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

Modelos y algoritmos para diseño de redes de comunicaciones con requisitos de supervivencia

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

Emanuel Delgadillo

Directora: Irene Loiseau

Buenos Aires, Febrero de 2013

MODELOS Y ALGORITMOS PARA DISEÑO DE REDES DE COMUNICACIONES CON REQUISITOS DE SUPERVIVENCIA

Una red de comunicaciones se dice superviviente si puede continuar brindando servicio pese a la falla de alguno de sus componentes. Hacia fines de los noventa surgió el concepto de p-ciclos como alternativa prometedora a las tecnologías existentes (malla y anillos) para el diseño de redes supervivientes. Las topologías basadas en p-ciclos combinan las mejores características de las mismas para asegurar la supervivencia: velocidad en la recuperación y poca capacidad redundante o sea menor costo. A partir de la necesidad de diseñar redes basadas en p-ciclos de costo mínimo, surgieron varios problemas de optimización combinatoria. Uno de ellos es el *Spare Capacity Allocation problem* (SCA). En este problema se tiene una red con demandas asociadas a cada enlace. Se debe determinar la disposición de la capacidad redundante mediante la ubicación de p-ciclos de forma que quede garantizada la recuperación de las comunicaciones ante la falla de alguno de sus enlaces. Un modelo simple de PLE para resolver este problema, requiere la enumeración de todos los ciclos del grafo para construir una solución óptima, lo cual puede hacer el problema intratable. Entonces a partir de este modelo se diseñaron heurísticas basadas en elegir *a priori* algunos ciclos prometedores. Por otro lado algunos autores propusieron modelos de PLE que no requieren la enumeración de los ciclos del grafo. Nosotros introducimos un nuevo modelo de programación entera para SCA que tampoco requiere la enumeración *a priori* de ciclos candidatos. Los resultados obtenidos luego de la experimentación muestran un mejor desempeño respecto a los modelos previos. También presentamos una heurística para SCA sin enumeración de ciclos basada en este modelo y en una heurística golosa propuesta por otros autores.

Palabras claves: redes supervivientes, p-ciclos, optimización combinatoria, modelos de programación lineal entera, heurísticas

MODELS AND ALGORITHMS FOR COMMUNICATION NETWORK DESIGN WITH SURVIVABILITY REQUIREMENTS

A telecommunication network is said survivable if it is still able to provide service after any of its components fails. In the late 90's the concept of p-cycles appeared as a promising alternative to previous technologies (mesh and ring). P-cycle based networks gather their best characteristics to achieve survivability: restoration speed and efficient use of spare capacity. Due to the need to minimize costs of p-cycles based networks, several combinatorial optimization problems were introduced. The Spare Capacity Allocation Problem (SCA) is one of these. In this problem, there is a network with a discrete demand for each link. Spare capacity location has to be determined by means of placing p-cycles over the network, so that full restoration is possible in case of any link fails. A simple ILP model to solve this problem requires explicit enumeration of all simple cycles of the graph, turning this problem intractable. Based on this model, several heuristics were designed to select *a priori* candidate cycles. On the other hand, some authors have proposed ILP models that do not require enumeration of graph cycles. Here we propose a new integer linear programming model that does not require *a priori* candidate cycle enumeration to solve SCA. We achieved better results than previous models. We also present an heuristic without cycle enumeration for SCA, based on both this model and a greedy heuristic proposed by other authors.

Keywords: survivable networks, p-cycles, combinatorial optimization, integer linear programming models, heuristics

Índice general

1. INTRODUCCIÓN

La industria de las telecomunicaciones ha sido desde siempre fuente de problemas de optimización. Y con la introducción de la fibra óptica aparecieron numerosos nuevos problemas, resultantes de modelar el diseño y el ruteo en estas nuevas redes de comunicaciones. Muchos de estos problemas son muy difíciles de resolver y además constituyen por sí mismos problemas de optimización combinatoria y de algoritmos en grafos interesantes de estudiar *per se*, a pesar de posibles cambios en la tecnología. Uno de ellos es el de la supervivencia de una red de comunicaciones. Una red se dice superviviente (*survivable*) si cuando una de sus componentes falla (nodo o enlace) todavía permite que las comunicaciones entre los nodos que conecta se realicen, o sea la red tiene una forma de redirigir el tráfico por un camino alternativo.

Para lograr supervivencia de una red es necesario contar con recursos redundantes. En particular, algo muy importante es tener capacidad extra, es decir que los enlaces que unen nodos de la red permitan el paso de más tráfico que el habitual en caso ser necesario debido a un fallo en algún otro sitio de la red. De manera que se debe realizar un diseño previo para administrar esta capacidad extra destinada a brindar supervivencia. Durante mucho tiempo antes del surgimiento de los p-ciclos, los dos enfoques empleados para esto fueron inicialmente usando la topología de malla (*mesh*) y posteriormente usando la tecnología de anillos (*ring*).

1.1. *Mesh* vs *Ring*

Existen distintas tecnologías que implementan la recuperación en la topología de malla (ver por ejemplo [5]). Pero el concepto en todas estas tecnologías es el mismo. Y lo mismo ocurre con anillos: diversas tecnologías con el mismo concepto [12].

Usando la recuperación en una red de malla, la capacidad redundante está distribuida por la totalidad de los enlaces de la red y será utilizada en cuanto ocurra alguna falla en uno de ellos. En dicho caso, luego de detectar la falla, el sistema encargado de la recuperación (que podría ser centralizado o distribuido) deberá tomar las acciones necesarias para redirigir el tráfico que debía utilizar la capacidad del enlace caído. Lo importante a destacar es que por cada falla habrá un problema de ruteo distinto y se deberá poner mucho esfuerzo (tiempo) para redirigir todo el tráfico ya que la capacidad redundante con la que se cuenta no tiene ninguna configuración particular predefinida.

Algo muy distinto ocurre con la recuperación usando anillos. En este caso, cada enlace está incluido en algún anillo formado dentro de la red (que puede pensarse como un ciclo simple) de manera que los dos nodos que une tienen como mínimo dos caminos alternativos: el enlace y el resto del anillo. Ante la falla de un enlace la acción que debe tomarse es muy simple: solo se debe enviar el tráfico por el sentido opuesto en el anillo, lo cual es muy importante ya que con esto es posible utilizar un mecanismo muy simple para lograr rápidamente la recuperación del servicio. En contrapartida, se requiere al menos de un 100 % de redundancia. Esto hace que no sean eficientes en el uso de capacidad.

La recuperación de la comunicación en una topología de malla es más eficiente en este sentido porque cada unidad de capacidad redundante puede ser reutilizada en más caminos en la totalidad de la red, dependiendo de la falla ocurrida. (En un anillo se protegen sólo

los enlaces que forman parte del anillo). Una red de estas características requiere solamente entre un 50 % y 70 % de redundancia para una protección total dependiendo de la topología de la red (contra el 100 % que requieren los anillos como mínimo, pudiendo ser este número mucho mayor en la práctica). Sin embargo, debido a que para lograr la recuperación se necesita resolver un nuevo problema de ruteo, la recuperación por malla no puede ser tan rápida como con anillos, además de que los dispositivos utilizados en los nodos para redirigir los datos deben ser claramente más complejos y costosos.

Por lo tanto, cada uno de estos enfoques tiene ventajas y desventajas claramente diferenciadas. La recuperación por anillos ofrece los menores tiempos de recuperación, con los más simples mecanismos de reconfiguración y equipamiento de bajo costo. La alternativa de recuperación por malla ofrece una eficiencia mucho mayor (menor redundancia) pero pagando el costo de equipamiento más costoso y resignando la rápida recuperación que proveen los anillos. En la búsqueda de un nuevo enfoque que combine la velocidad de los anillos y la eficiencia de malla surgió el concepto de *p-ciclos*.

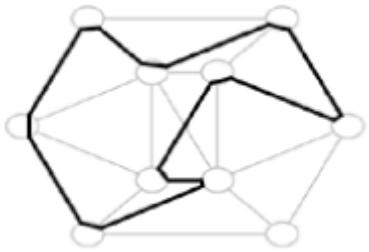


Fig. 1.1: Un *p*-ciclo

1.2. P-ciclos

Los p-ciclos (*preconfigured protection cycles*, Figura 1.1) fueron introducidos en [11] como una alternativa muy prometedora para el diseño de redes con requerimientos de *supervivencia*. Son interesantes porque combinan la simple y rápida conmutación de los anillos y la eficiencia en el uso de capacidad redundante de malla. El método de los p-ciclos se basa en la formación de caminos cerrados formados solamente por la capacidad redundante de la red destinada a su recuperación ante fallas. (Los p-ciclos son ciclos simples en términos de teoría de grafos). Cada p-ciclo brinda una unidad de capacidad de protección y la acción de reconfiguración necesaria ante cualquier falla está determinada de antemano y es similar a la utilizada con anillos, logrando así una recuperación en tiempos similares.

La diferencia con los anillos es que cada p-ciclo protege además de los enlaces que lo forman (protección *on-span*, o también llamada *on-cycle*), a los enlaces que unen nodos del ciclo que pero que no son parte del mismo (protección *straddle*). Esta mínima diferencia tiene un gran impacto en el uso de capacidad redundante y permite que se logre un uso eficiente comparable a la recuperación por malla. Para la protección *on-span* el comportamiento es igual al de anillos, ya que el p-ciclo provee **un** camino alternativo ante la falla, como se ve en la Figura 1.2. Pero para la protección *straddle*, debido a que el ciclo permanece intacto (Figura 1.3), brinda **dos** caminos alternativos sin requerir capacidad extra.

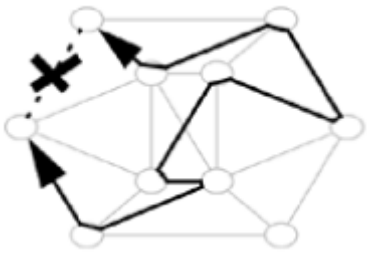


Fig. 1.2: Falla un enlace en el ciclo. El p-ciclo provee un camino alternativo.

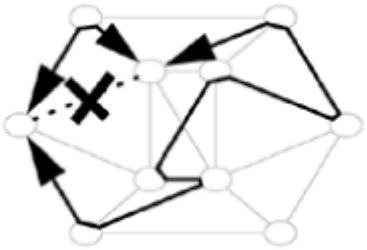


Fig. 1.3: Falla un enlace *straddle*. El p-ciclo provee dos caminos alternativos.

Desde su introducción en 1998 se presentaron distintas formas de implementarlos en redes de comunicaciones reales, fundamentalmente en redes de fibra ópticas (ver por ejemplo [5, Cap. 10: p-Cycles], [12], [1]). También fueron planteados varios problemas de optimización combinatoria para el **diseño** de redes supervivientes con p-ciclos, como por ejemplo el problema SCA sobre el que trata el presente trabajo.

1.3. Objetivos y organización de este trabajo

Diseñar redes que puedan sobrevivir a cualquier número de fallas de enlaces sería demasiado caro. De modo que en este tipo de problemas se tratará solo el caso en que se diseña la red para poder recuperarse en cada momento ante una sola falla (en particular para el caso en que falla un enlace). También se debe tener en cuenta que se considera una topología 2-conexa. Esto es cuando hay al menos dos rutas que no tienen enlaces en común entre cada par de nodos.

En este trabajo nos concentraremos en el problema SCA (*Spare Capacity Allocation*) para el diseño de la red superviviente con p-ciclos. En este problema se tiene una red con demandas discretas asociadas a cada enlace. Estas demandas deberán poder ser redirigidas por la capacidad redundante de la red en caso de que falle algún enlace. La capacidad extra (discreta) estará configurada formando p-ciclos y cada unidad de capacidad adicional tiene un costo que depende de cada enlace. En la práctica este costo está relacionado con la distancia física del enlace. Cada p-ciclo aporta una unidad de protección a los enlaces **on-span** y dos unidades a los enlaces **straddle**. Se pueden colocar varias copias de cada p-ciclo de ser necesario, las cuales estarían protegiendo el mismo subconjunto de enlaces. El objetivo es encontrar un conjunto de p-ciclos de costo mínimo que proteja completamente todos los enlaces de la red (sus demandas asociadas). El costo de cada p-ciclo estará determinado por la suma de los costos de poner una unidad de capacidad

extra en cada uno de sus enlaces.

Se puede encontrar este problema con otros nombres en la literatura: SCO por *Spare capacity optimization problem*, y también SCP por *Spare capacity placement problem*. En el trabajo de Schupke [21] se muestra que este problema es NP-Hard por medio de una reducción del problema del ciclo hamiltoniano.

Como se verá más adelante, muchas heurísticas propuestas tienen en común que toman en cuenta algún conjunto de ciclos candidatos para construir una solución. Esto tiene como desventaja que se hace muy difícil la selección de un conjunto adecuado de ciclos candidatos ya que el número total de ciclos aumenta de forma exponencial a medida que crece la red, además de que se limita anticipadamente la calidad de la solución. De modo que tendremos como principal objetivo atacar el problema desde otra perspectiva, definiendo un modelo de programación entera que no considera *a priori* un conjunto de ciclos candidatos. Los experimentos computacionales se realizaron utilizando una de las herramientas comerciales más efectivas disponible.

En el Capítulo 2 presentaremos el problema formalmente y mencionaremos algunas variantes. En el Capítulo 3 haremos una revisión de los trabajos realizados hasta el momento que pueden ser encontrados en la literatura haciendo énfasis en algunos en particular que se relacionan con el presente trabajo. En el Capítulo 4 presentaremos un algoritmo basado en la metaheurística GRASP que fue por donde comenzamos a abordar la resolución del problema SCA. En el Capítulo 5 presentaremos un nuevo modelo de programación lineal entera mixta para solucionar este problema, y presentaremos mejoras al modelo para obtener mejores resultados computacionales. En el Capítulo 6 expondremos una nueva heurística basada en el modelo anterior para el mismo problema. En el Capítulo 7 presentaremos los resultados computacionales obtenidos en el trabajo. Finalmente en el Capítulo 8 enumeraremos las conclusiones y posible trabajo futuro.

2. DESCRIPCIÓN DEL PROBLEMA

2.1. *Spare Capacity Allocation Problem*

El problema SCA consiste en proteger una red ante una falla simple de uno de sus enlaces (cualquiera de ellos, pero uno solo al mismo tiempo). Por cada enlace existe una demanda discreta en unidades de capacidad (también puede llamarse número de canales). Se dejan fuera de consideración otras complicaciones adicionales de las redes ópticas. Por ejemplo, se asume que se trabaja con una red que permite un intercambio de todos los canales en cada nodo y que se cuenta siempre con suficientes puntos de conversión de longitud de onda.

Además, es necesario que entre cada par de nodos de la red existan al menos dos caminos disjuntos, ya que de otro modo no sería posible la recuperación ante la falla de cualquiera de sus enlaces.

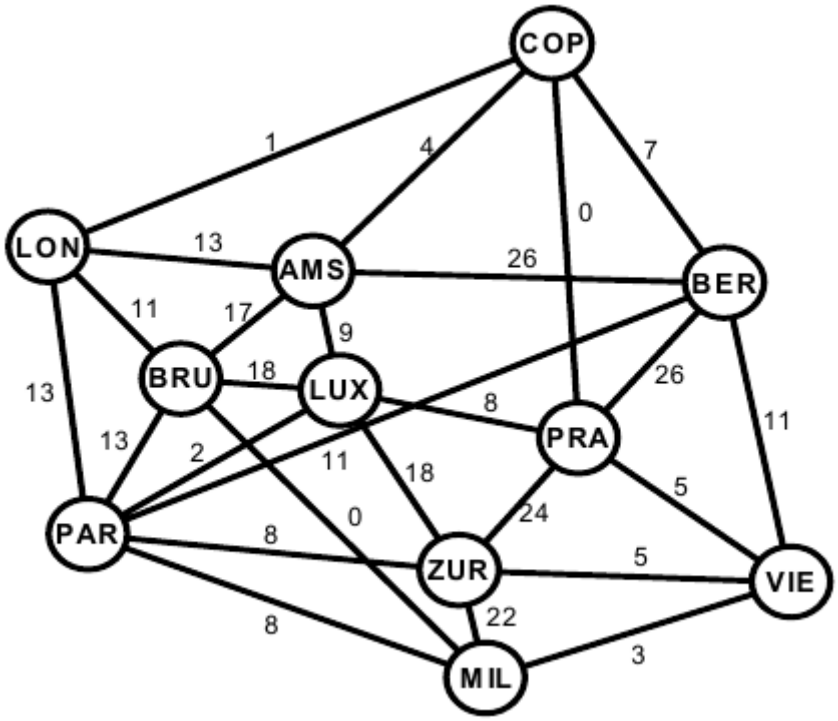


Fig. 2.1: Instancia COST239 con demandas en sus ejes.

Entonces se puede ver la red como un grafo 2-conexo en el que los enlaces están representados por el conjunto de ejes E y una solución del problema será un subconjunto de ciclos del conjunto P, que incluye los ciclos del grafo que pueden ser elegidos (todos los ciclos simples). Los ciclos que forman parte de la solución se pueden repetir si es necesario. En otras palabras, la solución puede incluir varias "copias" del mismo p-ciclo.

Cada ciclo de la solución aporta:

- 1 unidad de capacidad de protección a cada eje del ciclo (*on-span*).

- 2 unidades de capacidad de protección a cada eje que no está en el ciclo pero sus nodos incidentes sí lo están (*straddling*).

Una instancia del problema es un grafo $G = (V, E)$ 2-conexo para el cual también se tienen los siguientes datos de entrada:

d_i Es la demanda en unidades de capacidad (valor entero) para cada eje $i \in E$.

c_i Es el costo de cada unidad de capacidad adicional para cada eje $i \in E$ (siempre positivo).

Además, dado un ciclo definimos los siguiente coeficientes:

δ_j^i Valor binario que indica si el ciclo j contiene al eje i.

p_j^i Es la protección que brinda el ciclo j al eje i ($p_j^i \in \{0, 1, 2\}$ para cada $i \in E$, $j \in P$).

Se define el costo de cada ciclo como la suma de los costos de sus ejes:

$$C_j = \sum_{i \in E} c_i \delta_j^i \qquad \forall j \in P$$

La solución al problema deberá proveer suficiente protección para cubrir las demandas con el mínimo costo. Por ejemplo en la Figura 2.1 se encuentra la instancia COST239, que es un caso de prueba estándar muy utilizado para SCA y en la Figura 2.2 se encuentra una solución para dicha instancia (los ciclos necesarios y el número de copias para cada uno).

