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

Una Variación del Problema de Ordenamiento Lineal

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

Brian Curcio

Directoras: Isabel Méndez-Díaz, Paula Zabala

Buenos Aires, 2013

Una Variación del Problema de Ordenamiento Lineal

El objetivo de esta tesis es desarrollar un algoritmo exacto para una variante del problema de ordenamiento lineal basado en un modelo de programación lineal entera.

El problema de ordenamiento lineal es un problema clásico de optimización combinatoria estudiado desde hace más de 50 años. Los diversos escenarios reales que provienen de situaciones surgidas en variados campos de la industria y otros sectores, han dado origen a distintas variaciones de este problema.

La gran cantidad de aplicaciones que poseen los problemas de esta familia hacen que los mismos no sólo tengan interés teórico, sino también, una gran importancia práctica.

La versión del problema abordada en este trabajo consiste en encontrar un orden total de elementos de manera que se minimice la suma de las penalidades entre pares de ellos, donde la penalidad se define como el producto entre la distancia a la que se encuentran los dos elementos y un costo asociado a ellos. Este problema recibe el nombre de Problema de Ordenamiento Lineal con Penalidades *POLP*.

POLP pertenece a la clase de problemas NP-Difícil. Para estos problemas no se conocen algoritmos que encuentren la solución en tiempo polinomial.

Como muchos de los problemas de Optimización Combinatoria, *POLP* puede ser modelado mediante formulaciones de programación lineal entera o entera mixta.

Los algoritmos *Branch-and-Cut* son una de las herramientas más efectivas que se conoce para resolver un modelo de programación lineal entera. Especialmente las implementaciones basadas en combinatoria poliedral han permitido incrementar el tamaño de las instancias resueltas.

En el transcurso del trabajo se analizan diversas formulaciones de programación lineal entera que permiten modelar el problema. Luego de seleccionar la más prometedora según ciertos criterios, se realiza un estudio poliedral para determinar características de la cápsula convexa del conjunto de soluciones factibles.

En base a estas familias de desigualdades válidas, se desarrolló e implementó un algoritmo *Branch-and-Cut* para resolver el problema. También se consideraron factores decisivos en la eficiencia de este tipo de algoritmos, como la incorporación de heurísticas iniciales y primales, distintas estrategias de selección de variable de *branching*, y esquemas de recorrido del árbol de *Branch-and-Bound*.

Finalmente se muestran resultados experimentales sobre instancias de prueba que permiten evaluar la eficiencia del algoritmo desarrollado.

Palabras claves: Optimización Combinatoria, Ordenamiento Lineal, Programación Lineal Entera, Estudio poliedral, Branch and Cut

Índice general

1.. Introducción 1

2.. Preliminares 5
2.1. Introducción a la Programación Lineal Entera 5
2.1.1. Definición 5
2.1.2. Formulaciones 6
2.1.3. Algoritmos para Problemas de Programación Lineal Entera 8
2.2. Definiciones sobre grafos 15

3.. Modelo 17
3.1. Modelos previos 17
3.1.1. Ordenamiento lineal 17
3.1.2. Etiquetado lineal mínimo 18
3.1.3. Viajante de comercio 19
3.2. Modelos propuestos 21
3.2.1. Modelo 1 - Adaptación *POL* 22
3.2.2. Modelo 2 - Adaptación *ELM* 23
3.2.3. Modelo 3 - Adaptación *MTZ* 23
3.2.4. Modelo 4 - Adaptación *SD* 24
3.3. Comparación y elección de modelo 26

4.. Estudio Poliedral 31
4.1. Modelo elegido 31
4.2. Igualdades del Sistema Minimal 32
4.3. Desigualdades Válidas 32
4.4. Desigualdades del poliedro del *POL* 37

5.. Algoritmo Branch-and-Cut 39
5.1. Branch-and-Bound 39
5.1.1. Selección de variable de *branching* 39
5.1.2. Estrategias de recorrido del árbol 40
5.1.3. Heurística Inicial 40

5.1.4. Heurística Primal 41
5.2. Planos de Corte 41
5.2.1. Algoritmos de separación 41

6.. Experimentación Computacional 45
6.1. Implementación del algoritmo y entorno de experimentación 45
6.2. Evaluación de heurísticas 45
6.3. Evaluación de cortes 48
6.3.1. Sin cortes 48
6.3.2. Desigualdades de tipo 2 49
6.3.3. Desigualdades de tipo 4 51
6.3.4. Desigualdades de tipo 5 52
6.3.5. Desigualdades de tipo 6 53
6.3.6. Desigualdades de tipo 7 53
6.3.7. Desigualdades de tipo 8 55
6.3.8. Todas las desigualdades válidas 56
6.4. Evaluación de variables de branching 58
6.5. Reevaluación de heurísticas 58
6.6. Experimentación contra CPLEX 60

7.. Conclusiones y Trabajo Futuro 67

Bibliografía 69

1

Introducción

El objetivo de esta tesis es abordar, desde un enfoque de programación lineal entera, una variante del problema de ordenamiento lineal (POL). El POL es un problema clásico de optimización combinatoria estudiado desde los años 60 [4] que consiste en encontrar un orden de elementos minimizando un costo asociado a que un elemento esté ubicado antes que otro en el orden.

