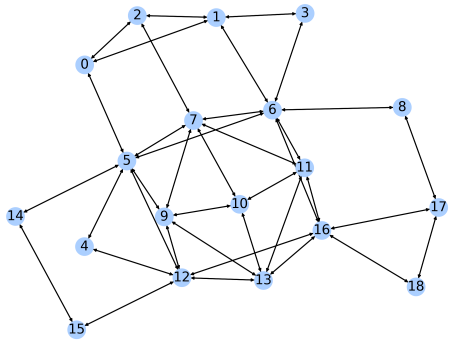
(f) 14n-46m-Generic-Deutsche-Telekom-DT ó 14n-46m-DT



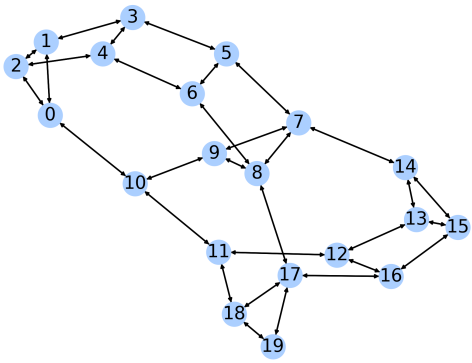
(g) 15n-46m-NSF



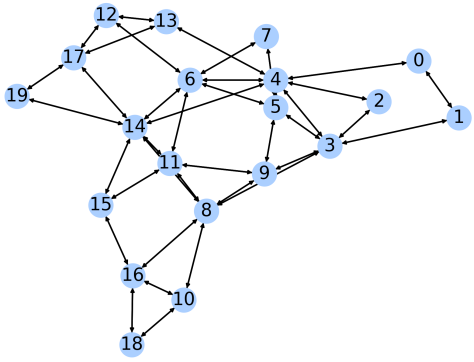
(h) 16n-46m-EURO



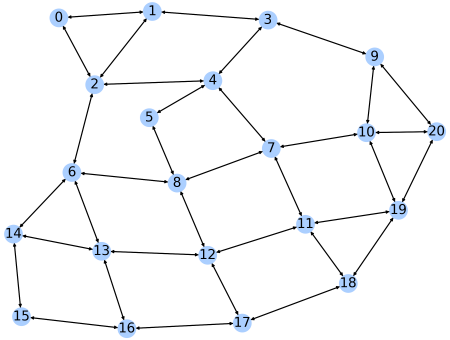
(i) European optical network: 19n-76m-EON19



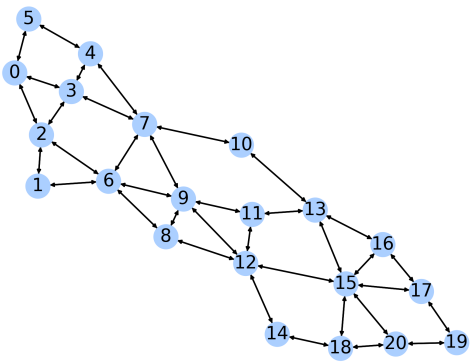
(j) 20n-62m-ARPANet



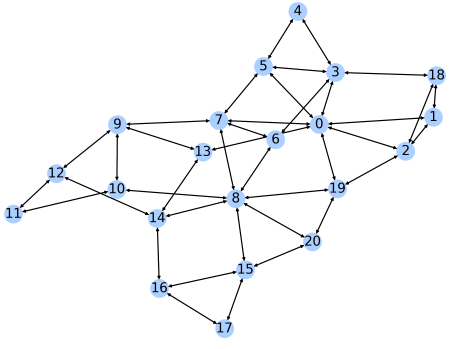
(k) 20n-78m-EON20



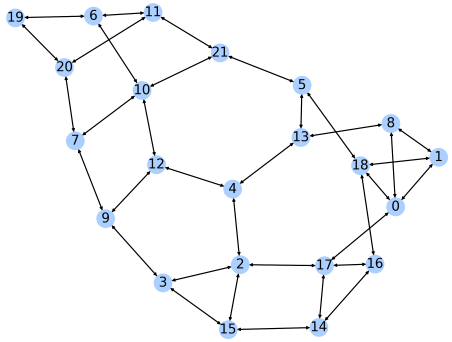
(l) 21n-70m-SpanishTelefonica ó 21n-70m-Spain



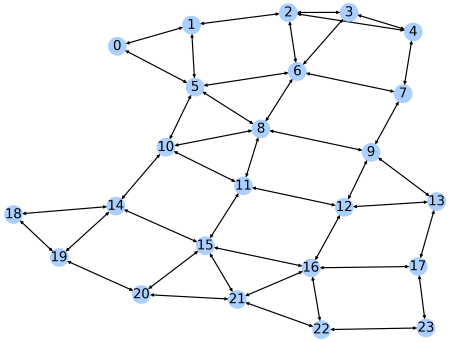
(m) 21n-72m-Italian



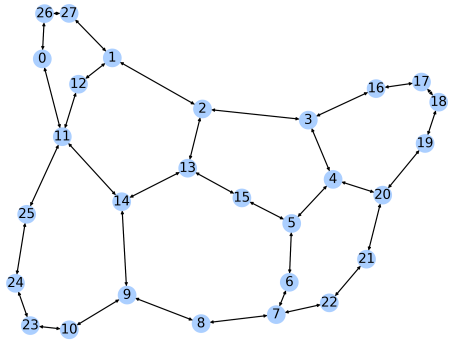
(n) 21n-78m-UKNet



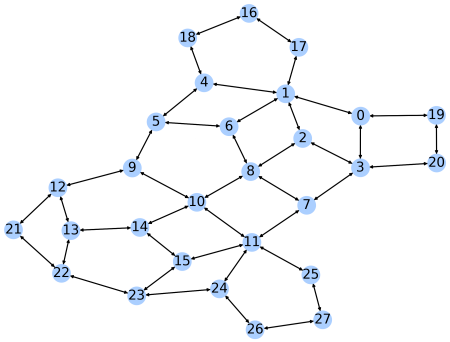
(ñ) 22n-70m-British-telecom ó 22n-70m-BT



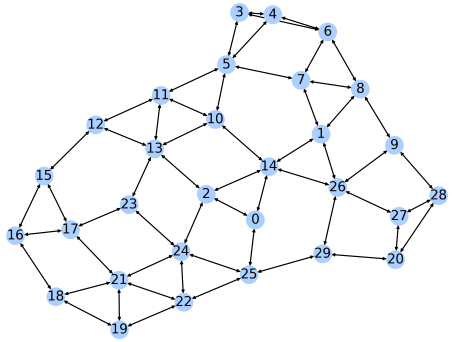
(o) 24n-86m-UBN24



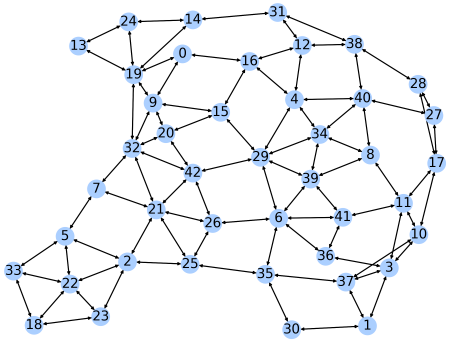
(p) 28n-68m-EON



(q) 28n-82m-EURO28



(r) 30n-112m-Spain



(s) 43n-176m-EuroLarge ó 43n-176m-Euro

Fig. 9.1: Las 20 topologías tomadas de la literatura.