2.1.1. Modelo para SCA

Usando la notación anterior presentaremos un modelo de programación lineal entera muy simple que aparece en la literatura [7].

Se tienen las variables enteras x_j (≥ 0). Cada una representa el número de copias del ciclo j seleccionadas para proteger la red.

$$\min \sum_{j \in P} C_j x_j \tag{2.1}$$

$$\text{s.t.} \sum_{j \in P} p_j^i x_j \geq d_i, \forall i \in E \tag{2.2}$$

La función objetivo representa la suma de los costos de los ciclos que forman la solución. Las restricciones exigen que se cubran las demandas para cada eje. La cantidad de variables de este modelo puede ser exponencial respecto al tamaño del grafo en el peor caso. Esto se debe a que la cantidad de ciclos que tiene el grafo puede crecer exponencialmente a medida que crece la instancia a resolver, y como se tiene una variable por cada ciclo, la cantidad de variables también crecerá exponenecialmente.

Sin embargo este modelo sirve como base de varias heurísticas reportadas en la literatura.

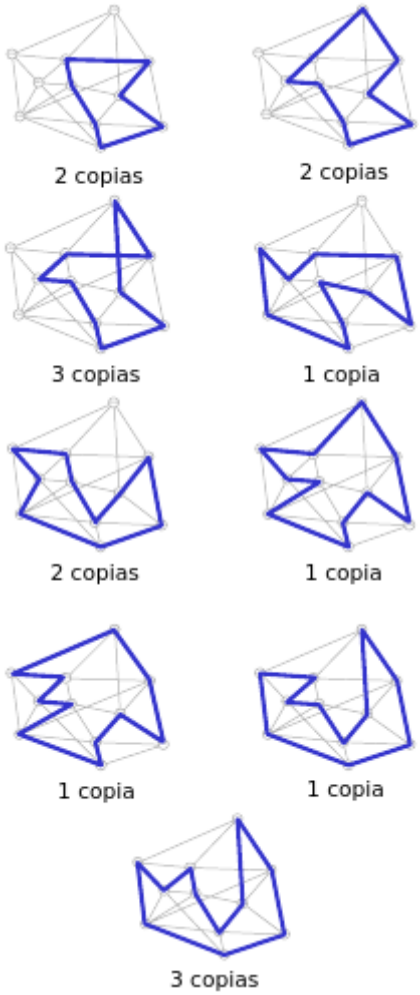


Fig. 2.2: Solución para la instancia COST239.

2.2. Extensiones y otros problemas relacionados

Existen algunas variantes del problema SCA y otros problemas relacionados sobre diseño de redes usando p-ciclos. En una variante de SCA, se puede tener como restricción adicional para resolver el problema que la longitud de los p-ciclos utilizados en la solución no exceda cierto límite [15]. Hay dos alternativas posibles:

1. Límite en el número de saltos: Esto significa que hay un número máximo de enlaces que puede atravesar un ciclo, sin importar el costo o longitud de los enlaces.

2. Límite en la *circunferencia*: Esto quiere decir que la circunferencia de cada p-ciclo no puede exceder un valor determinado. En la literatura se llama circunferencia del p-ciclo a la suma de los costos de sus enlaces. Esto se debe a que el costo en instancias reales está relacionado con la longitud.

Si bien puede existir alguna combinación de ambas, como por ejemplo que a la circunferencia se le agregue el número de saltos por algún factor, en la gran mayoría de los casos

se considera alguna de estas dos alternativas en caso de querer imponer un límite en la longitud de los ciclos.

En cualquiera de los dos casos no es necesario modificar el modelo presentado anteriormente. Simplemente basta con excluir las variables x_j que corresponden a los ciclos que exceden el límite previsto, o equivalentemente definir al conjunto P como los ciclos simples del grafo que cumplen esta condición.

Otro problema relacionado es el JCP (*Joint spare & working capacity problem*), en el cual no se tienen las demandas para cada eje. Se tiene una demanda entre cada par de nodos y se debe resolver simúltaneamente el problema de determinar los p-ciclos y de rutear las demandas. O sea, las demandas de cada eje deben ser determinadas junto con la capacidad redundante logrando una red superviviente. Se puede encontrar una presentación de este problema en [7].

También existe otra tecnología de supervivencia muy relacionada que utiliza FIPP p-ciclos. El nombre de FIPP p-ciclos viene de *failure independent path protecting*, y son en sí una generalización de p-ciclos en las que se protege no solo a los ejes *straddle* sino que también a los caminos simples (disjuntos al ciclo) que hay entre cada par de nodos del ciclo. Se puede encontrar una descripción más detallada en [8].

3. ESTADO DEL ARTE

A continuación presentaremos algunas heurísticas y modelos de programación entera para el problema SCA que se encuentran en la literatura.

3.1. Heurísticas

3.1.1. Preselección de ciclos candidatos

Como se vio previamente, el modelo presentado en la Sección 2.1.1 para el problema de SCA tiene como principal desventaja para ser utilizado en la práctica que el número de variables puede ser exponencial, ya que tiene una variable por cada ciclo simple del grafo. Esto es una limitación para el tamaño de las instancias que se pueden resolver en forma exacta.

Los primeros trabajos publicados basados en este modelo plantean seleccionar solamente un subconjunto del total de los ciclos simples del grafo. Estos ciclos seleccionados son llamados ciclos *candidatos* o *elegibles*. Se presentaron distintas heurísticas para seleccionar un conjunto de ciclos candidatos y luego, a partir de este conjunto, resolver el problema utilizando el modelo mencionado. De este modo, no está garantizado que se pueda obtener una solución óptima, y la calidad de la solución dependerá del conjunto de ciclos candidatos seleccionados. Como veremos más adelante este conjunto de ciclos candidatos puede servir de base de otras heurísticas para el mismo problema.

En [23] se presenta un algoritmo llamado SLA (*Straddling Link Algorithm*) que selecciona los ciclos más cortos posibles que contienen a cada eje como *straddle* haciendo dos búsquedas de camino mínimo. El número de ciclos obtenidos es a lo sumo $|E|$. Sin embargo, esto no provee un buen conjunto para la solución ya que los ciclos obtenidos tienen pocos nodos y por lo tanto pocos ejes **straddle**.

En [6] se definen, dado un ciclo j de P, el "puntaje topológico" y la "eficiencia *a priori*", que sirven como criterio para evaluar la calidad de los ciclos candidatos seleccionados por las heurísticas de selección de ciclos.

Puntaje Topológico es el número total de canales que puede proteger el p-ciclo sin tener en cuenta las demandas.

$$TS_j = \sum_{i \in E} p_j^i \tag{3.1}$$

Eficiencia a priori es el puntaje topológico divido por el costo del p-ciclo.

$$AE_j = \frac{TS_j}{C_j} \tag{3.2}$$

En [2] se proponen varios algoritmos para selección de ciclos. Estos toman como entrada un conjunto de ciclos candidatos inicial (se propone que sea el obtenido por SLA) y devuelve un conjunto más grande aplicando operadores. En resumen éstos operadores son:

SP-Add toma un ciclo existente y, por cada uno de sus ejes, construye un nuevo ciclo eliminando el eje en cuestión (que pasa a ser *straddle*) y reemplazándolo por el camino mínimo entre sus nodos que sea disjunto al ciclo original.

Expand realiza la misma operación que SP-Add, solo que si tiene éxito vuelve a repetirla con otro eje del ciclo original, hasta que no sea posible expandirlo más de esta manera.

Grow aplica primero SP-Add al conjunto de ciclos original. Luego realiza lo mismo que Expand con la diferencia que cada vez que repite la operación de buscar un camino disjunto al ciclo lo hace sobre los ejes del nuevo ciclo (es decir, que expande a partir de los nuevos ejes que fueron agregados además de los que ya estaban en el ciclo original).

El algoritmo *Grow* es el que obtiene una mayor eficiencia *a priori* promedio, aunque también es el que devuelve un conjunto de ciclos más numeroso. Asumiendo que empieza con a los sumo $|E|$ ciclos, devuelve a lo sumo $|E|^2|V|$ ciclos.

En [16] se propone una heurística para seleccionar ciclos a partir de una búsqueda DFS. Se tiene en cuenta la eficiencia *a priori* y se incluyen los ciclos más cortos que protegen a cada eje. Los resultados reportados muestran un mejor desempeño de este algoritmo sobre *Grow*.

3.1.2. Heurísticas constructivas

En la literatura se encuentran trabajos sobre heurísticas que construyen una solución para SCA a partir de un conjunto dado de ciclos candidatos (este conjunto puede ser el obtenido por alguna heurística de selección o bien el conjunto total de ciclos simples del grafo).

Una de las heurísticas más citada y sobre la cual se comparan los resultados de otros trabajos se llama CIDA (*Capacitated Iterative Design Algorithm*) y fue presentada en [2]. Es un algoritmo goloso en el que se construye la solución agregando un ciclo a la vez, para lo cual se define la "eficiencia real" (*actual efficiency*) de la siguiente manera:

$$E_j = \frac{1}{C_j} \sum_{i \in E} {w'}_i p_j^i \tag{3.3}$$

${w'}_i$ es la demanda no protegida del eje i. En cada iteración del algoritmo CIDA, se calcula la eficiencia real para cada ciclo candidato, se elige el que posea la mayor agregándolo a la solución y se actualizan las demandas, hasta proteger todos los ejes. A diferencia de la eficiencia *a priori*, se tienen en cuenta las demandas ya que se utiliza para construir una solución y no solo para seleccionar ciclos candidatos. En [1] se presenta una definición alternativa de la eficiencia real:

$${E'}_j = \frac{1}{C_j} \sum_{i \in E} \texttt{min}({w'}_i, p_j^i) \tag{3.4}$$

Sin embargo, se obtienen mejores resultados para instancias reales con la primer definición, ya que ésta impone prioridad a seleccionar primero los ciclos que protegen los ejes con mayor demanda.

Algoritmo 1 Pseudocódigo de CIDA

```
1: procedure CIDA
2:     inicializar demanda[], copiasEnSolucion[], ConjuntoDeCiclos
3:     ConjuntoDeCiclos ← EnumerarCiclos()
4:     while demanda[i] > 0 para algún eje i do
5:         MejorCiclo ← 0
6:         for all ciclo p en ConjuntoDeCiclos do
7:             Calcular E_p
8:             if E_p > E_MejorCiclo then
9:                 MejorCiclo ← p
10:            end if
11:        end for
12:        copiasEnSolucion[MejorCiclo] ← copiasEnSolucion[MejorCiclo] + 1
13:        for cada eje on-span i de MejorCiclo do
14:            demanda[i] ← demanda[i] - 1
15:        end for
16:        for cada eje straddle i de MejorCiclo do
17:            demanda[i] ← demanda[i] - 2
18:        end for
19:    end while
20: end procedure
```

El Algoritmo 1 corresponde al pseudocódigo de esta heurística. El procedimiento **EnumerarCiclos** puede ser enumerar todos los ciclos simples del grafo o bien utilizar los algoritmos *Grow*, *Expand* u otro algoritmo de selección de ciclos candidatos. La complejidad del algoritmo va a depender de cómo se construye este conjunto de ciclos.

En [17] se presenta otra heurística constructiva para este problema similar a CIDA. Se plantean otros criterios para la selección iterativa de ciclos candidatos para la solución. Los resultados reportados muestran un mejor desempeño que CIDA.

3.1.3. Selección de candidatos y construcción conjunta

En [9] se presenta una heurística a la que llaman GA-ILP que se basa en un algoritmo genético para realizar la selección de ciclos candidatos a medida que se busca una solución. El objetivo de esta heurística es obtener soluciones de calidad para instancias cuyo tamaño permite enumerar todos los ciclos pero no puede aplicarse el modelo de la Sección 2.1.1 por cuestiones de tiempo. Cada individuo de la población consiste en un conjunto de ciclos candidatos. Para comenzar, se calcula el conjunto de todos los ciclos simples y se divide en partes iguales, de modo que cada uno de estos subconjuntos será un individuo de la población inicial. En cada iteración se calcula la función objetivo para cada individuo: ésta es el resultado de resolver el problema de forma exacta (con el modelo presentado en la Sección 2.1.1) considerando solamente el subconjunto de ciclos representado por el individuo. La combinación se realiza entre los individuos de mejor función objetivo, de a pares, tomando la mitad de índices de cada uno y obteniendo así dos nuevos individuos para la nueva población. También se tiene un operador de mutación que modifica algunos individuos de la nueva población cambiando algunos índices elegidos al azar por otros

que estén en individuos no elegidos para la combinación. Si en alguna iteración ningún individuo logra superar el mejor valor de función objetivo obtenido en la iteración anterior, el procedimiento se detiene (es la condición de parada).

Este trabajo fue comparado con otro presentado en [14] que se basa en generación de columnas, obteniendo mejores resultados (menor costo para protección total) a pesar de que el de [14] se basa en FIPP p-ciclos. Por tratarse de una generalización de p-ciclos, el modelo utilizado (se presenta también en [14]) tiene un conjunto de soluciones mayor, que incluye al conjunto de soluciones del modelo de la Sección 2.1.1.

En [10] se modifica GA-ILP para resolver una instancia particular de 200 nodos y 394 ejes, la más grande reportada en la literatura. El número de ciclos de esta instancia es tan alto que no hace posible enumerarlos a todos, por lo que no es posible aplicar GA-ILP de forma directa. De manera que el procedimiento adoptado es el siguiente:

1. Por cada nodo se hace una selección de ciclos candidatos que lo incluyan, usando un algoritmo basado en DFS presentado en [5, Cap. 10: p-Cycles]. Así se obtienen $|V|$ conjuntos de ciclos candidatos. Se debe tener en cuenta que posiblemente cada uno de ellos por separado no provea ciclos suficientes para proteger toda la red.

2. Cada uno de estos subconjuntos se reduce por separado obteniendo $|V|$ subconjuntos aún más pequeños de ciclos (se reduce el número de ciclos candidatos). En este paso se utiliza una versión modificada de GA-ILP, en la cual se usa un modelo alternativo que no exige 100 % de protección pero que penaliza en la función objetivo el hecho de que existan demandas no protegidas.

3. Se combinan todos los subconjuntos de ciclos candidatos reducidos en el paso anterior en un único subconjunto de candidatos. Este nuevo conjunto sí posee ciclos suficientes para proteger toda la red.

4. Se resuelve el problema con este nuevo subconjunto utilizando el modelo original.

No hemos encontrado otros trabajos que comparen los resultados obtenidos con los de este algoritmo.

3.2. Enfoques basados en PLE sin enumeración de ciclos

Al hacer una selección previa de ciclos candidatos, se pueden dejar fuera ciclos que forman parte de una solución óptima, que ya no podrá ser encontrada. Por eso sería importante contar con todos los ciclos posibles para construir la solución.

Existen trabajos basados en programación lineal entera que evitan hacer una enumeración explícita de ciclos candidatos. En estos trabajos se presentaron varios modelos para el problema SCA distintos al presentado en la Sección 2.1.1. A partir de estos modelos se resuelve el problema utilizando un algoritmo Branch and Cut (en todos los trabajos reportados se utiliza el paquete comercial CPLEX [13]). Todos estos modelos tienen en común que el número de ciclos seleccionados para la solución tiene una cota máxima J determinada de antemano. Y de este valor J depende el número de variables y restricciones que posee el modelo, además del tamaño del grafo. La ventaja sobre el modelo presentado en 2.1.1 es que el número de variables y restricciones crece de manera polinomial respecto al tamaño de la instancia y de J.

El primer modelo fue presentado en [21]. Tiene un número de variables y restricciones elevado. El tiempo de ejecución que requiere es muy alto comparado con otros enfoques como los presentados anteriormente.

En [22] se presentan tres modelos y se comparan los resultados computacionales entre sí. Uno de ellos (lo llaman *Cycle Exclusion*) es el que obtiene los mejores resultados en calidad de solución y tiempo de ejecución, además de ser el más compacto: $3J(|E| + |V|)$ variables y $4J|E|+2J|V|+|E|+J$ restricciones. En este modelo las variables y restricciones pueden dividirse de forma tal que cada grupo representa un ciclo de la solución. Cada uno de estos grupos tiene incluidas restricciones de eliminación de *subtours* similares a las de Miller-Tucker-Zemlin (MTZ) para el problema del viajante de comercio.

Presentaremos el modelo *Cycle Exclusion* introducido en [22] porque será utilizado para comparar nuestro trabajo. Para este modelo se considera que por cada eje (u, v) se tienen dos vectores $u \to v$ y $v \to u$ que no pueden usarse simultáneamente. Los ciclos se forman con estos vectores y para el ciclo de la solución no importa el sentido utilizado. A cada nodo se le asigna un valor de voltaje y se debe cumplir que si se usa el vector $u \to v$ el voltaje de u debe ser menor que el de v. Además solo un nodo puede tener dos vectores salientes ("root"). Se puede ver un ejemplo en la figura 3.1. No existe forma de asignar voltajes a los nodos del ciclo de la derecha cumpliendo estas las reglas.

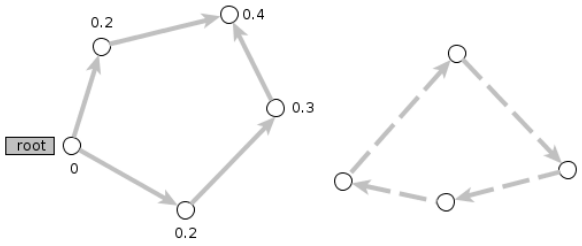


Fig. 3.1: Vectores y nodos con voltaje asociado.

Notación y datos de entrada:

J : Máximo número de ciclos para la solución.

j : Índice para el ciclo, $j \in \{1, 2, ..., J\}$.

V : Conjunto de nodos.

E : Conjunto de ejes.

$d_{(u,v)}$: Demanda del eje (u, v).

$c_{(u,v)}$: Costo por cada unidad de capacidad adicional en el eje (u, v).

α : Una constante predefinida tal que $0 < \alpha \leq 1/|V|$.

Variables:

y^j_{uv} : Binaria. Vale 1 sii se utiliza el vector $u \to v$ en el eje (u, v), en el ciclo j.

z_u^j : Binaria. Vale 1 sii el nodo u está en el ciclo j.

$x_{(u,v)}^j$: Binaria. Vale 1 sii el eje (u, v) puede ser protegido por el ciclo j.

r_u^j : Binaria. Vale 1 sii el nodo u es "root" en el ciclo j.

p_u^j : Variable real (≥ 0). Valor de voltaje del nodo u en el ciclo j.