El POL se encuentra en la literatura bajo distintos nombres - máximo subdigrafo acíclico, máximo conjunto de arcos consistentes, entre otros - y surge de diversas disciplinas en donde el problema es de interés, desde antropología [9] y *scheduling* hasta economía y psicología [15].

Se suele plantear como un problema sobre un grafo en el cual se busca un orden total de los vértices de manera tal que por cada par i y j, si el vértice i está posicionado antes que el j en el orden entonces existe un costo asociado c_{ij}. El objetivo es encontrar un orden que minimice la sumatoria de todos los costos.

El problema fue clasificado como NP-difícil por Garey y Johnson [8], por lo tanto no se conocen algoritmos que puedan determinar la solución eficientemente, y no es posible en la práctica resolver de forma exacta el problema para instancias grandes. Existen muchos trabajos en la última década que proponen resolver el problema utilizando algoritmos *Branch-and-Bound* [3] y planos de corte [19].

Al igual que el POL la variación a estudiar se puede formular sobre un grafo. Dado un orden total de vértices se define la distancia como el valor absoluto de la diferencia entre las posiciones en las que se encuentran dichos vértices. También se define la penalidad entre un par de vértices en un orden como el producto entre la distancia en la que se encuentran en ese orden y un costo que está asociado a dicho par de vértices. El problema de ordenamiento lineal con penalidades ($POLP$) consiste en encontrar el orden que minimice la suma de las penalidades entre todo par de vértices. Se encuentra definido en [5] como un caso particular de la clase de problemas denominada *etiquetado de grafos*.

Escrito formalmente, entre dos vértices i y j existe un costo asociado c_{ij} y se nota d_{ij} la distancia entre ellos, lo que se intenta encontrar es un orden de los vértices que minimice:

$$\sum_{1 \leq i < j \leq N} c_{ij} d_{ij}$$

Por ejemplo en una instancia de tamaño 4:

El costo de un orden es $c_{12}d_{12} + c_{13}d_{13} + c_{14}d_{14} + c_{23}d_{23} + c_{24}d_{24} + c_{34}d_{34}$
El costo de $\langle 1, 2, 3, 4 \rangle$ es $4 \times 1 + 8 \times 2 + 4 \times 3 + 3 \times 1 + 2 \times 2 + 1 \times 1 = 40$
El costo de $\langle 4, 2, 1, 3 \rangle$ es $4 \times 1 + 8 \times 1 + 4 \times 2 + 3 \times 2 + 2 \times 1 + 1 \times 3 = 31$
Se puede ver que el costo de $\langle 4, 2, 1, 3 \rangle$ es mínimo entre todos los órdenes y por lo tanto es solución al problema.

$$c = \begin{pmatrix} 0 & 4 & 8 & 4 \\ 4 & 0 & 3 & 2 \\ 8 & 3 & 0 & 1 \\ 4 & 2 & 1 & 0 \end{pmatrix}$$

En este trabajo se concentrará en el caso en que los costos son todos positivos y simétricos ($c_{ij} = c_{ji}$).

Una aplicación del *POLP* se puede encontrar en [1]. En ese problema es necesario ubicar nodos eléctricos en una línea, algunos de estos nodos eléctricos están unidos por cables y el objetivo es minimizar la longitud necesaria de cableado . En este caso los costos entre pares de vértices valen 0 ó 1 dependiendo si debe existir un cable que los una. Esta versión particular del problema también se conoce como *linear arrangement problem*.

Otro escenario consiste en ubicar depósitos equidistantes en fila (como en un puerto), entre los cuales existe cierto flujo de mercadería. El costo de ubicar dos depósitos en determinadas posiciones dependerá del flujo que hay entre ellos y la distancia a la cual se ubican. Una generalización de este caso es muy estudiada y se llama *single row facility layout problem*.

El *POLP* también es NP-difícil, por lo que hasta el momento no es posible resolverlo eficientemente. Para este tipo de problemas difíciles existen dos enfoques para ser abordados computacionalmente. Uno es tratar de resolverlo mediante heurísticas, es decir, encontrar la mejor solución posible buscando en un subconjunto acotado de las soluciones por cierto margen de tiempo. Por otro lado es posible intentar resolverlo en forma exacta, asegurando encontrar el óptimo del problema. Para esto, se puede utilizar algún método de enumeración de soluciones. En esto último consisten los algoritmos *Branch-and-Cut*, en recorrer el espacio de soluciones de manera inteligente podando ciertas soluciones para las cuales se tiene la certeza que no son óptimas.