Modelo:

$$min \sum_j \sum_{(u,v)\in E} c_{(u,v)}(y_{uv}^j + y_{vu}^j) \tag{3.5}$$

s.t.

$$y_{uv}^j + y_{vu}^j \leq 1 \qquad \forall j \leq J, \forall (u,v) \in E \tag{3.6}$$

$$\sum_{(u,v)\in E} (y_{vu}^j + y_{uv}^j) = 2 \times z_u^j \qquad \forall j \leq J, \forall u \in V \tag{3.7}$$

$$\sum_{j\leq J} (2 \times x_{(u,v)}^j - y_{uv}^j - y_{vu}^j) \geq d_{(u,v)} \qquad \forall (u,v) \in E \tag{3.8}$$

$$\sum_{u\in V} r_u^j \leq 1 \qquad \forall j \leq J \tag{3.9}$$

$$\sum_{(u,v)\in E} y_{uv}^j \leq 1 + r_u^j \qquad \forall j \leq J, \forall u \in V \tag{3.10}$$

$$p_v^j - p_u^j \geq \alpha \times y_{uv}^j - (1 - y_{vu}^j) \qquad \forall j \leq J, \forall (u,v),(v,u) \in E \tag{3.11}$$

$$x_{(u,v)}^j \leq (z_u^j + z_v^j)/2 \qquad \forall j \leq J, \forall (u,v) \in E \tag{3.12}$$

El costo total es minimizado por 3.5. Las restricciones 3.6 asocian cada eje (u, v) con a lo sumo un vector ($u \to v$ o $v \to u$) en cada ciclo. Con 3.7 se definen los ciclos: cada nodo tiene dos o ningún eje incidente. Las restricciones 3.8 aseguran un 100 % de protección. Con 3.9 se permite un solo nodo "root" por ciclo. Las restricciones 3.10 dicen que solo el nodo "root" puede tener dos vectores salientes. Para cada vector utilizado, 3.11 especifica que el nodo destino tiene mayor voltaje que el nodo origen. De este modo se garantiza que las variables y con mismo índice j forman un solo ciclo. Las restricciones 3.12 indican que un eje solo puede ser protegido si los dos nodos sobre los que incide pertenecen al ciclo.

Continuando con los trabajos encontrados en la literatura, en [18] se presenta otro modelo en el cual la formación de ciclos de la solución se hace por medio de una base de ciclos y restricciones de eliminación de *subtours* de MTZ. La base de ciclos se forma a partir de un árbol generador del grafo: cada eje que no está en el árbol forma un ciclo al agregarlo. Todos estos ciclos forman la base, y todos los ciclos del grafo pueden ser formados a partir de la unión disjunta de los ciclos de la base. El modelo tiene $5J|E| + 2J|V|$ variables y $5|E| + 2J|V| + J + |E|$ restricciones y obtiene resultados ligeramente mejores al modelo *Cycle Exclusion* de [22], usando CPLEX 10.2.

No conocemos otros trabajos de p-ciclos sin enumeración de ciclos candidatos aparte de los mencionados en esta sección.

4. TRABAJO PRELIMINAR: UN ALGORITMO GRASP PARA SCA

El primer enfoque que utilizamos para abordar el problema SCA fue heurístico. Implementamos un algoritmo basado en la metaheurística GRASP [3].

4.1. *Greedy Randomized Adaptative Search Procedures* (GRASP)

En un problema de optimización, tenemos un conjunto de soluciones S y una función objetivo $f : S \longrightarrow \mathbb{R}$ y queremos encontrar una solución $s^* \in S$ tal que $f(s^*) \leq f(s) \quad \forall s \in S$. GRASP es una metaheurística para obtener soluciones aproximadas a este tipo de problemas. Cada iteración de GRASP consiste en dos fases fundamentales:

- Fase de construcción.
- Fase de búsqueda local.

La mejor solución global se guarda como resultado. El Algoritmo 2 muestra el esquema básico de GRASP.

Algoritmo 2 Esquema general de GRASP

```
function GRASP
    while criterio de parada no satisfecho do
        s ← ConstructGreedyRandomizedSolution()
        LocalSearch(s)
        UpdateSolution(s, bestSolution)
    end while
    return bestSolution
end function
```

Durante la fase de construcción, la solución se construye paso por paso, de a un elemento por vez. Para cada problema que se quiera resolver se deberá definir claramente que es un **elemento** de la solución. En cada iteración, el próximo elemento a ser agregado a la solución se determina ordenando los elementos candidatos según un criterio goloso (*greedy*). Este criterio mide el costo incremental que aporta cada elemento candidato a la solución. La heurística es adaptativa porque en cada iteración se actualiza este costo incremental para reflejar el aporte del último elemento agregado. El componente probabilístico de GRASP se caracteriza por elegir al azar uno de los mejores elementos, pero no necesariamente el mejor de todos. Para esto se arma en cada iteración una lista con los mejores elementos, llamada lista restringida de candidatos (RCL por sus siglas en inglés). La longitud de la lista RCL es un parámetro a determinar en cada problema particular. El esquema general de la fase constructiva se presenta con el Algoritmo 3. Puede existir una fase intermedia de **reparación** en caso de que la etapa de construcción devuelva soluciones no factibles. En esta fase se modifican los elementos necesarios de la solución para hacerla factible.

Las soluciones obtenidas en la fase constructiva pueden no ser óptimas localmente respecto a algún criterio de vecindario. El vecindario $N(s)$ es un conjunto de soluciones

Algoritmo 3 Fase de construcción

```
function ConstructGreedyRandomizedSolution
    inicializar solución s
    inicializar el conjunto de elementos candidatos
    evaluar el costo incremental de cada elemento
    while solución s incompleta do
        construir lista RCL
        e ← elemento al azar de RCL
        s ← s ⋃ {e}
        actualizar el conjunto de elementos candidatos
        recalcular el costo incremental de cada elemento
    end while
    return s
end function
```

vecinas de s, lo que quiere decir que las soluciones de $N(s)$ pueden obtenerse cambiando algún elemento de s o realizando alguna operación sobre s. Del mismo modo, la solución s debe poder ser obtenida a partir de las soluciones de $N(s)$ realizando el proceso inverso. Todo esto dependerá de cada problema particular. Luego de la fase constructiva de GRASP se aplica un algoritmo de búsqueda local, en el cual se reemplaza iterativamente la solución obtenida por otra solución vecina con mejor valor de función objetivo, hasta que s sea óptima localmente. El esquema general de búsqueda local se presenta en el Algoritmo 4. La búsqueda local implementada dependerá de cada problema particular. Por ejemplo, la solución reemplazante puede ser la que tenga mejor valor de función objetivo en todo $N(s)$ o bien la primera en ser encontrada que supere a la que se quiere reemplazar. O puede pasar que si el vecindario definido es demasiado grande, no se explora todo por el costo computacional que implicaría y sólo se revisa un subconjunto del mismo.

Algoritmo 4 Fase de búsqueda local

```
procedure LocalSearch(s)
    while exista t ∈ N(s) tal que f(t) < f(s) do
        elegir una solución t ∈ N(s) tal que f(t) < f(s)
        s ← t
    end while
end procedure
```

Si la solución obtenida al finalizar el proceso de búsqueda local supera a la mejor obtenida hasta el momento, la reemplaza y se repite todo el procedimiento, siempre y cuando no se cumpla el criterio de parada de GRASP. El criterio de parada puede ser, por ejemplo, superar un número máximo de iteraciones totales, un número máximo de iteraciones sin mejorar la mejor solución encontrada, alcanzar una cota conocida, o cualquier otro criterio dependiente del problema particular.

4.2. Algoritmo basado en GRASP para SCA

Presentaremos el algoritmo desarrollado para SCA basado en GRASP. Para esto asumimos que además de los datos de entrada contamos con un conjunto P de ciclos candidatos a partir del cual se debe construir una solución.

4.2.1. Fase de Construcción

La fase de construcción implementada es una adaptación de la heurística CIDA introducida por Grover *et al.* en [2]. La solución de SCA es un conjunto (con repetidos) de ciclos, o bien un conjunto de pares < ciclo, número de copias >. Por lo tanto, los elementos para construir la solución iterativamente serán copias de los ciclos candidatos del conjunto P. El criterio utilizado para armar la lista RCL con los mejores candidatos será la **eficiencia real** cuya definición se encuentra en la sección 3.1.2.

En cada iteración se guardan los K ciclos con la mayor eficiencia real en la lista RCL (El parámetro K será determinado en base a la experimentación.) Luego se elige alguno de estos ciclos al azar y se agrega una copia del mismo a la solución. El pseudocódigo correspondiente se encuentra descripto en el Algoritmo 5. La función MejoresCiclos devuelve los K mejores ciclos de P según la eficiencia pasada como parámetro.

Algoritmo 5 Fase de construcción de nuestro algoritmo

```
1: procedure ConstruccionSCA(demandas[], Solucion, P)
2:     while demandas[i] > 0 para algún eje i do
3:         for all ciclo p en P do
4:             Calcular EficienciaReal[p]
5:         end for
6:         ListaRCL ← MejoresCiclos(K,EficienciaReal[],P)
7:         e ← elegir ciclo al azar de ListaRCL
8:         Agregar una copia del ciclo e a Solucion
9:         for cada eje on-span i de e do
10:            demandas[i] ← demandas[i] - 1
11:        end for
12:        for cada eje straddle i de e do
13:            demandas[i] ← demandas[i] - 2
14:        end for
15:    end while
16:    EliminarRedundancias(Solucion)
17: end procedure
```

Existe un primer inconveniente con el problema SCA. El conjunto de soluciones factibles no es finito. Podemos hacerlo finito si restringimos las soluciones factibles a ser minimales. Esto quiere decir, que si se quita una copia de algún ciclo de la solución, la misma deja de ser factible. Esto se logra con el procedimiento **EliminarRedundancias** que revisa iterativamente los ciclos de una solución y evalúa si pueden ser retirados, y en ese caso retira una copia. Repite el proceso hasta que la solución sea minimal. Por hacer esto también devuelve una solución de mayor calidad.

4.2.2. Fase de Búsqueda Local

Otro inconveniente que surge es la dificultad de definir un vecindario apropiado para este problema considerando que los elementos de la solución son copias de ciclos candidatos. Las opciones consideradas fueron muchas, pero ninguna resultó prometedora.

Entonces reemplazamos la búsqueda local por un simple algoritmo de mejora cuyo objetivo es reemplazar ciclos de la solución por otros que se encuentren en el conjunto de candidatos P, siguiendo nuevamente el criterio de la eficiencia real. El Algoritmo 6 muestra el procedimiento implementado. Por cada ciclo c de la solución se quita una copia y se reconstruye la solución prohibiendo el uso del ciclo c. La solución original es reemplazada por la solución con menor costo encontrada de esta manera. Se repite el proceso hasta que no se logre reducir el costo.

Algoritmo 6 Fase de búsqueda local de nuestro algoritmo para SCA

```
1: procedure MejorarSolucion(demandas[], Solucion, P)
2:     repeat
3:         mejorVecino ← Solucion
4:         SolAux ← Solucion
5:         for all ciclo c de Solucion do
6:             Quitar una copia de c de SolAux
7:             Marcar c en P para que no sea elegido.
8:             CIDA(demandas[],SolAux,P)
9:             if costo(SolAux) < costo(mejorVecino) then
10:                mejorVecino ← SolAux
11:            end if
12:            SolAux ← Solucion
13:        end for
14:        Solucion ← mejorVecino
15:    until No se reduce costo(Solucion)
16: end procedure
```

El procedimiento CIDA en este contexto puede ser definido por la fase de construcción presentada anteriormente estableciendo la longitud K de la lista RCL en 1. No podemos considerar este algoritmo de búsqueda local porque no tenemos un vecindario definido, ya que no hay garantía de que una vez modificada una solución se pueda recuperar nuevamente realizando una operación o movimiento inverso.

4.2.3. Esquema final

El Algoritmo 7 es el esquema final basado en GRASP. La complejidad del mismo depende del procedimiento **EnumerarCiclos** el cual es externo al algoritmo propuesto. Como criterio de parada establecimos un número máximo de iteraciones totales y de iteraciones sin mejorar la solución (el algoritmo termina al superar alguno de estos límites). Estos parámetros se ajustan durante la experimentación.

Algoritmo 7 Algoritmo GRASP para SCA

```
1: function GRASPBuilder(instanciaSCA)
2:     inicializar demandas[]
3:     inicializar MejorSolucion
4:     P ← EnumerarCiclos(instanciaSCA)
5:     Solucion ← ∅
6:     while criterio de parada no satisfecho do
7:         ConstruccionSCA(demandas[], Solucion, P)
8:         MejorarSolucion(demandas[], Solucion, P)
9:         if costo(Solucion) < costo(MejorSolucion) then
10:            MejorSolucion ← Solucion
11:        end if
12:    end while
13:    return MejorSolucion
14: end function
```

4.3. Resultados y conclusiones

A continuación comentaremos brevemente los resultados computacionales obtenidos y explicaremos la razón por la cual no continuamos abordando la resolución del problema SCA de esta manera.

Realizamos pruebas con la instancia estándar COST_239 y varias instancias reales de USA (las mismas que se mencionarán en el capítulo 7).

Tomaremos como referencia los resultados obtenidos por la heurística CIDA y por CPLEX utilizando el modelo presentado en la sección 2.1.1.

Nos concentraremos en analizar la calidad de las soluciones obtenidas y no el tiempo de ejecución. El algoritmo propuesto realiza varias iteraciones similares a las que hace la heurística CIDA y por lo tanto tomará más tiempo. Se espera que por este costo se obtenga el beneficio de obtener soluciones de mayor calidad. En cuanto al tiempo requerido por CPLEX, al ser exponencial en el peor caso, el tiempo requerido será comparable al de las heurísticas al resolver instancias pequeñas. A partir de un cierto tamaño de entrada el tiempo se incrementará drásticamente y superará ampliamente al de las heurísticas. Esto último dependerá fuertemente de las implementaciones particulares que hagamos de las heurísticas, que pueden ser optimizadas dependiendo de las estructuras de datos utilizadas. Por tales motivos no nos concentraremos en el tiempo de ejecución.

Para las pruebas realizadas establecimos distintos conjuntos de ciclos candidatos:

- Pequeño: ciclos obtenidos por el algoritmo SLA presentado en la sección 3.1.1.
- Grande: todos los ciclos hamiltonianos.
- Completo: todos los ciclos simples del grafo.

Para cada grupo de ciclos calculamos la mejor solución posible utilizando CPLEX y el modelo de la sección 2.1.1. Como se esperaba, con los dos primeros conjuntos no es posible encontrar una solución óptima. Y además, con el primer conjunto (el más pequeño) la mejor solución posible es mucho peor que con el segundo. En resumen, los resultados obtenidos para cada conjunto de ciclos candidatos fueron los siguientes:

- Pequeño: tanto con CIDA como con nuestro algoritmo se obtuvieron soluciones de muy baja calidad. Esto se debe a que el costo de la mejor solución posible con estos ciclos candidatos está muy lejos del óptimo (alrededor de un 20 % dependiendo de la instancia). Sin embargo, si comparamos respecto a esta mejor solución posible que depende del conjunto de candidatos, obtuvimos menos del 0.5 % de costo adicional contra 6-8 % de CIDA.

- Grande: nuestro algoritmo obtuvo mejores resultados que CIDA, 6-12 % respecto al óptimo contra 10-20 % de CIDA. En cuanto al mejor costo que puede obtenerse con este conjunto de ciclos, nuestro algoritmo estuvo a un 3-4 % dependiendo de cada instancia.

- Completo: los resultados obtenidos con nuestro algoritmo fueron similares a los obtenidos con el conjunto de ciclos anterior (6-11 % respecto al óptimo), es decir no hubo mejoras a pesar de disponer de más ciclos para elegir. Por otro lado, el desempeño de CIDA aumentó, logrando igualar al de nuestro algoritmo.

Vemos que a medida que crece el número de ciclos candidatos la calidad de la solución encontrada por nuestro algoritmo se ve reducida respecto a CIDA. Nuestro algoritmo tiene un gran desempeño sólo dentro de conjuntos de candidatos acotados, para los cuales la resolución por un algoritmo exacto es factible. Por lo tanto no podemos ver esto como una ventaja. La heurística CIDA a pesar de ser muy simple da resultados aceptables teniendo todos los ciclos del grafo, lo cual no sería posible con instancias de gran escala.

Algo que consideramos más importante aún fue evaluar cómo estaban constituidas las soluciones obtenidas. Comparando los índices de los ciclos utilizados por la solución óptima y por las soluciones obtenidas por CIDA y nuestro algoritmo, notamos que estas soluciones son muy distintas. Además notamos que un pequeño cambio en alguno de los ciclos llevaban a soluciones infactibles o de costo significativamente mayor.

Esto nos llevó a descartar una posible mejora de nuestro algoritmo con Path Relinking [20], ya que no tenemos indicios de poder encontrar soluciones de mejor calidad transformando una solución de calidad en otra realizando cambios de ciclos completos. Además está la dificultad adicional de que en el camino nos encontraríamos con muchas soluciones no factibles.

El objetivo inicial que teníamos era continuar el trabajo incorporando este algoritmo basado en GRASP dentro de un marco más amplio, como una metaheurística basada en poblaciones (la opción más fuerte era una Búsqueda Dispersa [20], o bien podía ser un Algoritmo Genético [19]), en el cual no exista un conjunto de ciclos candidatos fijo. La idea original era contar con varios conjuntos acotados de ciclos candidatos, los cuales puedan ser combinados definiendo operaciones para ello, y por medio del algoritmo GRASP propuesto construir soluciones para SCA.

Sin embargo, ante las dificultades encontradas (principalmente la característica de las soluciones) decidimos descartar esta posibilidad. Todo esto nos llevó a trabajar con otro enfoque en el que se evite hacer o depender de una enumeración explícita de ciclos candidatos.

5. MODELO SIN ENUMERACIÓN DE CICLOS PARA SCA

Los trabajos basados en heurísticas tienen en común que buscan reducir el conjunto de ciclos que pueden ser elegidos para la solución. Aunque luego se resuelva de forma exacta a partir de este conjunto reducido, la calidad de la solución se verá afectada. Y esto se acentúa cuanto más grande es la instancia a resolver dado que los métodos de selección buscan ser escalables, es decir que seleccionan una cantidad polinomial de ciclos respecto a la cantidad de ejes del grafo.

Es por eso que se tiene una ventaja sobre estos algoritmos si no se utiliza un conjunto de candidatos y por lo tanto se tienen en cuenta todos los ciclos simples del grafo para la solución.

Hemos formulado un nuevo modelo de programación lineal entera para resolver el problema SCA. El mismo no tiene en cuenta un conjunto de ciclos candidatos.

Tenemos coincidencias con otros modelos existentes que fueron mencionados en la Sección 3.2:

- Un límite predeterminado de J ciclos como máximo.
- Variables y restricciones que agrupadas representan un ciclo particular de la solución.
- Restricciones de eliminación de *subtours* para lograr el ítem anterior.

Por otro lado, para poder tener mejores resultados, se buscó lograr con la nueva formulación:

- Una forma mejor de eliminar *subtours* que MTZ.
- Menor cantidad de variables enteras.

El valor J arbitrario merece atención especial, ya que depende no sólo del tamaño del grafo sino también de las demandas de los ejes. El mínimo número de ciclos necesario para proteger una red con p-ciclos es $J_1 \geq \lceil \frac{D}{2} \rceil$ siendo D la demanda más alta de cualquier eje del grafo. También sabemos que la cota máxima para la cantidad de ciclos de una solución óptima es $J_2 \leq \sum_{i \in E} d_i$ donde d_i es la demanda del eje i. Entonces tenemos:

$$\left\lceil \frac{D}{2} \right\rceil \leq J \leq \sum_{i \in E} d_i$$

En el peor caso, la cota superior podría ser muy ajustada y podríamos tener una cantidad muy grande de ciclos en la solución. Sin embargo, en redes reales el número de ciclos de las soluciones óptimas no está lejos de la cota inferior. Por ejemplo, en la instancia COST_239 que es uno de los casos de prueba más citados, el valor $D = 11$, por lo que $J_1 \geq 6$ y el número de ciclos en todas sus soluciones óptimas es de 7. En redes reales los costos están relacionados con las distancias y las demandas de los ejes no son muy distintas entre sí (no hay valores muy grandes y otros muy chicos). Además, D no es demasiado grande, y si lo fuera se debería realizar la planificación en otro nivel de multiplexación (Esto quiere decir que cada unidad discreta de capacidad debe representar un mayor número de canales de comunicación).

Antes de introducir nuestro modelo, lo describiremos brevemente y de forma informal. Nos concentraremos solamente en cómo formar un ciclo de la solución, ya que luego se verá que agregar más ciclos será trivial ya que contamos con una cota J. También será fácil agregar restricciones para representar que se satisfacen las demandas.

Para seleccionar los ejes que forman un ciclo tendremos variables de decisión binarias (una por cada eje) y una formulación similar (sólo en esta parte) a la correspondiente al problema del viajante de comercio (simétrico). De modo que tendremos que garantizar que no se formen *subtours*. Esto es algo que podría parecer innecesario debido a que en la solución puede haber más de un ciclo. El problema que trae evitar la eliminación de *subtours* es que se requiere más esfuerzo (muchas restricciones) para poder reconocer los ejes *straddle*. Esto ya fue hecho en [22]. Uno de los modelos es muy complejo y no dió buenos resultados.

La eliminación de *subtours* aplicada en nuestro modelo está basada en **flujo** y es similar a la presentada por Gavish y Graves [4] para el TSP. Al grafo original adicionamos dos nodos más, s y t (fuente y sumidero), adyacentes a todos los nodos originales, pero no entre sí. Además impondremos las siguientes reglas:

- Se permitirá que exista flujo de s a cualquier otro nodo del grafo, pero sólo a uno a la vez por ciclo.
- Se permitirá flujo en cualquier sentido entre los nodos originales del grafo, pero sólo en el caso de que el eje en cuestión esté seleccionado para formar parte del ciclo.
- Existirá flujo de los nodos originales hacia t cuyo valor puede ser exclusivamente 1 ó 0.

Para que un eje sea considerado protegido debe ocurrir que el flujo de sus nodos hacia t no sea nulo. Esto se ve en la figuras 5.2 y 5.3. El costo del ciclo se verá reflejado en la función objetivo y dependerá de los ejes elegidos para ser parte del ciclo. De manera que no se podrá dar el caso de la figura 5.1 en el que hay un segundo ciclo de tres nodos con un flujo de valor V, ya que este ciclo elevará el costo de la solución y no aportará a cubrir las demandas según la formulación que presentaremos.

Por último, para diferenciar si la protección es de 1 o 2 unidades de capacidad bastará con verificar si el eje pertenece o no al ciclo. Si el eje es parte del ciclo la protección es de 1 unidad de capacidad (Fig. 5.2) y en el otro caso es de 2 unidades (Fig. 5.3).

Notación y Datos de entrada

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

$c_{(u,v)}$ costo del eje (u, v)

$d_{(u,v)}$ demanda del eje (u, v)

J máximo número de ciclos posible para formar la solución.

j índice para los ciclos ($1 \leq j \leq J$).

- Se agregan nodos s (fuente) y t (sumidero) para flujo.

Se debe tener en cuenta que la instancia es un grafo no dirigido, por lo que un eje que incida sobre los nodos u y v será notado como (u, v) o (v, u) indistintamente.

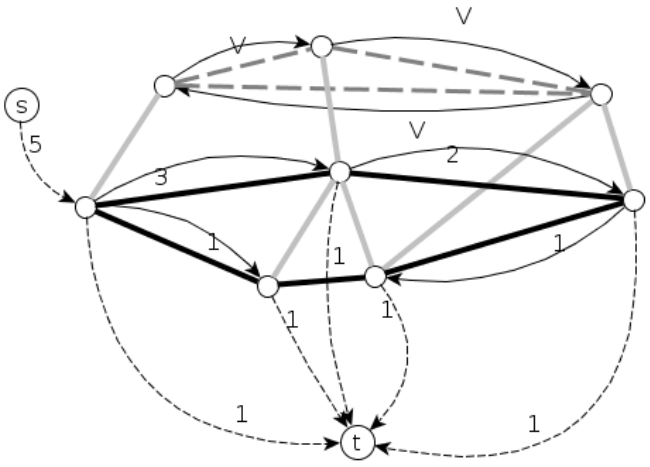


Fig. 5.1: Un ciclo para la solución.

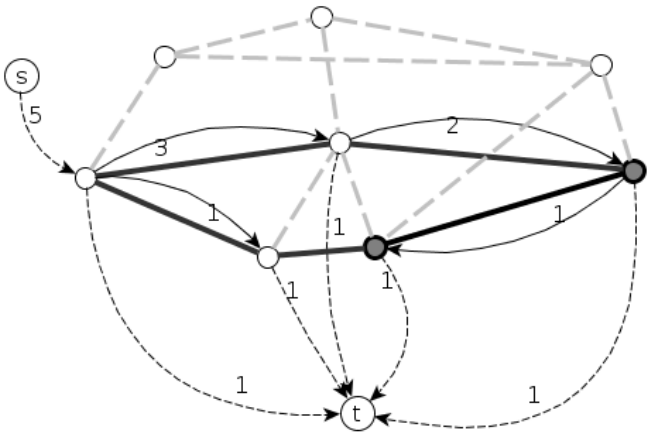


Fig. 5.2: Ejemplo de protección *on-span*.

Variables

$z^j_{(u,v)}$: Binaria. $\forall (u,v) \in E$. Vale 1 si se usa el eje (u,v) en el ciclo j, 0 en caso contrario.

$y^j_{(u,v)}$: Real (≥ 0). $\forall (u,v) \in E$. Representa las unidades de capacidad de protección que brinda el ciclo j al eje (u,v). Si bien este es un valor entero en la realidad (0, 1 ó 2), se puede dejarlo real ya que por las restricciones tomará siempre un valor entero.

x^j_{uv} : Real (≥ 0). $\forall u \in V, v \in N_G(u)$ y para $u = s, v \in V$. Representa el flujo en el eje $u \to v$ dirigido en el ciclo j. Esta variable al igual que las siguientes es necesaria para garantizar que las variables $z^j_{(u,v)}$ forman un ciclo para cada j.

b^j_u : Binaria. $\forall u \in V$. Representa el flujo en el eje $u \to t$ dirigido en el ciclo j. Alternativamente se puede ver a esta variable como que vale 1 si y sólo si el nodo u es parte del ciclo j.

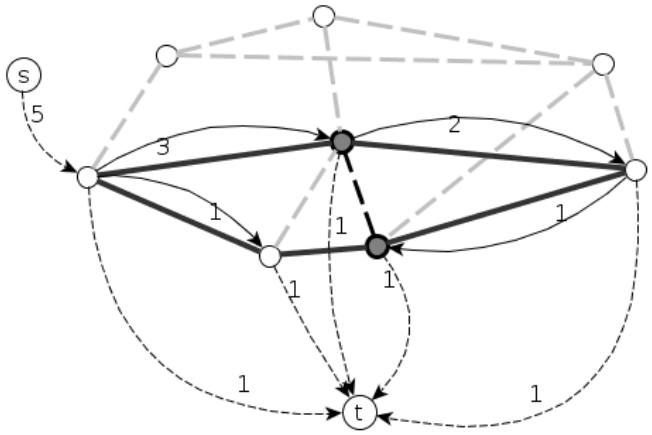


Fig. 5.3: Ejemplo de protección *straddle*.

r_u^j : Binaria. $\forall u \in V$. Vale 1 si se permite flujo de s hacia u para el ciclo j, 0 en caso contrario.

Modelo

$$min \sum_j \sum_{(u,v)\in E} c_{(u,v)} z^j_{(u,v)} \tag{5.1}$$

s.t.

$$\sum_{u\in V} r_u^j \leq 1 \qquad \forall j \leq J \tag{5.2}$$

$$x^j_{su} \leq |V| \times r_u^j \qquad \forall j \leq J, \forall u \in V \tag{5.3}$$

$$\sum_{v\in N_G(u)\cup\{s\}} x^j_{vu} - \sum_{v\in N_G(u)} x^j_{uv} = b_u^j \qquad \forall j \leq J, \forall u \in V \tag{5.4}$$

$$x^j_{uv} + x^j_{vu} \leq (|V|-1) \times z^j_{(u,v)} \qquad \forall j \leq J, \forall (u,v) \in E \tag{5.5}$$

$$\sum_{(u,v)\in E} z^j_{(u,v)} = 2 \times b_u^j \qquad \forall j \leq J, \forall u \in V \tag{5.6}$$

$$y^j_{(u,v)} \leq 2 \times b_w^j \qquad \forall j \leq J, \forall (u,v) \in E, \forall w \in \{u,v\} \tag{5.7}$$

$$y^j_{(u,v)} \leq 2 - z^j_{(u,v)} \qquad \forall j \leq J, \forall (u,v) \in E \tag{5.8}$$

$$\sum_{j\leq J} y^j_{(u,v)} \geq d_{(u,v)} \qquad \forall (u,v) \in E \tag{5.9}$$

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

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

$$b_u^j \in \{0,1\}, \quad r_u^j \in \{0,1\} \qquad \forall j \leq J, \forall u \in V$$

La función objetivo 5.1 toma en cuenta los costos de los ejes de todos los ciclos, que es lo que se debe minimizar. Las restricciones 5.2 garantizan que, por cada ciclo, sea solo un nodo el que tenga flujo desde el nodo s, y las 5.3 se complementan con las anteriores para lograr esto. El coeficiente $|V|$ puede ser reemplazado por otro valor mayor o igual. Con $|V|$ como mínimo se garantiza que se pueden formar todos los ciclos simples. Las restricciones 5.4 corresponden a la conservación de flujo en cada nodo: El flujo entrante es igual al saliente. Con 5.5 se garantiza que sólo haya flujo en los ejes seleccionados para formar parte del ciclo. El valor $(|V|-1)$ puede ser reemplazado por otro mayor. $(|V|-1)$ es el mínimo valor que permite que puedan haber ciclos de longitud menor o igual a $|V|$ (todos los ciclos simples). Con 5.6 se forman los ciclos. Por sí solas, estas restricciones dicen que todos los nodos tienen o bien 2 ejes incidentes, o bien ninguno. De este modo es necesario eliminar los *subtours* que podrían formarse. Las restricciones 5.7 y 5.8 imponen límites sobre la protección que brinda el ciclo, haciéndola consistente con el problema:

- Si un nodo sobre el que incide un eje no es parte del ciclo la protección para ese eje será nula. En caso de que sea parte puede ser a lo sumo 2 (*straddle*). Esto indican las restricciones 5.7. Notar que existen dos restricciones de este tipo por cada ciclo y por cada eje.

- Si un eje no es usado por el ciclo la protección puede ser a lo sumo 2 (*straddle*). En caso de que sea usado el valor será a lo sumo 1 (de hecho es lo que debe valer exactamente). Esto está determinado por las restricciones 5.8.

Las restricciones 5.9 son las únicas que no están definidas para cada ciclo. Hay una por cada eje y garantiza que se provea la capacidad redundante suficiente para el eje correspondiente.

5.1. Comparación con otras formulaciones

Nuestra formulación es comparable a las existentes en la literatura que han dado los mejores resultados, en relación a cantidad de restricciones y variables, la cual crece linealmente respecto a la cantidad de ejes de la instancia. Sin embargo, cabe notar que la cantidad de variables enteras es notoriamente menor. Esto se resume en la tabla 5.1. Haremos referencia a nuestro modelo como FLOW. Los otros dos, WYH y CB, son los mejores modelos presentados en [22] y [18] respectivamente.

Modelo	Variables totales	Variables enteras	Restricciones
WYH	$3J(\|E\|+\|V\|)$	$3J\|E\|+2J\|V\|$	$4J\|E\|+2J\|V\|+J+\|E\|$
CB	$5J\|E\|+2J\|V\|$	$5J\|E\|+J\|V\|$	$5J\|E\|+2J\|V\|+J+\|E\|$
FLOW	$3J\|E\|+2J\|V\|$	$J\|E\|+2J\|V\|$	$4J\|E\|+3J\|V\|+J+\|E\|$

Tab. 5.1: Comparación con otras formulaciones.

5.2. Restricciones adicionales

5.2.1. Simetría

Uno de los inconvenientes principales del modelo presentado y también de todos los modelos mencionados en la Sección 3.2 es que permiten múltiples soluciones simétricas. Es posible evitarlas introduciendo restricciones adicionales al modelo, las cuales no son necesarias para resolver el problema SCA pero prohíben la formación de soluciones simétricas.

Es posible hacer una clasificación en dos tipos de simetría:

1. Simetría de *source*: Dado un ciclo j' de la solución representado por las variables correspondientes, puede obtenerse el mismo ciclo una cantidad de veces igual a la longitud del mismo. Esto se deben porque necesariamente alguna variable $r_u^{j'}$ tendrá valor igual a 1 y las demás a 0, pero el nodo u correspondiente puede ser cualquiera de los que forman parte del ciclo.

   Se resuelve agregando el conjunto de restricciones 5.10, por medio de las cuales se exige una prioridad en los nodos que pueden recibir el flujo del nodo s agregado para la eliminación de *subtours*. Se debe tener en cuenta particularmente para este caso que los subíndices u-v que se refieren a los nodos son etiquetas numéricas que van de 1 hasta $|V|$.

$$u \times r_u^j \leq u - \sum_{v \in V, v<u} b_v^j \qquad \forall j \leq J, \forall u \in V \tag{5.10}$$

   Para cada ciclo de la solución se impone que, entre todos los nodos del ciclo, el único cuya variable r puede valer 1 es el de menor etiqueta numérica. Por ejemplo, teniendo un ciclo formado por los nodos 3, 4 y 6, la única variable r que puede valer 1 es la correspondiente al nodo 3 (ver Figura 5.4, las restricciones 5.10 permiten sólo el caso de la izquierda y prohíben los otros dos).

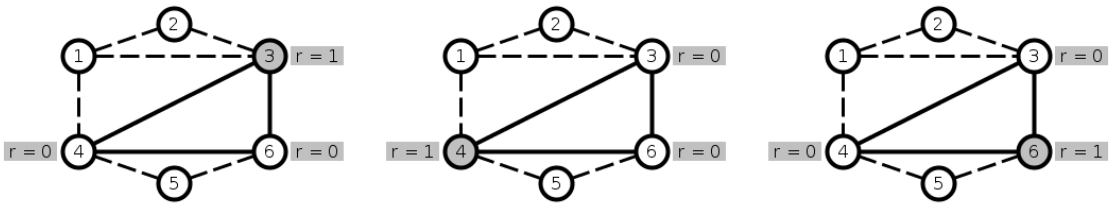


Fig. 5.4: Ejemplo de simetría de "source".

2. Simetría por permutación: Si separamos las variables del modelo en J conjuntos de acuerdo al índice j correspondiente, cada uno de estos conjuntos corresponde a un ciclo de la solución. Entonces, dada una solución factible al problema, se puede obtener otra igual intercambiando los valores de las variables de estos conjuntos entre sí. Por ejemplo, intercambiar los valores de b_u^1 y b_u^2 para cada nodo $u \in V$, los de $z_{(u,v)}^1$ y $z_{(u,v)}^2$ para cada eje $(u, v) \in E$, y así con todas las variables con índice j igual a 1 y 2. Es decir que, en el peor de los casos, a partir de una solución de J ciclos distintos se pueden obtener otras $J! - 1$ soluciones iguales.

Presentaremos por completitud las restricciones 5.11 que servirían para evitar esto, imponiendo un orden en los ciclos que forman la solución. Sin embargo, en pruebas preliminares se observó que agregándolas al modelo no se obtienen buenos resultados y es mejor no incluirlas y dejar las herramientas *built-in* que tiene CPLEX 12.4 [13] para tratar el inconveniente de la simetría.

$$\sum_{(u,v)\in E} \beta_{(u,v)} z^{j}_{(u,v)} \geq \sum_{(u,v)\in E} \beta_{(u,v)} z^{j+1}_{(u,v)} \qquad \forall j \leq J-1 \tag{5.11}$$

Los coeficientes $\beta_{(u,v)}$ determinarán si se puede eliminar totalmente la simetría por permutación o solo una parte. Elegimos como valores $\beta_{(u,v)} = log(p_{(u,v)})$ siendo $p_{(u,v)}$ un número primo (uno distinto para cada eje). Sabemos que cada número tiene una factorización uńica en números primos, por lo que si esta factorización es distinta, los números son distintos. Entonces queremos asignar a cada ciclo del grafo un número distinto, y por lo tanto, una factorización en primos distinta, para tener un orden total y evitar la simetría con las restricciones 5.11. Teniendo un número primo por eje $p_1, p_2, ..., p_{|E|}$, sabemos que para dos ciclos distintos j_1 y j_2 ocurre

$$\prod_{(u,v)\in E} (p_{(u,v)})^{z^{j_1}_{(u,v)}} \geq \prod_{(u,v)\in E} (p_{(u,v)})^{z^{j_2}_{(u,v)}}$$

o viceversa, en cuyo caso se intercambian los índices. Debido a que la función logaritmo es creciente, también ocurre

$$\sum_{(u,v)\in E} log(p_{(u,v)}) z^{j_1}_{(u,v)} \geq \sum_{(u,v)\in E} log(p_{(u,v)}) z^{j_2}_{(u,v)}$$

Así se elimina totalmente esta simetría.