El *POLP* no fue encontrado en la literatura bajo un enfoque de programción lineal entera. Se encontraron generalizaciones y casos particulares que sirvieron para motivar distintas formulaciones para el problema. Basados en los resultados de Grötschel [12] obtenidos en su trabajo abordando el problema de ordenamiento lineal mediante un algoritmo *Branch-and-Cut*, el objetivo de este trabajo será desarrollar e implementar un algoritmo *Branch-and-Cut* para resolver el *POLP*. Para esto será necesario proponer modelos de programación lineal entera para el problema, explorar distintas formas de recorrer el espacio de soluciones asociados a estos, y realizar un estudio poliedral para poder podar, de la mejor forma posible, este espacio de soluciones.

El trabajo se presenta en 6 capítulos:

1. **Preliminares** En este capítulo se introducen algunos conceptos utilizados. Contiene

una introducción a la programación lineal entera y las nociones sobre grafos usadas para plantear formalmente los problemas relacionados.

2. **Modelo** Para tratar el problema bajo un enfoque de programación lineal entera es necesario tener un modelo que lo describa. Dado que no se encontraron modelos de programación lineal entera, se evaluaron distintos modelos de problemas relacionados y se trataron de adaptar algunos de ellos. En este capítulo se presentan los distintos modelos encontrados y se proponen las adaptaciones que permiten describir el problema estudiado.

3. **Estudio poliedral** En este capítulo se trabaja sobre el modelo elegido y el estudio de características de su conjunto de soluciones factibles. Se busca encontrar igualdades y desigualdades válidas para dichas soluciones. Esta información se utilizará para desarrollar el algoritmo *Branch-and-Cut* que resuelve el problema.

4. **Algoritmo Branch-and-Cut** En este capítulo se presenta el algoritmo para resolver el problema. Se describe de qué forma se analiza el conjunto de soluciones factibles y la estrategia utlizada para introducir las desigualdades del capítulo anterior.

5. **Experimentos computacionales** El objetivo de este capítulo es mostrar con experimentos computacionales cómo cada una de las características descriptas en el capítulo anterior ayudan a mejorar la performance del algoritmo desarrollado.

6. **Conclusiones y Trabajo futuro** Por último se resumen las conclusiones obtenidas de los capítulos anteriores y se describen algunas líneas futuras de trabajo.

2

Preliminares

2.1. Introducción a la Programación Lineal Entera

2.1.1. Definición

La Programación Lineal es una herramienta para modelar problemas de optimización donde se debe minimizar una función lineal, denominada función objetivo, sujeta a un conjunto de restricciones, también lineales. Todo problema de Programación Lineal es resoluble de manera eficiente [14], pero existen muchos problemas que no se pueden modelar con esta técnica a menos que se restrinja el dominio de las variables.

La Programación Lineal Entera está íntimamente relacionada con la Programación Lineal, pero tiene la diferencia de que todas las variables están restringidas a tomar valores enteros.

La Programación Lineal Entera Mixta (PEM) es una generalización de las dos anteriores, en donde algunas de las variables están restringidas a valores enteros.

La PEM no sólo puede modelar el mismo tipo de problemas que Programación Lineal, por definición, sino que también puede hacerlo con problemas donde las variables son enteras. Esto permite describir problemas donde no se pueden tener valores reales, por ejemplo en los problemas de producción de unidades indivisibles. Además permite modelos mucho más potentes, ya que la presencia de variables enteras o binarias permite la modelización de restricciones lógicas como conjunciones, disyunciones y consecuencias. El caso más emblemático es el problema SAT, que puede ser modelado mediante PEM y no mediante Programación Lineal.

Con PEM se pueden modelar problemas que son NP-Difíciles, no existe ningún algoritmo polinomial que resuelva cualquier problema de PEM, a menos que P = NP. Una demostración de que la PEM es NP-Difícil se obtiene justamente mostrando que SAT se puede reducir a un problema de PEM [6].

La PEM puede ser utilizada para modelar variados problemas de optimización, entre los que se encuentran problemas de ruteo de vehículos, de telecomunicaciones, de planificación de producción, de asignación de tareas y turnos, entre otros.

En las próximas secciones se detallan algunos conceptos básicos de la PEM y se presentan los algoritmos generales de resolución conocidos. Para una ampliación de los temas

aquí introducidos, ver [23].