En los otros modelos CB y WYH también se encontraban estos mismos problemas de simetría, además de otro tipo de simetría que hemos evitado en esta formulación. La misma ocurría porque se utilizaban dos variables enteras por cada eje (como si el grafo fuera dirigido) y por cada ciclo, dando la posibilidad de tener el mismo ciclo en ambos sentidos. En esos trabajos no se han reportado soluciones al problema de la simetría, por lo que estamos introduciendo una posible solución también para ellos.

5.2.2. Otras restricciones

Además de las restricciones para eliminar múltiples soluciones simétricas es posible obtener mejores resultados teniendo en cuenta otras restricciones que no son necesarias ni quitan soluciones simétricas. Un ejemplo de esto son las igualdades 5.12 que reducen los tiempos de ejecución como se verá más adelante en la Sección 7.2. Simplemente dicen que el flujo saliente del nodo s es igual al flujo entrante t, lo cual puede deducirse por otras restricciones del modelo.

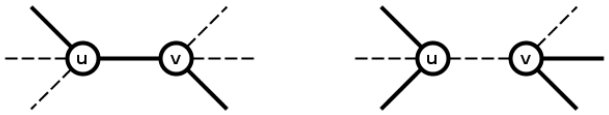
$$\sum_{u\in V} x^{j}_{su} = \sum_{u\in V} b^{j}_{u} \qquad \forall j \leq J \tag{5.12}$$

Las desigualdades 5.13 describen la protección de un eje (u, v) en relación al uso de los ejes que inciden sobre los nodos u y v, a excepción del mismo (u, v). Esto permite

ajustar mejor el valor de las variables $y^j_{(u,v)}$ que representan la protección en el caso de una relajación de integralidad de las variables $z^j_{(u,w)}$. Las desigualdades son válidas en los 3 casos de protección (*straddle*, *on-span* y nula) y en los casos de protección no nula se cumplen por igualdad. En el caso *on-span* (Figura 5.5a) cada sumatoria vale 1, por lo que la cota para la variable $y^j_{(u,v)}$ es de 1. En el caso *straddle* (Figura 5.5b) cada sumatoria tendrá valor 2, de modo que la cota es de 2. Es muy importante para el caso *straddle*, ya que un valor alto de $y^j_{(u,v)}$ aporta a la factibilidad de la solución con un bajo costo (función objetivo) ya que no se utiliza el eje (u, v) para el ciclo j.

$$y^j_{(u,v)} \leq \frac{1}{2} \times \Big(\sum_{(u,w)\in E, w\neq v} z^j_{(u,w)} + \sum_{(v,w)\in E, w\neq u} z^j_{(v,w)} \Big) \qquad \forall j \leq J, \forall (u,v) \in E \tag{5.13}$$

Es esperable que agregar estas restricciones de buen resultado ya que se complementan con las restricciones 5.7. Esto se verá más adelante en la Sección 7.3.



(a) Protección *on-span* *(b)* Protección *straddle*

Fig. 5.5: Ejemplos de protección del eje (u, v)

5.3. Extensiones del modelo a problemas con límites impuestos sobre los ciclos

En ciertas situaciones se puede dar que se impone un límite sobre la longitud de los ciclos como se explicó en la Sección 2.2. Esto puede ser incorporado muy fácilmente a nuestro modelo en ambos casos:

- Para límite de saltos L_H simplemente se deben modificar algunos coeficientes. Hay dos alternativas:

  1. Reemplazar el valor de $|V|$ por L_H en las restricciones 5.3.
  2. Reemplazar el valor de $|V| - 1$ por $L_H - 1$ en las restricciones 5.5.

  En ambos casos la cantidad de nodos con flujo hacia t no puede ser mayor a L_H y por lo tanto esto limita la longitud del ciclo a L_H ejes.

- Para límite de circunferencia L_C deben ser agregadas las restricciones que siguen a continuación:

$$\sum_{(u,v)\in E} c_{(u,v)} z^j_{(u,v)} \leq L_C \qquad \forall j \leq J \tag{5.14}$$

  Esto impone el límite de costo L_C en cada ciclo.

6. HEURÍSITICA SIN ENUMERACIÓN DE CICLOS PARA SCA

En este capítulo presentaremos un algoritmo para resolver SCA basado en la heurística CIDA introducida en [2] y en nuestro modelo.

6.1. Modelo de PLE para maximizar eficiencia real

El procedimiento de CIDA es muy simple: a partir de un conjunto de ciclos candidatos preseleccionados, la heurística elige el ciclo con la mayor **eficiencia real** (ver definición en la sección 3.1.2). Luego agrega una copia del ciclo elegido a la solución, actualiza las demandas y vuelve a repetir el procedimiento hasta que todas las demandas sean satisfechas (el pseudocódigo se encuentra en la página 11).

Esta heurística ha dado resultados aceptables en la práctica. El éxito está en la definición de la *eficiencia real*. Para el enfoque que venimos trabajando basado en programación lineal entera, nos puede ser útil para proveer una solución inicial para utilizar nuestro modelo con CPLEX. También puede ser útil para estimar el valor de J, ya que este valor arbitrario debe ser determinado de forma heurística en la práctica.

Tiene como desventaja que los buenos resultados (calidad de la solución) dependen del conjunto de ciclos candidatos con el que se cuenta para construir una solución. Para tener soluciones de calidad se necesita contar con todos los ciclos simples del grafo. Sin embargo esto no sólo es una desventaja porque enumerar los ciclos es una tarea de fuerza bruta, sino que el propio algoritmo CIDA se convierte en un procedimiento de fuerza bruta porque tiene que calcular la eficiencia real para cada ciclo candidato.

Por eso modificamos el modelo introducido para SCA en el capítulo anterior para poder resolver un nuevo problema que consiste en *encontrar el ciclo con la mayor eficiencia real* dada una instancia para SCA. El objetivo es obtener un modelo de PLE para resolver este nuevo problema.

El primer inconveniente que surge al modelar el problema es que la función objetivo planteada no es más una función lineal, sino un cociente entre funciones lineales. A continuación presentaremos el modelo propuesto, explicaremos el procedimiento para transformarlo en un modelo de PLE y presentaremos las modificaciones correspondientes.

Los datos de entrada son los mismos que para SCA:

- La red es un grafo 2-conexo $G = (V, E)$

$c_{(u,v)}$ costo del eje (u, v)

$d_{(u,v)}$ demanda del eje (u, v)

Se agregan nodos s (fuente) y t (sumidero) para flujo, ya que se hace la eliminación de *subtours* por flujo.

Variables:

$z_{(u,v)}$ Binaria. $\forall (u, v) \in E$. Vale 1 si se usa el eje (u, v), 0 en caso contrario.

$y_{(u,v)}$ Binaria. $\forall (u, v) \in E$. Vale 1 si el ciclo brinda protección *straddle* al eje (u, v).

x_{uv} Real (≥ 0). $\forall u \in V, v \in N_G(u)$ y para $u = s, v \in V$. Representa el flujo en el eje $u \to v$ dirigido.

b_u Binaria. $\forall u \in V$. Corresponde al flujo en el eje $u \to t$ dirigido. Alternativamente se puede ver a esta variable como que vale 1 si y sólo si se utiliza el nodo u para el ciclo.

r_u Binaria. $\forall u \in V$. Vale 1 si se permite flujo de s hacia u, 0 en caso contrario.

Notar que hay un cambio más en las variables (las variables y no significan lo mismo que en el modelo anterior), para tener sólo variables binarias en la función objetivo.

Modelo:

$$max \quad \frac{\sum_{(u,v)\in E} d_{(u,v)} \times (2 \times y_{(u,v)} + z_{(u,v)})}{\sum_{(u,v)\in E} c_{(u,v)} z_{(u,v)}} \tag{6.1}$$

s.t.

$$\sum_{u\in V} r_u \leq 1 \tag{6.2}$$

$$x_{su} \leq |V| \times r_u \qquad \forall u \in V \tag{6.3}$$

$$\sum_{v\in N_G(u)\cup\{s\}} x_{vu} - \sum_{v\in N_G(u)} x_{uv} = b_u \qquad \forall u \in V \tag{6.4}$$

$$x_{uv} + x_{vu} \leq (|V| - 1) \times z_{(u,v)} \qquad \forall (u,v) \in E \tag{6.5}$$

$$\sum_{(u,v)\in E} z_{(u,v)} = 2 \times b_u \qquad \forall u \in V \tag{6.6}$$

$$y_{(u,v)} \leq b_w \qquad \forall (u,v) \in E, \forall w \in \{u,v\} \tag{6.7}$$

$$y_{(u,v)} \leq 1 - z_{(u,v)} \qquad \forall (u,v) \in E \tag{6.8}$$

$$u \times r_u \leq u - \sum_{v\in V, v<u} b_v \qquad \forall u \in V \tag{6.9}$$

$$z_{(u,v)} \in \{0,1\}, \quad y_{(u,v)} \in \{0,1\} \qquad \forall (u,v) \in E$$

$$x_{uv} \in \mathbb{R}_+ \qquad (\forall u \in V, v \in N_G(u)) \wedge (u = s, v \in V)$$

$$b_u \in \{0,1\}, \quad r_u \in \{0,1\} \qquad \forall u \in V$$

La función objetivo 6.1 es la definición de la **eficiencia real** a partir de las variables del modelo. La mayoría de las restricciones son prácticamente iguales a las presentadas (sin el índice j por supuesto porque ahora la solución es un solo ciclo). Las restricciones 6.2 garantizan que sea sólo un nodo el que tenga flujo desde el nodo s, y las 6.3 se complementan con las anteriores para lograr esto. Las restricciones 6.4 corresponden a

la conservación de flujo en cada nodo: el flujo entrante es igual al saliente. Con 6.5 se garantiza que se sólo haya flujo en los ejes seleccionados para formar parte del ciclo. Con 6.6 se forman los ciclos: dicen que todos los nodos tienen o bien 2 ejes incidentes, o ninguno. Por eso es necesario eliminar los *subtours* que podrían formarse. Las restricciones 6.7 y 6.8 cambian respecto al modelo para SCA ya que el significado de las variables y cambió. Sirven para limitar la protección que brinda el ciclo (en este caso sólo la protección *straddle*). Se agregan también las restricciones 6.9 para eliminar soluciones simétricas.

Ahora el objetivo es modificar este modelo para obtener uno equivalente de PLE. Explicaremos las modificaciones necesarias en un caso más simple para evitar confusiones por la complicación de tener variables binarias con distinto nombre (y y z). La función objetivo en cuestión es de la forma:

$$\frac{\sum_{i \in T} k_i \times t_i}{\sum_{i \in T} l_i \times t_i}$$

para algún conjunto T, coeficientes k, l y variables binarias t. Se asume también que el denominador será siempre mayor estricto que cero.

Reemplazamos el denominador por una nueva variable

$$q_0 = \frac{1}{\sum_{i \in T} l_i \times t_i} \tag{6.10}$$

Agregamos nuevas variables reales no negativas

$$q_i = t_i \times q_0 \qquad \forall i \in T \tag{6.11}$$

y la función objetivo nos queda lineal:

$$\sum_{i \in T} k_i \times q_i$$

Las ecuaciones 6.10 y 6.11 deben estar incluidas en el modelo como restricciones. Sin embargo, debemos hacer algunos cambios para que sean lineales.

Empezamos por la ecuación 6.10. Multiplicamos ambos lados por el denominador:

$$(\sum_{i \in T} l_i \times t_i) \times q_0 = 1$$

Y utilizamos las igualdades 6.11 para obtener una restricción lineal:

$$\sum_{i \in T} l_i \times q_i = 1 \tag{6.12}$$

El último inconveniente a resolver son los productos $q_i = t_i \times q_0$ que también deben ser introducidos al modelo como restricciones. Es posible calcular cotas para q_0 a partir de los coeficientes l_i:

$$L \leq q_0 \leq U$$

Y como las variables t_i son binarias, las restricciones que introducimos son las siguientes:

$$L \times t_i \leq q_i \leq U \times t_i \tag{6.13}$$

$$L \times (1 - t_i) \leq q_0 - q_i \leq U \times (1 - t_i) \tag{6.14}$$

Estas restricciones implican 6.11. Notar que como cada variable t_i es binaria tenemos solamente dos casos.

- Si $t_i = 0$ entonces 6.13 implica que $q_i = 0$ y por lo tanto vale $q_i = t_i \times q_0$. En este caso 6.14 son las cotas para q_0.
- En el caso $t_i = 0$, 6.14 implica que $q_i = q_0$ y entonces vale $q_i = t_i \times q_0$. 6.13 implica $L \leq q_i \leq U$, lo cual no es una contradicción porque en este caso sabemos que $q_i = q_0$ y que L y U son cotas para q_0.

Finalmente introduciremos las modificaciones necesarias para transformar el modelo propuesto para maximizar la **eficiencia real** en uno de PLE. Se agregan las siguientes variables:

Δ ($\in \mathbb{R}_+$). Variable auxiliar que representa la inversa del costo del ciclo.

$\theta_{(u,v)}$ ($\in \mathbb{R}_+$). Variable auxiliar para el producto $z_{(u,v)} \times \Delta$.

$\mu_{(u,v)}$ ($\in \mathbb{R}_+$). Variable auxiliar para el producto $y_{(u,v)} \times \Delta$

Se modifica la función objetivo:

$$\sum_{(u,v)\in E} d_{(u,v)} \times (2 \times \mu_{(u,v)} + \theta_{(u,v)}) \tag{6.15}$$

Se agregan las siguientes restricciones:

$$\sum_{(u,v)\in E} c_{(u,v)} \times \theta_{(u,v)} = 1 \tag{6.16}$$

$$L \times z_{(u,v)} \leq \theta_{(u,v)} \leq U \times z_{(u,v)} \qquad \forall (u,v) \in E \tag{6.17}$$

$$L \times (1 - z_{(u,v)}) \leq \Delta - \theta_{(u,v)} \leq U \times (1 - z_{(u,v)}) \qquad \forall (u,v) \in E \tag{6.18}$$

$$L \times y_{(u,v)} \leq \mu_{(u,v)} \leq U \times y_{(u,v)} \qquad \forall (u,v) \in E \tag{6.19}$$

$$L \times (1 - y_{(u,v)}) \leq \Delta - \mu_{(u,v)} \leq U \times (1 - y_{(u,v)}) \qquad \forall (u,v) \in E \tag{6.20}$$

Con la restricción 6.16 se garantiza el valor:

$$\Delta = \frac{1}{\sum_{(u,v)\in E} c_{(u,v)} z_{(u,v)}}$$

Las restricciones 6.17 - 6.20 garantizan que se cumplan los productos $\theta_{(u,v)} = z_{(u,v)} \times \Delta$ y $\mu_{(u,v)} = y_{(u,v)} \times \Delta$.

Las constantes U y L se ajustan de acuerdo a los costos $c_{(u,v)}$: se calculan cotas C^- (inferior) y C^+ (superior) para el costo total del ciclo formado y se establece $L = \frac{1}{C^+}$ y $U = \frac{1}{C^-}$.

Ahora que tenemos este modelo de PLE (vamos a llamarlo $MCIDA$) para resolver el problema de encontrar el ciclo con mayor eficiencia real, redefinimos la heurística CIDA (algoritmo 8).

Algoritmo 8 Pseudocódigo de CIDA utilizando el nuevo modelo de PLE

```
1: procedure CIDA
2:     inicializar demanda[], copiasEnSolucion[]
3:     while demanda[i] > 0 para algún eje i do
4:         resolver MCIDA
5:         MejorCiclo ← resultado de MCIDA
6:         copiasEnSolucion[MejorCiclo] ← copiasEnSolucion[MejorCiclo] + 1
7:         for cada eje on-span i de MejorCiclo do
8:             demanda[i] ← demanda[i] - 1
9:         end for
10:        for cada eje straddle i de MejorCiclo do
11:            demanda[i] ← demanda[i] - 2
12:        end for
13:    end while
14: end procedure
```

6.2. Modelo alternativo

El trabajo realizado anteriormente fue para obtener el mismo resultado que obtendríamos si utilizáramos la heurística CIDA con todos los ciclos del grafo, pero evitando la enumeración explícita de los mismos. A partir del Algoritmo 8 podemos reemplazar el modelo utilizado ($MCIDA$) por cualquier otro que simplemente seleccione un ciclo maximizando algún criterio arbitrario de eficiencia a partir de una instancia de SCA. Es por eso que en lugar de hacer la transformación propuesta del modelo podemos simplemente definir otra función objetivo que en lugar de maximizar un cociente, maximice una resta y no haya que hacer modificaciones significativas al modelo original propuesto en el capítulo anterior para SCA.

Presentaremos a continuación la alternativa propuesta. Los datos de entrada son los mismos. Las variables son:

$z_{(u,v)}$ Binaria. $\forall (u,v) \in E$. Vale 1 si se usa el eje (u,v), 0 en caso contrario.

$y_{(u,v)}$ Real (≥ 0). $\forall (u,v) \in E$. Es la protección otorgada por el ciclo al eje (u,v).

x_{uv} Real (≥ 0). $\forall u \in V, v \in N_G(u)$ y para $u = s, v \in V$. Es el flujo en el eje $u \to v$ dirigido.

b_u Binaria. $\forall u \in V$. Es el flujo en el eje $u \to t$ dirigido. Alternativamente se puede ver a esta variable como que vale 1 si y sólo si se utiliza el nodo u para el ciclo.

r_u Binaria. $\forall u \in V$. Vale 1 si se permite flujo de s hacia u, 0 en caso contrario.

El modelo queda así:

$$max \quad \left(\sum_{(u,v)\in E} d_{(u,v)} \times y_{(u,v)} \right) - \omega \left(\times \sum_{(u,v)\in E} c_{(u,v)} z_{(u,v)} \right) \tag{6.21}$$

s.t.

$$\sum_{u \in V} r_u \leq 1 \tag{6.22}$$

$$x_{su} \leq |V| \times r_u \qquad \forall u \in V \tag{6.23}$$

$$\sum_{v \in N_G(u) \cup \{s\}} x_{vu} - \sum_{v \in N_G(u)} x_{uv} = b_u \qquad \forall u \in V \tag{6.24}$$

$$x_{uv} + x_{vu} \leq (|V| - 1) \times z_{(u,v)} \qquad \forall (u,v) \in E \tag{6.25}$$

$$\sum_{(u,v) \in E} z_{(u,v)} = 2 \times b_u \qquad \forall u \in V \tag{6.26}$$

$$y_{(u,v)} \leq b_w \qquad \forall (u,v) \in E, \forall w \in \{u,v\} \tag{6.27}$$

$$y_{(u,v)} \leq 1 - z_{(u,v)} \qquad \forall (u,v) \in E \tag{6.28}$$

$$\sum_{(u,v) \in E} d_{(u,v)} \times y_{(u,v)} \geq 1 \tag{6.29}$$

$$u \times r_u \leq u - \sum_{v \in V, v < u} b_v \qquad \forall u \in V \tag{6.30}$$

$$z_{(u,v)} \in \{0,1\}, \quad y_{(u,v)} \in \mathbb{R}_+ \qquad \forall (u,v) \in E$$

$$x_{uv} \in \mathbb{R}_+ \qquad (\forall u \in V, v \in N_G(u)) \wedge (u = s, v \in V)$$

$$b_u \in \{0,1\}, \quad r_u \in \{0,1\} \qquad \forall u \in V$$

Tenemos un coeficiente ω en la función objetivo 6.21 que podremos ajustar para obtener distintas soluciones.

Las restricciones 6.22 a 6.26 y 6.30 no cambiaron respecto al modelo anterior por lo que ya fueron explicadas. Las restricciones 6.27 y 6.28 no cambian respecto al modelo para SCA presentado en el capítulo anterior. Se agrega una nueva restricción 6.29 para garantizar que se cubren al menos una unidad de capacidad de demanda. Si no estuviera, la solución óptima podría consistir en todas las variables z con valor 0 o bien en un ciclo de bajo costo que no satisface ninguna demanda aún no protegida.

Para hacer referencia a este modelo lo llamaremos en adelante MHEUR(ω).

7. RESULTADOS COMPUTACIONALES

Presentaremos los resultados obtenidos a partir de las pruebas computacionales realizadas. Para este trabajo utilizamos CPLEX 12.4 [13] para resolver el problema SCA a partir de nuestro modelo en una PC Intel Core 2 Duo 1.8 GHz con 4GB de RAM.

CPLEX implementa un algoritmo *branch-and-cut* que básicamente consiste en relajaciones lineales, *branching*, cortes y heurísticas. Este algoritmo resuelve una serie de subproblemas continuos (programación lineal). Para hacerlo de forma eficiente, construye un árbol en el cual cada uno de estos subproblemas es un nodo. La raíz del árbol es la relajación lineal del problema original de programación entera. Si la solución de la relajación tiene una o más variables enteras con valores fraccionarios, intenta encontrar cortes, los cuales son restricciones que cortan áreas de la región factible de la relajación que contienen soluciones fraccionarias. El algoritmo puede generar distintos tipos de cortes por defecto y además permite agregar otros tipos de cortes definidos por el usuario. Si después de eso la solución de la relajación sigue teniendo variables enteras con valor fraccionario, escoge una de estas variables para generar dos subproblemas (*branching*), cada uno con cotas más restrictivas en dicha variable (si la variable es binaria, un nodo fija su valor en 0, y el otro en 1). Los subproblemas pueden resultar en una solución entera, no factible, o en otra solución fraccionaria. En este último caso, repite el proceso.

Para la experimentación realizada, las instancias de prueba son el caso de prueba estándar COST_239 e instancias reales de USA cuyo tamaño va desde los 6 a 13 nodos y de los 15 a 39 ejes.

En la práctica no se conoce el valor óptimo de J (o sea, la cantidad mínima de ciclos que tiene una solución óptima) y se determina algún valor de forma heurística. Pero para poder testear el modelo propuesto se determinó previamente el valor óptimo de J. Se resolvió cada instancia, enumerando en primer lugar todos los ciclos simples del grafo mediante el algoritmo propuesto en [5, Cap. 10: p-Cycles]. Luego se corrió el modelo de la sección 2.1.1 con CPLEX. Es importante aclarar que el tamaño de las instancias empleadas permite obtener el J óptimo. Para instancias más grandes podría no ser posible. El valor de J establecido es la cantidad de ciclos de la solución óptima obtenida de esta manera. De este modo también se cuenta con el valor óptimo de la función objetivo para cada instancia para tenerlo como referencia, para saber si se encontró una solución óptima a pesar de no alcanzar un *gap* de 0 %.

Para todos nuestros tests se se limitó el tiempo de ejecución a 100 segundos.

7.1. Comparación con otros modelos

Además de nuestro modelo, hemos implementado los de [22] (WYH) y [18] (CB). A diferencia de los resultados reportados por dichos autores (usando CPLEX 10), en nuestros experimentos con CPLEX 12.4 los mejores resultados para estos dos modelos se obtuvieron con WYH (con las misma instancias) . Por lo tanto, las comparaciones las haremos con

este modelo.

Previamente hemos determinado las condiciones para un mejor desempeño del modelo WYH. Se configuraron los parámetros de CPLEX del mismo modo que recomiendan en [22]:

- MIP Emphasis: 1 (factibilidad sobre optimalidad).
- Probing level: 2 (agresivo)
- Frecuencia para heurísticas RINS: 3
- Frecuencia para heurística periódica: 3
- Cortes(Todos): 2 (Uso agresivo)
- MIP dive strategy: 3 (Guided dive)
- Symmetry breaking: 3 (Muy agresivo)

Con esta configuración la ejecución demoró más tiempo para todas las instancias (cabe aclarar que en ese otro trabajo se utilizó una versión anterior, CPLEX 10.0), de hecho no llegó a alcanzar un *gap* de 0 % dentro del límite de tiempo establecido para ninguna de ellas. De modo que los resultados presentados en este trabajo corresponden al modelo presentado en [22] sin modificaciones y los parámetros de CPLEX en su configuración automática.

			Función objetivo			Gap (%)		Tiempo (seg.)	
Instancia	Ciclos	J	Óptimo	FLOW	WYH	FLOW	WYH	FLOW	WYH
COST_239	3531	7	32340	32340	32390	0.7	0.85	100	100
VZ_US_PIP_001	106967	7	32240	32640	32460	5.07	4.59	100	100
VZ_US_PIP_002	37286	6	37370	37370	37370	0	0.48	28	100
VZ_US_PIP_003	37286	7	35170	35170	35170	0	0.43	59.3	100
VZ_US_PIP_004	37286	8	39980	40840	40920	7.86	8.25	100	100
VZ_US_PIP_005	15341	7	24937	25117	24937	5.24	5.06	100	100
VZ_US_PIP_006	15341	6	26190	26190	26190	0	0.02	41.9	100
VZ_US_PIP_007	15341	7	30534	30534	30534	0	0.02	1.23	100
VZ_US_PIP_008	15341	7	32721	32961	33557	1.95	4.83	100	100
VZ_US_PIP_009	15341	7	37071	37071	37071	0.42	0.42	100	100
VZ_US_PIP_010	15341	8	44084	44084	44084	0	0	33.1	98
VZ_US_PIP_011	3354	8	42054	42054	42054	0.02	0.02	100	100
VZ_US_PIP_012	1636	8	35754	35754	35754	0	0.5	7.1	100
VZ_US_PIP_013	167	7	29984	29984	29984	0	1.17	13.4	100
VZ_US_PIP_014	70	6	14740	14740	14740	0	0	0.67	10.13
VZ_US_PIP_015	18	6	14775	14775	14775	0	0	0.19	4.16

Tab. 7.1: Comparción de FLOW con WYH

En la tabla 7.1 se ven los resultados obtenidos utilizando nuestro modelo y el de WYH para las instancias mencionadas. Se puede apreciar un mejor desempeño por parte de

nuestro modelo en la gran mayoría de los casos, logrando una ventaja muy grande en varias de las instancias.

También en esta primera tabla se presentan para cada instancia la cantidad total de ciclos y el valor de J determinado anteriormente para encontrar una solución óptima.

Presentaremos los resultados obtenidos luego del procesamiento del nodo raíz en la tabla 7.2. Podemos ver que desde este punto nuestro modelo permite obtener mejores cotas inferiores a partir de las relajaciones lineales. Debemos aclarar que CPLEX está aplicando cortes y utilizando heurísticas para encontrar soluciones factibles. Hay un caso para WYH en el que no logró encontrar una solución entera y por tal motivo tiene una "X" en la columna *gap*. También vale notar que para la instancia más pequeña con nuestro modelo sólo tuvo que procesar el nodo raíz para encontrar la solución óptima.

	Mejor cota		Gap (%)		Tiempo (seg.)	
Instancia	FLOW	WYH	FLOW	WYH	FLOW	WYH
COST_239	31815.0	31765.0	9.33	12.9	1.77	1.67
VZ_US_PIP_001	30945.0	30925.0	21.04	16.87	1.79	1.91
VZ_US_PIP_002	37120.0	37120.0	10.62	0.67	0.66	0.63
VZ_US_PIP_003	34430.0	34365.0	9.59	20.68	1.25	1.51
VZ_US_PIP_004	37156.668	37156.668	22.28	33.7	2.06	3.69
VZ_US_PIP_005	23514.0	23396.5	46.14	X	2.18	2.09
VZ_US_PIP_006	25729.5	25815.666	7.11	7.22	1.1	2.18
VZ_US_PIP_007	30193.0	30193.0	1.92	6.34	0.46	0.77
VZ_US_PIP_008	31752.6	31753.719	14.66	13.42	1.83	4.74
VZ_US_PIP_009	36763.285	36729.0	5.13	8.7	1.3	3.18
VZ_US_PIP_010	43622.715	43618.11	7.17	2.2	2.16	4.88
VZ_US_PIP_011	41942.6	41942.6	4.4	9.96	1.49	4.97
VZ_US_PIP_012	35394.332	35457.0	7.33	12.23	1.19	1.44
VZ_US_PIP_013	29049.0	29544.0	14.34	7.48	0.46	0.55
VZ_US_PIP_014	14690.0	14690.0	8.24	8.24	0.33	0.33
VZ_US_PIP_015	14750.0	14525.0	0	8.21	0.17	0.35

Tab. 7.2: Comparación de FLOW con WYH (nodo raíz)

Es importante aclarar que no hicimos comparaciones con otros trabajos que hacen uso de una enumeración previa de ciclos candidatos debido a que esos enfoques no tienen garantizada una solución óptima. Por ejemplo, los resultados presentados en [16] muestran que luego de hacer la preselección de candidatos en redes de tamaño comparable al de VZ_US_PIP_005, la mejor solución que puede obtenerse tiene aproximadamente un 20 % de costo adicional respecto al óptimo.

Tampoco hicimos una comparación de tiempos con redes de gran escala para las cuales la enumeración de ciclos comienza a tornarse impracticable. En [22] se realizó esta comparación con una instancia de 30 nodos y 62 ejes. El resultado fue que utilizando el modelo WYH se logró obtener soluciones de calidad (gap 4.83 %) en un tiempo comparable al necesario para encontrar todos los ciclos simples del grafo (en total 13343782), lo cual demuestra una gran ventaja de estos enfoques sin enumeración de ciclos sobre el modelo presentado en la sección 2.1.1.

7.2. Resultados obtenidos con restricciones adicionales al modelo

A continuación presentaremos los resultados obtenidos a partir de añadir restricciones adicionales a nuestro modelo. En primer lugar hemos agregado las restricciones 5.10 para eliminar soluciones simétricas (notar que no se eliminan todas). Se puede notar que aportan a lograr un mejor desempeño en la tabla 7.3 (FLOW 1).

Además de esto, hemos agregado también las restricciones 5.11 para eliminar las soluciones simétricas por permutación de ciclos. Sin embargo, el tiempo de ejecución para todas las instancias fue excesivo y por eso no lo presentamos aquí. En todos los casos demoró mucho tiempo en encontrar buenas soluciones enteras. A diferencia de lo que venía sucediendo, CPLEX no lograba encontrar soluciones factibles en el nodo raíz, y lo lograba después de procesar muchos nodos (por ejemplo, más de 150 con la intancia COST_239 y más de 1000 con VZ_US_PIP_001). Al final del límite de tiempo impuesto se obtenía en varias instancias un *gap* superior al 10 % y otro punto interesante a destacar es que no se lograba mejorar la cota obtenida por relajaciones lineales del nodo raíz.

En un principio atribuimos este mal desempeño a problemas numéricos, sin embargo, al establecer el valor de los coeficientes $\beta_{(u,v)}$ en 1 (es decir que no se eliminan todas las soluciones simétricas) también se obtenían malos resultados comparables a los anteriores.

Con el modelo WYH también fueron realizadas algunas pruebas preliminares adicionando restricciones para eliminar simetría similares a las presentadas. Tanto las restricciones 5.10 como las 5.11 pueden ser aplicadas a este modelo renombrando las variables adecuadamente. En estas pruebas se obtuvo un peor desempeño para todas las instancias, siendo igual de malo en el caso de las restricciones 5.11. De modo que podemos comparar estos nuevos resultados de la tabla 7.3 con los la tabla 7.1 correspondientes al modelo WYH, notando que ahora se logra una mayor ventaja en todas las instancias.

Además hemos incorporado las restricciones 5.12 al modelo y los resultados se encuentran también en la tabla 7.3 (FLOW 2). Si bien no parece haber un mejor desempeño en general, los resultados son similares a excepción de la instancia VZ_US_PIP_004 para la cual se logró una mejora notoria. La diferencia al agregar estas restricciones ocurre desde la fase de preprocesamiento (previa al *branch and cut*). Las J restricciones agregadas tienen como resultado un mayor aprovechamiento de esta etapa ya que aumenta significativamente el número de coeficientes modificados y cotas ajustadas por CPLEX.

En adelante, las siguientes pruebas se realizarán con estas restricciones (5.10 y 5.12) agregadas a la formulación.

7.3. Resultados aplicando cortes

CPLEX utiliza en su configuración por defecto utiliza muchas desigualdades conocidas como cortes para eliminar soluciones fraccionarias obtenidas en las relajaciones lineales. Durante la ejecución utilizando nuestro modelo logra aplicar muchos cortes, en su mayoría de los tipos *flow* y *cover*. Hemos realizado varias pruebas modificando los parámetros correspondientes para elegir la frecuencia con que se aplican (uso moderado, agresivo, etc.). Aún así no se obtienen mejoras, por el contrario, se ve un desempeño ligeramente peor que al dejar la configuración predeterminada.

Hemos presentado en el capítulo anterior las desigualdades 5.13, las cuales fueron aplicadas a CPLEX como *user cuts*. Esto quiere decir que CPLEX puede utilizar estas

	Función objetivo			Gap (%)		Tiempo (seg.)	
Instancia	Óptimo	FLOW 1	FLOW 2	FLOW 1	FLOW 2	FLOW 1	FLOW 2
COST_239	32340	32340	32340	0.46	0.61	100	100
VZ_US_PIP_001	32240	32320	32300	2.78	2.86	100	100
VZ_US_PIP_002	37370	37370	37370	0	0	1.41	1.96
VZ_US_PIP_003	35170	35170	35170	0	0	63	40.9
VZ_US_PIP_004	39980	40690	40220	6.8	4.38	100	100
VZ_US_PIP_005	24937	24937	24937	0.99	0.72	100	100
VZ_US_PIP_006	26190	26190	26190	0	0	23.2	53.5
VZ_US_PIP_007	30534	30534	30534	0	0	0.61	0.55
VZ_US_PIP_008	32721	32721	32736	1.04	1.61	100	100
VZ_US_PIP_009	37071	37071	37071	0	0	7.43	54.8
VZ_US_PIP_010	44084	44084	44084	0	0	58.2	23.3
VZ_US_PIP_011	42054	42054	42054	0	0	27.4	58.5
VZ_US_PIP_012	35754	35754	35754	0	0	11.1	3.87
VZ_US_PIP_013	29984	29984	29984	0	0	5.85	4.34
VZ_US_PIP_014	14740	14740	14740	0	0	0.31	0.48
VZ_US_PIP_015	14775	14775	14775	0	0	0.25	0.19

Tab. 7.3: Resultados agregando restricciones al modelo — (FLOW 1: (5.10)) — (FLOW 2: (5.10) y (5.12))

desigualdades como cortes tanto en la raíz como en cualquier nodo del árbol. Los resultados se ven en la tabla 7.4. Las columnas "sin cortes" corresponden a los resultados obtenidos anteriormente. Si bien se ve que demora más tiempo en varias instancias que ya eran resueltas por completo anteriormente, también se logra obtener mejores resultados en las instancias más difíciles (como VZ_US_PIP_001 y VZ_US_PIP_004 en las el gap es similar pero se logra mejorar el valor de la función objetivo).

De ahora en más, los próximos *tests* tendrán incluidos estos cortes.

7.4. Prioridad en las variables

Un parámetro importante que hay que determinar en CPLEX es la prioridad de variables para elegir durante el proceso de *branching*. Las variables enteras que tengan valor fraccionario en la relajación lineal serán elegidas para el *branching* respetando la prioridad establecida (si es que existe). Las variables para elegir son r_u^j, b_u^j ($\forall u \in V, j \leq J$) y $z_{(u,v)}^j$ ($\forall (u,v) \in E, j \leq J$).

Estableceremos prioridades con el siguiente criterio: Un posible orden apropiado debe dar prioridad a las variables r_u^j ya que estas determinan el valor de muchas otras. En particular, cuando se establece un valor de 1 para alguna de ellas, las demás variables r del ciclo j serán forzadas a valer 0. Asimismo, las variables b_w^j para $w < u$ también serán forzadas a valer 0. Luego se debe dar prioridad, en segunda instancia, a las variables b_w^j, ya que estas determinan los nodos utilizados para el ciclo. Una vez que están todos elegidos, muchas variables $z_{(u,v)}^j$ tendrán 0 como valor forzado.

Previamente probamos con otros órdenes, obteniendo los resultados esperados, es decir,

	Función objetivo			Gap (%)		Tiempo (seg.)	
Instancia	Óptimo	s/cortes	c/cortes	s/cortes	c/cortes	s/cortes	c/cortes
COST_239	32340	32340	32340	0.61	0.56	100	100
VZ_US_PIP_001	32240	32300	32255	2.86	2.73	100	100
VZ_US_PIP_002	37370	37370	37370	0	0	1.96	1.94
VZ_US_PIP_003	35170	35170	35170	0	0	40.9	80
VZ_US_PIP_004	39980	40220	39980	4.38	4.41	100	100
VZ_US_PIP_005	24937	24937	24937	0.72	0.72	100	100
VZ_US_PIP_006	26190	26190	26190	0	0	53.5	75.77
VZ_US_PIP_007	30534	30534	30534	0	0	0.55	1.17
VZ_US_PIP_008	32721	32736	32721	1.61	0.41	100	100
VZ_US_PIP_009	37071	37071	37071	0	0	54.8	78.77
VZ_US_PIP_010	44084	44084	44084	0	0	23.3	13
VZ_US_PIP_011	42054	42054	42054	0	0	58.5	60.5
VZ_US_PIP_012	35754	35754	35754	0	0	3.87	11.18
VZ_US_PIP_013	29984	29984	29984	0	0	4.34	4.23
VZ_US_PIP_014	14740	14740	14740	0	0	0.48	0.5
VZ_US_PIP_015	14775	14775	14775	0	0	0.19	0.17

Tab. 7.4: Resultados agregando *user cuts*

un desempeño más pobre respecto a la configuración inicial en la que todas las variables tienen la misma prioridad.

De modo que se realizaron más pruebas en las que respetamos el criterio previsto con ciertas variantes:

- Opción 1: Misma prioridad para cada variable r_u^j. Misma prioridad para cada variable b_u^j, pero menor que para r_u^j. Misma prioridad para cada variable $z_{(u,v)}^j$, pero menor que para b_u^j.

- Opción 2: Todas las variables del ciclo j tienen mayor prioridad que todas las del ciclo $j+1$. Para las variables correspondientes a cada ciclo, se respeta la prioridad de la opción 2. Con esto buscamos que se vayan formando los ciclos de la solución de a uno por uno de forma ordenada.

Los resultados están en la tabla 7.5, en la cual se ve que la opción 1 fue la mejor para nuestras instancias. Además, con esto se logró encontrar la solución óptima para todas las instancias dentro del límite establecido y se redujo el *gap* en las instancias que resultaron ser más difíciles (VZ_US_PIP_001 y VZ_US_PIP_004). La opción 2 degradó el rendimiento respecto a la configuración predeterminada con todas las variables con las misma prioridad.

Las pruebas siguientes tendrán establecida la prioridad según el criterio explicado en la opción 1.

	Función objetivo			Gap (%)		Tiempo (seg.)	
Instancia	Óptimo	opc.1	opc.2	opc.1	opc.2	opc.1	opc.2
COST_239	32340	32340	32340	0	0.62	79.6	100
VZ_US_PIP_001	32240	32240	32400	1.24	3.16	100	100
VZ_US_PIP_002	37370	37370	37370	0	0	2.22	3.14
VZ_US_PIP_003	35170	35170	35170	0	1	33.7	100
VZ_US_PIP_004	39980	39980	40090	0.77	4.23	100	100
VZ_US_PIP_005	24937	24937	24937	0	1.23	27.5	100
VZ_US_PIP_006	26190	26190	26190	0	0.82	17.59	100
VZ_US_PIP_007	30534	30534	30534	0	0	0.93	2.34
VZ_US_PIP_008	32721	32721	32736	0	2.65	82	100
VZ_US_PIP_009	37071	37071	37071	0	0.42	47.2	100
VZ_US_PIP_010	44084	44084	44084	0	1.04	6.85	100
VZ_US_PIP_011	42054	42054	42054	0	0.19	34.3	100
VZ_US_PIP_012	35754	35754	35754	0	0.79	7.79	100
VZ_US_PIP_013	29984	29984	29984	0	0	4.45	44.5
VZ_US_PIP_014	14740	14740	14740	0	0	0.49	0.5
VZ_US_PIP_015	14775	14775	14775	0	0	0.17	0.19

Tab. 7.5: Resultados estableciendo prioridades en las variables

7.5. Otros parámetros sobre selección de variables

Hemos realizado más experimentos modificando otros parámetros para variar las estrategias de branching. Uno de estos parámetros es la dirección de *branching*: Una vez elegida una variable entera que tiene un valor fraccionario en la relajación lineal, se tienen dos subproblemas nuevos en los que se ponen como cotas (para dicha variable) su valor redondeado hacia abajo y hacia arriba (cota superior e inferior respectivamente). En este caso que tenemos sólo variables binarias, lo que se hace es fijar el valor de 0 para la variable en cuestión en uno de los subproblemas, y de 1 en el otro.

En la tabla 7.6 se encuentran los resultados obtenidos al fijar la dirección de *branching*. La mejor solución entera encontrada es la óptima en todos los casos a excepción de los casos marcados con asterisco (*) en la columna de *gap*. Se puede ver que no se logra ningún beneficio forzando una dirección particular y la mejor opción es dejar decidir a CPLEX (configuración predeterminada).

Otro parámetro a determinar es la regla de selección de variable de acuerdo a su valor fraccionario. Las opciones probadas fueron las siguientes:

- *Minimum infeasibility*: Se escoge la variable v cuyo valor fraccionario sea más próximo a un valor entero, o sea que cumpla que $(\lceil v \rceil - v)$ o $(v - \lfloor v \rfloor)$ sea mínimo. En caso de variables binarias a 0 ó 1.

- *Maximum infeasibility*: Lo contrario al caso anterior. Se elige la variable v cuyo valor esté más próximo a $(\lceil v \rceil + \lfloor v \rfloor)/2$. En caso de variables binarias a 0.5.

- *Based on pseudo costs*: Selección de variable basada en pseudo-costos que son derivados de los precios sombra.

	Gap (%)			Tiempo (seg.)		
Instancia	Down	Up	Auto	Down	Up	Auto
COST_239	0.34	0.34	0	100	100	79.6
VZ_US_PIP_001	1.55(*)	1.14	1.24	100	100	100
VZ_US_PIP_002	0	0	0	2.23	1.86	2.22
VZ_US_PIP_003	0	0	0	15.6	14.6	33.7
VZ_US_PIP_004	10.86(*)	3.29(*)	0.77	100	100	100
VZ_US_PIP_005	0.63	0	0	100	52	27.5
VZ_US_PIP_006	0	0	0	11	22	17.6
VZ_US_PIP_007	0	0	0	0.81	0.85	0.93
VZ_US_PIP_008	0.28	0	0	100	66.74	82
VZ_US_PIP_009	0	0	0	58.6	38.4	47.2
VZ_US_PIP_010	0	0	0	37.7	17.5	6.85
VZ_US_PIP_011	0	0	0	53.9	43.3	34.3
VZ_US_PIP_012	0	0	0	6.66	5.82	7.79
VZ_US_PIP_013	0	0	0	2.57	2.8	4.45
VZ_US_PIP_014	0	0	0	0.49	0.51	0.5
VZ_US_PIP_015	0	0	0	0.18	0.18	0.19

Tab. 7.6: Resultados fijando una dirección de *branching*

- *Strong branching*: Consiste en resolver parcialmente un número de subproblemas con *branch* tentativos para ver cual de todos es el más prometedor.

Los resultados obtenidos se encuentran en la tabla 7.7. Aquí también los casos en los que no se logró alcanzar una solución entera óptima están marcados con un (*) en la columna de *gap*. Nuevamente, observamos que no se logra una mejora respecto a la configuración predeterminada, la cual consiste en permitir a CPLEX elegir la mejor estrategia basada en el problema particular y al progreso de la resolución (Observar los resultados obtenidos antes en la columna "Auto" de la tabla 7.6).

7.6. Estrategias de selección de nodos

Otro parámetro de CPLEX (y de cualquier implementación de branch-and-bound) para determinar es la estrategia de selección de nodo. Esta será la regla empleada para decidir sobre que nodo continúa el procedimiento cuando se regresa al árbol cuando un nodo es infactible o se poda. Las distintas alternativas probadas fueron:

- *Deph first search(DFS)*: Escoge el nodo creado más recientemente.
- *Best Bound*: Elige el nodo con el mejor valor valor de función objetivo en la relajación lineal asociada. Es la estrategia que toma CPLEX por defecto y por lo tanto la que veníamos usando hasta el momento.
- *Best Estimate*: Selecciona el nodo con la mejor estimación del valor de la función objetivo para una solución entera que sería obtenido de un nodo una vez que todas las infactibilidades de integralidad sean removidas.

	Gap (%)				Tiempo (seg.)			
Instancia	min. i.	max. i.	p. c.	s. b.	min. i.	max. i.	p. c.	s. b.
COST_239	1.18	0.78	0.13	0.9(*)	100	100	100	100
VZ_US_PIP_001	1.63	1.01	1.04	1.6	100	100	100	100
VZ_US_PIP_002	0	0	0	0	2.49	2.06	1.65	4.92
VZ_US_PIP_003	0	0	0	0	67.8	16.0	26.5	45.4
VZ_US_PIP_004	4.29(*)	1.91	0.19	4.73(*)	100	100	100	100
VZ_US_PIP_005	0.58	0	0.38	1.48	100	97.0	100	100
VZ_US_PIP_006	0	0	0	0	13.5	10.6	10.2	13.7
VZ_US_PIP_007	0	0	0	0	0.92	0.77	0.95	1.45
VZ_US_PIP_008	0.3	0.28	0.03	0.58	100	100	100	100
VZ_US_PIP_009	0.28	0	0	0.42	100	38.7	40.2	100
VZ_US_PIP_010	0	0	0	0	28.0	13.1	16.3	64.1
VZ_US_PIP_011	0.02	0	0	0.02	100	46.0	27.9	100
VZ_US_PIP_012	0	0	0	0	15.3	7.27	9.48	38.5
VZ_US_PIP_013	0	0	0	0	5.11	5.31	4.57	8.01
VZ_US_PIP_014	0	0	0	0	0.47	0.39	0.63	0.54
VZ_US_PIP_015	0	0	0	0	0.18	0.17	0.2	0.19

Tab. 7.7: Resultados fijando variable para *branching* — min. i.: minimum infeasibility — max. i.: maximum infeasibility — p. c.: pseudo costs — s. b.: strong branching

Los resultados obtenidos están en la tabla 7.8 y se puede notar que el mejor rendimiento se obtiene con la configuración por defecto (Best Bound). Nuevamente, los casos en los que no se encontró una solución óptima son los marcados con (*).

7.7. Heurísticas

Periódicamente CPLEX puede aplicar una heurística que intenta obtener una solución entera a partir de la información disponible, como por ejemplo de la solución a la relajación lineal del nodo en proceso. El objetivo es intentar encontrar soluciones factibles de una manera simple y económica (en poco tiempo). Es posible fijar la frecuencia con la que se aplica esta heurística, es decir cada cuantos nodos se utiliza.

Otra heurística implementada por CPLEX se llama *relaxation induced neighborhood search* (RINS). Esta heurística intenta mejorar la solución entera obtenida con mejor valor de función objetivo, por lo que se aplica solamente una vez que se encontró al menos una solución factible. Se declara en el manual que es una heurística muy buena para encontrar soluciones factibles de alta calidad aunque puede ser muy costosa (en tiempo). También es posible establecer la frecuencia de aplicación.

Probamos alterar la frecuencia de ambas heurísticas (cada cuantos nodos se aplica) estableciendo distintos valores (5, 10, 20, 30, 50). No presentamos los resultados ya que no fueron buenos, en todas las instancias se demoraba mucho más la ejecución comparado con la configuración automática.

	Gap (%)			Tiempo (seg.)		
Instancia	DFS	B. Est.	B. Bound	DFS	B. Est.	B. Bound
COST_239	1.62	1.62	0	100	100	79.6
VZ_US_PIP_001	4.06(*)	4.17(*)	1.24	100	100	100
VZ_US_PIP_002	0	0	0	1.6	2.1	2.22
VZ_US_PIP_003	0	0	0	37.85	11.15	33.7
VZ_US_PIP_004	5.89	4.53(*)	0.77	100	100	100
VZ_US_PIP_005	0.51	0	0	100	39.9	27.5
VZ_US_PIP_006	0	0	0	15.1	16.8	17.6
VZ_US_PIP_007	0	0	0	0.88	0.84	0.93
VZ_US_PIP_008	0	0	0	80.2	43.8	82
VZ_US_PIP_009	0	0	0	37.3	68	47.2
VZ_US_PIP_010	0	0	0	25.9	7.39	6.85
VZ_US_PIP_011	0	0	0	26.8	52.9	34.3
VZ_US_PIP_012	0	0	0	7.11	7.15	7.79
VZ_US_PIP_013	0	0	0	4.8	6	4.45
VZ_US_PIP_014	0	0	0	0.48	0.48	0.5
VZ_US_PIP_015	0	0	0	0.18	0.17	0.19

Tab. 7.8: Resultados variando estrategias de selección de nodos

7.8. Parámetro de Énfasis

CPLEX provee un parámetro (*MIP Emphasis*) para determinar el balance que hará durante el proceso de optimización entre encontrar soluciones enteras factibles y probar la optimalidad de la mejor solución factible encontrada

Las opciones testeadas fueron las siguientes:

- **BALANCED**: Es la opción por defecto y por lo tanto la que veníamos utilizando. CPLEX hacen un balance parejo entre demostrar lo más rápido posible la optimalidad de una solución, y encontrar pronto soluciones factibles de calidad.
- **FEASIBILITY**: CPLEX pone más esfuerzo en encontrar soluciones factibles, pagando el costo de demorar más en encontrar la solución óptima.
- **OPTIMALITY**: Se pone menos esfuerzo en encontrar anticipadamente soluciones factibles.
- **BESTBOUND**: Se hace aún más esfuerzo en probar optimalidad buscando siempre en función de las mejores cotas obtenidas, haciendo práctimamente accidental el hecho de hallar soluciones factibles durante este proceso.

La tabla 7.9 tiene los resultados obtenidos. La opción por defecto (no se muestra aquí pero puede verse en la columna "B.Bound" de la tabla 7.8) es la que claramente da los mejores resultados y nos indica lo importante que es mantener el balance con nuestro modelo. La opción FEASIBILITY encuentra soluciones enteras de calidad ligeramente más rápido que la opción BALANCED, pero la diferencia no es significativa. La opción

por defecto es la mejor para encontrar soluciones óptimas y es muy buena para hallar soluciones de calidad anticipadamente.

	Gap (%)			Tiempo (seg.)		
Instancia	Feas.	Opt.	B. Bound	Feas.	Opt.	B. Bound
COST_239	1.74(*)	0.37(*)	0	100	100	91.5
VZ_US_PIP_001	3.94	0.74(*)	54.11(*)	100	100	100
VZ_US_PIP_002	0	0	0	1.95	4.3	14.6
VZ_US_PIP_003	0	0	0	10.0	24.1	92.9
VZ_US_PIP_004	7.68(*)	10.89(*)	21.28(*)	100	100	100
VZ_US_PIP_005	0.72	0.15	0.72	100	100	100
VZ_US_PIP_006	0	0	0	6.13	6.96	66.3
VZ_US_PIP_007	0	0	0	0.74	0.81	6.15
VZ_US_PIP_008	0.28	0	1.33(*)	100	57.0	100
VZ_US_PIP_009	0	0	0.21	57.2	78.1	100
VZ_US_PIP_010	0	0	0	10.4	12.2	79.0
VZ_US_PIP_011	0.02	0	0.24(*)	100	68.5	100
VZ_US_PIP_012	0	0	0	14.9	10.5	34.1
VZ_US_PIP_013	0	0	0	6.16	6.6	12.7
VZ_US_PIP_014	0	0	0	0.4	0.45	1.42
VZ_US_PIP_015	0	0	0	0.12	0.22	0.4

Tab. 7.9: Resultados variando el parámetro de énfasis

7.9. *Probing*

Probing es una técnica aplicada por CPLEX entre el preprocesamiento y la solución de la relajación lineal en la raíz del árbol. Deriva implicaciones lógicas de fijar el valor de cada variable binaria a 0 ó 1. Puede consumir mucho tiempo, pero se espera que esta inversión inicial sea beneficiosa para modelos difíciles, aunque también puede incrementar el tiempo total del proceso de optimización.

CPLEX permite fijar el nivel de aplicación de *probing*. En la tabla 7.10 se encuentran los resultados obtenidos fijando los distintos niveles disponibles. Estos resultados muestran que con nuestro modelo no se logra un mejor desempeño con algún nivel particular de *probing* y es mejor dejar este parámetro establecido en su configuración predeterminada.

7.10. *Dynamic search* vs *branch-and-cut* clásico

A partir de la versión 11 de CPLEX se introdujo una nueva metodología llamada *dynamic search* que busca ser un método general de búsqueda y puede ser desactivado para utilizar el algoritmo branch and cut estándar. Realizamos pruebas con CPLEX 12.4 desactivando *dynamic search* para verificar si nuestro modelo se beneficiaba con esta característica. En la tabla 7.11 se encuentran los resultados obtenidos, donde se ve claramente que el desempeño de *branch-and-cut* convencional es peor.

	Gap (%)				Tiempo (seg.)			
Instancia	des.	mod.	agr.	m.a.	des.	mod.	agr.	m.a.
COST_239	1.05	0.22	0.22	0.19	100	100	100	100
VZ_US_PIP_001	1.15	0	0	2.99(*)	100	92.6	94.6	100
VZ_US_PIP_002	0	0	0	0	1.55	1.81	1.47	3.91
VZ_US_PIP_003	0	0	0	0	34.7	33.2	32.6	36.5
VZ_US_PIP_004	7.33(*)	2.77(*)	2.23(*)	2.69	100	100	100	100
VZ_US_PIP_005	0	0	0	0	71.2	76.4	32.6	92.7
VZ_US_PIP_006	0	0	0	0	18.3	9.65	6.45	7.38
VZ_US_PIP_007	0	0	0	0	0.86	0.91	0.59	0.59
VZ_US_PIP_008	0	0	0	0	57.9	64.8	57.2	38.1
VZ_US_PIP_009	0	0	0	0	27.7	27.4	18.9	11.5
VZ_US_PIP_010	0.83	0	0	0	100	9.07	4.8	4.69
VZ_US_PIP_011	0	0	0	0	31.3	33.8	45.8	47.8
VZ_US_PIP_012	0	0	0	0	24.1	4.63	2.16	1.92
VZ_US_PIP_013	0	0	0	0	3.28	4.01	2.88	3.36
VZ_US_PIP_014	0	0	0	0	0.35	0.48	0.37	0.3
VZ_US_PIP_015	0	0	0	0	0.23	0.17	0.11	0.2

Tab. 7.10: Resultados fijando nivel de *probing* — des.: Desactivado — mod.:Moderado — agr.:Agresivo — m.a.:Muy agresivo

También realizamos esta prueba con el modelo WYH obteniendo el mismo resultado (mejor rendimiento con *dynamic search*).

7.11. Resumen de resultados

A partir de la experimentación realizada llegamos a la conclusión que los mejores resultados fueron obtenidos con nuestro modelo agregando como parte de la formulación las restricciones 5.10 y 5.12, introduciendo las desigualdades 5.13 como *user cuts* y estableciendo la siguiente prioridad de *branching* para las variables enteras:

- Misma prioridad para cada variable r_u^j.
- Misma prioridad para cada variable b_u^j, pero menor que para r_u^j.
- Misma prioridad para cada variable $z_{(u,v)}^j$, pero menor que para b_u^j.

El resto de los parámetros se dejaron en su configuración predeterminada.

Cabe aclarar que esto es dependiente de las instancias utilizadas y que CPLEX tiene muchos más parámetros para modificar, pero aún así estamos en condiciones de decir que comparando con los resultados obtenidos con los mejores modelos existentes (tomamos como referencia el de [22]), logramos una gran mejora.

7.12. Resultados de la heurística sin enumeración de ciclos para SCA

A continuación presentaremos los resultados obtenidos con la heurística introducida en el capítulo 6. Implementamos el Algoritmo 8 y lo utilizamos con los modelos MCIDA y

	Gap (%)		Tiempo (seg.)	
Instancia	B&C conv.	Dyn. Search	B&C conv.	Dyn. Search
COST_239	1.05(*)	0	100	79.6
VZ_US_PIP_001	2.97	1.24	100	100
VZ_US_PIP_002	0	0	1.64	2.22
VZ_US_PIP_003	0	0	28.3	33.7
VZ_US_PIP_004	8.89(*)	0.77	100	100
VZ_US_PIP_005	4.53	0	100	27.5
VZ_US_PIP_006	0	0	11.2	17.6
VZ_US_PIP_007	0	0	1.06	0.93
VZ_US_PIP_008	1.18	0	100	82.0
VZ_US_PIP_009	0	0	43.5	47.2
VZ_US_PIP_010	0	0	18.0	6.85
VZ_US_PIP_011	0	0	18.3	34.4
VZ_US_PIP_012	0	0	13.3	7.79
VZ_US_PIP_013	0	0	4.11	4.45
VZ_US_PIP_014	0	0	0.74	0.49
VZ_US_PIP_015	0	0	0.26	0.17

Tab. 7.11: Dynamic search y *branch and cut* convencional

MHEUR(ω). No impusimos límite de tiempo a CPLEX para resolver los modelos MCIDA y MHEUR en cada iteración. Entonces en ambos casos en cada iteración se obtienen las soluciones óptimas. Por lo tanto con el modelo MCIDA se obtienen los mismos resultados que utilizando la heurística CIDA original.

Para el modelo MCIDA establecimos los costos de cada eje del siguiente modo:

$$c_{(u,v)} = \frac{|V|}{2} \times \frac{c'_{(u,v)}}{c_{MAX}}$$

donde $c'_{(u,v)}$ es el costo de entrada (el original de la instancia) del eje (u, v) y c_{MAX} es el costo máximo de entrada. Esto lo hicimos para evitar problemas numéricos que retrasarían la ejecución debido a valores muy pequeños de las cotas U y L. Estas constantes deben ser cotas para la inversa del costo de los ciclos. Para la cota superior U calculamos la inversa de la suma de los 3 costos más bajos (cota inferior para el costo, superior para su inversa). Y a L le asignamos la inversa de la suma de los $|V|$ costos más altos (cota superior para el costo, inferior para su inversa). Si suponemos que la lista $[c_1, c_2, \ldots, c_{|E|}]$ contiene los costos ordenados en forma ascendente, entonces tenemos:

$$U = \left(\sum_{i=1}^{3} c_i \right)^{-1}$$

$$L = \left(\sum_{i=|E|-|V|+1}^{|E|} c_i \right)^{-1}$$

Instancia	MCIDA	MHEUR(0.1)	MHEUR(1)	MHEUR(3)	MHEUR(5)
COST_239	9.2	17.19	17.19	10.71	13.22
VZ_US_PIP_001	14.03	30.96	30.96	27.45	23.47
VZ_US_PIP_002	15.62	29.58	27.05	21.85	15.16
VZ_US_PIP_003	16.82	20.72	14.54	11.51	8.58
VZ_US_PIP_004	9.89	18.55	16.76	19.48	16.11
VZ_US_PIP_005	14.3	33.33	26.75	19.51	16.59
VZ_US_PIP_006	11.46	19.55	19.55	19.55	17.31
VZ_US_PIP_007	6.44	29.91	29.91	21.97	22.96
VZ_US_PIP_008	15.8	14.41	14.41	14.14	12.68
VZ_US_PIP_009	14.72	17.08	17.08	16.57	12.89
VZ_US_PIP_010	13.43	18.06	18.06	13.71	9.05
VZ_US_PIP_011	20.01	18.66	18.66	12.09	8.95
VZ_US_PIP_012	10.99	19.16	19.16	8.08	7.91
VZ_US_PIP_013	5.76	12.42	12.42	10.83	11.13
VZ_US_PIP_014	14.92	11.23	11.23	0	0
VZ_US_PIP_015	6.28	8.54	8.54	8.25	9.66

Tab. 7.12: Calidad de las soluciones obtenidas (% respecto al óptimo)

Además, agregamos las desigualdades 7.1 como *user cuts* para acelerar la ejecución.

$$y_{(u,v)} \leq \frac{1}{2} \times \Big(\sum_{(u,w)\in E, w\neq v} z_{(u,w)} + \sum_{(v,w)\in E, w\neq u} z_{(v,w)} \Big) + \frac{1}{2} \times z_{(u,v)} \qquad \forall (u,v) \in E \quad (7.1)$$

Para el modelo alternativo MHEUR(ω) establecimos los valores de ω en $0.1/c_{MAX}$, $1/c_{MAX}$, $3/c_{MAX}$ y $5/c_{MAX}$ (En las tablas de datos omitiremos el denominador c_{MAX} por cuestión de espacio). También agregamos las desigualdades 7.2 como *user cuts*.

$$y_{(u,v)} \leq \frac{1}{2} \times \Big(\sum_{(u,w)\in E, w\neq v} z_{(u,w)} + \sum_{(v,w)\in E, w\neq u} z_{(v,w)} \Big) \qquad \forall (u,v) \in E \qquad (7.2)$$

En la tabla 7.13 se presentan los resultados obtenidos respecto a la calidad de solución. El porcentaje reportado es el costo adicional respecto a una solución óptima (cuanto más bajo mejor):

$$100 \times \frac{\text{costo obtenido} - \text{costo óptimo}}{\text{costo óptimo}}$$

Con el modelo MCIDA se obtienen en general soluciones de mejor calidad. Las soluciones con MHEUR dependen del coeficiente ω, y en la mayoría de los casos no logran superar a CIDA. Sin embargo el tiempo necesario para MCIDA es significativamente mayor que para MHEUR. En la tabla 7.13 vemos los resultados de tiempos comparando ambos modelos. Solo reportamos el tiempo para un valor de ω porque cambiando este valor no hay cambios en el tiempo.

Otra cosa interesante para observar es la cantidad de ciclos obtenida en la solución. Esto se encuentra en la tabla 7.14. Vemos que con ambos modelos tenemos una buena

Instancia	MCIDA	MHEUR(1)
COST_239	6.75	0.21
VZ_US_PIP_001	132.6	0.17
VZ_US_PIP_002	24.88	0.34
VZ_US_PIP_003	21.81	0.18
VZ_US_PIP_004	33.88	0.7
VZ_US_PIP_005	18.33	0.33
VZ_US_PIP_006	19.57	0.22
VZ_US_PIP_007	18.2	0.17
VZ_US_PIP_008	39.2	0.22
VZ_US_PIP_009	39.49	0.26
VZ_US_PIP_010	35.85	0.39
VZ_US_PIP_011	4.04	0.33
VZ_US_PIP_012	6.92	0.25
VZ_US_PIP_013	1.48	0.22
VZ_US_PIP_014	0.57	0.08
VZ_US_PIP_015	0.33	0.06

Tab. 7.13: Tiempo en segundos de la heurística utilizando ambos modelos

aproximación para el valor J apropiado en muchas de las instancias. Sabíamos que con CIDA esto era así. Pero además vemos que con el otro modelo también se logra una buena aproximación.

Instancia	Óptimo	MCIDA	MHEUR(0.1)	MHEUR(1)	MHEUR(3)	MHEUR(5)
COST_239	7	8	6	6	7	9
VZ_US_PIP_001	7	10	6	6	6	7
VZ_US_PIP_002	6	9	6	6	6	6
VZ_US_PIP_003	7	10	6	6	6	6
VZ_US_PIP_004	8	9	6	7	8	9
VZ_US_PIP_005	7	12	6	6	7	8
VZ_US_PIP_006	6	7	6	6	6	7
VZ_US_PIP_007	7	9	7	7	8	8
VZ_US_PIP_008	7	10	6	6	7	7
VZ_US_PIP_009	7	10	7	7	9	8
VZ_US_PIP_010	8	12	8	8	8	8
VZ_US_PIP_011	8	12	8	8	9	10
VZ_US_PIP_012	8	10	8	8	8	10
VZ_US_PIP_013	7	8	7	7	10	11
VZ_US_PIP_014	6	6	5	5	6	6
VZ_US_PIP_015	6	6	6	6	7	8

Tab. 7.14: Cantidad de ciclos de la solución obtenida

8. CONCLUSIONES Y TRABAJO FUTURO

8.1. Conclusiones

En este trabajo nos enfocamos en la resolución del problema SCA sobre diseño de redes de comunicaciones con requerimientos de supervivencia utilizando el concepto de p-ciclos.

Luego de revisar la literatura sobre el problema SCA y realizar una resolución preliminar, notamos la importancia de abordar la resolución del mismo evitando hacer una enumeración explícita de ciclos candidatos. Teniendo esto último en consideración, introducimos un nuevo modelo de programación lineal entera para SCA.

También introducimos algunas mejoras al modelo, como restricciones para evitar soluciones simétricas y cortes, y realizamos experimentación con CPLEX 12.4 para evaluar su desempeño, utilizando instancias basadas en redes ópticas reales.

Durante la experimentación, evaluamos el comportamiento variando estrategias de *branching* y selección de nodos, entre otras cosas. Si bien no obtuvimos mejorar el desempeño cambiando la mayoría de las configuraciones por defecto de CPLEX (a excepción de establecer prioridades sobre las variables), logramos mejorar ampliamente los resultados obtenidos con otros modelos (sin enumeración de ciclos) existentes en la literatura. Además nuestro modelo brinda la posibilidad de encontrar soluciones factibles de calidad rápidamente, lo cual lo hace aplicable en la práctica a instancias de gran escala en las que no se requiera obtener necesariamente la solución óptima en cuanto al costo.

A partir de nuestro modelo y de la heurística CIDA introducida en [2], presentamos una heurística basada en PLE que no hace enumeración de ciclos candidatos. Para esto presentamos dos modelos alternativos. Con el primero se obtienen los mismos resultados que con el algoritmo CIDA pagando el costo de un alto tiempo de ejecución. Con el segundo se obtienen soluciones para SCA de menor calidad que con CIDA, pero el tiempo necesario para esto es mucho menor. En general, en ambos casos las soluciones tienen una cantidad de ciclos similar a las soluciones óptimas, lo cual nos brinda una aproximación para el valor J que debe ser determinado arbitrariamente en la práctica.

8.2. Trabajo futuro

Si bien se logró superar los resultados obtenidos por otros trabajos, aún tenemos mucho trabajo para realizar referido a SCA. En primer lugar, buscar más mejoras al modelo presentado (cortes, etc.) para reducir los tiempos de ejecución. Además aún tenemos el problema de la simetría referida a la permutación de ciclos en la solución, que no pudimos resolver con la introducción de restricciones adicionales al modelo.

La experimentación que nos quedó pendiente es utilizar las soluciones obtenidas con la heurística presentada en el capítulo 6 a partir del modelo MHEUR como solución inicial para resolver SCA con CPLEX con el modelo FLOW.

Además, este enfoque heurístico introducido que combina un simple algoritmo goloso con PLE, no tiene la gran desventaja de depender de un conjunto de ciclos candidatos. No encontramos un trabajo similar en la literatura. Sería interesante trabajar en optimizar su rendimiento para resolver instancias de gran escala.

Por otro lado, sería interesante trabajar para encontrar cotas ajustadas para J, que es

el valor máximo de ciclos para la solución que puede ser obtenida. No hemos encontrado trabajo sobre esto, y tener una cota representaría que tener que determinar arbitrariamente su valor no sea visto como una desventaja (un valor J más alto de lo necesario perjudica el tiempo de ejecución porque implica más variables y restricciones innecesarias).

Otra tarea pendiente es extender nuestro modelo para resolver el problema JCP que fue mencionado en la sección 2.2. En este problema las demandas de cada eje no son un dato de entrada y se deben establecer conjuntamente con la capacidad redundante. Esto fue hecho en [22] a partir del modelo WYH con el cual comparamos nuestro modelo en el capítulo 7, de manera que contamos con un trabajo previo para comparar resultados.

Bibliografía

[1] R. Asthana, Y. N. Singh, and W. D. Grover. p-Cycles: An overview. *IEEE Communications Surveys & Tutorials, VOL 12, NO 1*, 2010.

[2] J. Doucette, D. He, W. D. Grover, and O. Yang. Algorithmic Approaches for Efficient Enumeration of Candidate p-Cycles and Capacitated p-Cycle Network Design. *Fourth International Workshop on Design of Reliable Communication Networks, 2003. (DRCN 2003). Proceedings.*, 2003.

[3] T. A. Feo and M. Resende. Greedy Randomized Adaptative Search Procedures. *Journal of Global Optimization, 6, 109-134*, 1995.

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

[5] W. D. Grover. *Mesh-Based Survivable Networks: Options and Strategies for Optical, MPLS, SONET, and ATM Networking.* Prentice Hall PTR, 2003.

[6] W. D. Grover and J. Doucette. Advances in Optical Network Design with p-Cycles: Joint Optimization and Pre-Selection of Candidate p-Cycles. *Proceedings of IEEE/LEOS Summer Topicals 2002, Mont Tremblant, PQ, pp.49-50 (paper WA2)*, July 2002.

[7] W. D. Grover, J. Doucette, A. Kodian, D. Leung, A. Sack, M. Clouqueur, and G. Shen. Design of survivable networks based on p-cycles. In M. Resende and P. Pardalos, editors, *Handbook of Optimization in Telecommunications.* Spinger Science, 2006.

[8] W. D. Grover and A. Kodain. Failure-Independent Path Protection with p-Cycles: Efficient, Fast and Simple Protection for Transparent Optical Networks. *Transparent Optical Networks, 2005, Proceedings of 2005 7th International Conference*, 2005.

[9] W. D. Grover and D. P. Onguetou. Towards Solution of Very Large p-Cycle Network Design Problems with Combined GA-ILP Heuristic Methods. *Fifth Workshop on Optimization of Optical Networks (OON 2008)*, 2008.

[10] W. D. Grover and D. P. Onguetou. Solution of a 200-node p-Cycle Network Design Problem with GA-based Pre-Selection of Candidate Structures. *IEEE ICC 2009 Proceedings*, 2009.

[11] W. D. Grover and D. Stamatelakis. Cycle oriented distributed preconfiguration: ring-like speed with mesh-like capacity for self-planning network restoration. *IEEE ICC '98, vol. 1, pp. 537-543*, 1998.

[12] W. D. Grover and D. Stamatelakis. Bridging the Ring-Mesh Dichotomy With P-Cycles. *Proceedings of Design of Reliable Communication Networks (DRCN 2000), Munich, Germany, April 9-12, 2000, Session 5*, 2000.

[13] IBM ILOG CPLEX Optimizer. `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`.

[14] B. Jaumard, C. Rocha, D. Baloukov, and W.D. Grover. A Column Generation Approach for Design of Networks using Path-Protecting p-Cycles. *Proceedings of the 6th IEEE International Workshop on Design and Reliable Communication Networks (DRCN'07)*, 2007.

[15] A. Kodian, A. Sack, and W. D. Grover. p-Cycle Network Design with Hop Limits and Circumference Limits. *First International Conference on Broadband Networks. (BroadNets 2004). Proceedings.*, 2004.

[16] C. Liu and L. Ruan. Finding good candidate cycles for efficient p-Cycle network design. *13th International Conference on Computer Communications and Networks. ICCCN 2004. Proceedings.*, 2004.

[17] K. Lo, D. Habibi, Q. V. Phung, A. Rassau, and H. N. Nguyen. Efficient p-cycles design by heuristic p-cycle selection and refinement for survivable WDM mesh network. *Global Telecommunications Conference. GLOBECOM '06. IEEE*, 2006.

[18] A. Pecorari and I. Loiseau. Algorithms and Models for P-Cycle Design Without Cycle Enumeration. *Congreso Latino-Iberoamericano de Investigación Operativa*, 2012.

[19] C. R. Reeves. Genetic Algorithms. In M. Gendreau and J. Potvin, editors, *Handbook of Metaheuristics*. Spinger Science, 2010.

[20] M. Resende, C. Ribeiro, F. Glover, and R. Martí. Scatter Search and Path-Relinking: Fundamentals, Advances, and Applications. In M. Gendreau and J. Potvin, editors, *Handbook of Metaheuristics*. Spinger Science, 2010.

[21] D. A. Schupke. An ILP for optimal p-cycle selection without candidate cycle enumeration. *8th Conference on Optical Network Design and Modelling ONDM'04*, 2004.

[22] B. Wu, K. L. Yeung, and P. Ho. Ilp formulations for p-cycle design without candidate cycle enumeration. *IEEE/ACM Trans. Netw.*, 18(1):284–295, February 2010.

[23] H. Zhang and O. Yang. Finding protection cycles in DWDM networks. In *IEEE International Conference on Communications. ICC 2002.*, volume 5, pages 2756 – 2760, 2002.