2.1.2. Formulaciones

Un problema de PEM puede ser formulado de la siguiente manera donde I es el conjunto de variables enteras, C es el conjunto de variables continuas y m es la cantidad de restricciones:

$$
\begin{aligned}
\textit{Minimizar} \quad & \sum_{j \in I} c_j x_j + \sum_{j \in C} c_j x_j \\
\textit{sujeto a} \quad & \\
& \sum_{j \in I} a_{ij} x_j + \sum_{j \in C} a_{ij} x_j \leq b_i \quad i = 1, \ldots, m \\
& x_j \in \mathbb{Z}_+ \qquad \forall j \in I \\
& x_j \in \mathbb{R}_+ \qquad \forall j \in C
\end{aligned}
$$

Cada formulación PEM tiene asociado un poliedro $P = \{x \in \mathbb{R}^n_+ : Ax \leq b\}$ con $A \in \mathbb{R}^{m \times n}$ y $b \in \mathbb{R}^m$ y el conjunto de soluciones factibles $S = P \cap \{x \in \mathbb{R}^n : x_j \in \mathbb{Z}\ \forall j \in I\}$. A P se lo denomina *relajación lineal* de S.

Cada desigualdad presentada en las restricciones divide en dos el espacio donde viven las soluciones.

A continuación se presenta el poliedro asociado a una formulación con sólo dos variables, y ambas restringidas a ser enteras.

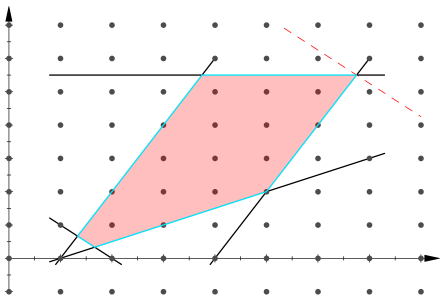

Fig. 2.1: Conjunto factible y poliedro asociado

El poliedro se encuentra definido como la intersección de las regiones definidas por las desigualdades presentes en la formulación. Si bien este poliedro contiene infinitos puntos, el conjunto de soluciones factibles S son los puntos enteros en x_j con $j \in I$.

La recta que se observa punteada corresponde a la función objetivo. Al ser ésta una función lineal, en el espacio queda representada por un hiperplano, en este caso una recta, que sólo tiene definida su pendiente ya que no tiene término independiente como las desigualdades de las restricciones. Es decir, la función objetivo es una curva de nivel con pendiente definida que toma diferentes valores según donde esté ubicada. Lo buscado es

que la función objetivo pase por el punto del conjunto factible de soluciones que menor valor le haga tomar. En el caso de la figura se puede ver la recta de la función objetivo sobre la solución que se encuentra en el vértice superior derecho.

En los problemas de Programación Lineal las soluciones factibles son todos los puntos del poliedro ya que no está la restricción de integralidad. Esto hace que en un poliedro acotado, como el mostrado en la figura, se pueda demostrar que una de las mejores soluciones está en uno de los puntos extremos o vértices del mismo. Por lo que en un problema de Programación Lineal, se pueden recorrer los vértices hasta encontrar el que tenga menor valor objetivo, siempre haciendo referencia a una minimización. Puede existir una cantidad de vértices exponencial con respecto a la cantidad de variables y restricciones por lo que en teoría puede resultar un método ineficente para encontrar la solución. Existen algoritmos basados en este método que presentan muy buenos resultados en la práctica.

Si llamamos $conv(S)$ a la cápsula convexa de S (menor poliedro que contiene a S), entonces PEM es equivalente a resolver mín$\{cx \mid x \in conv(S)\}$. Si $P = conv(S)$, el problema PEM puede ser resuelto por cualquier algoritmo de Programación Lineal. Esto sucede porque un algoritmo para resolver Programación Lineal puede encontrar el mínimo en todo el poliedro P y, por lo dicho anteriormente, este mínimo será alguno de los vértices del poliedro. Como en la cápsula convexa los vértices coinciden con puntos enteros, entonces cuando se minimice el problema sobre todo P, resultará que el mínimo es un punto entero, por lo que este punto es mejor o igual, en términos de la función objetivo, que cualquier otro punto de P y en particular que cualquier otro punto de S, ya que $S \subseteq P$.

Si se conociese la descripción de $conv(S)$ mediante una cantidad polinomial (en la cantidad de variables) de desigualdades lineales, se podría resolver el problema como uno de Programación Lineal, lo cual es computacionalmente fácil. Es más, aún en el caso en que esta caracterización no fuese polinomial, bajo ciertas circunstancias el problema podría ser resuelto en tiempo polinomial. Desafortunadamente, para la mayoría de los problemas no se ha podido obtener la descripción completa de la cápsula convexa y, en general, el número de restricciones lineales que la caracterizan es exponencial.

Existen muchas maneras de formular un problema dado. A veces con las mismas variables existen otras restricciones que también modelan el problema, pero que devienen lógicamente en poliedros asociados a la relajación lineal diferentes. Otras veces un problema se puede formular con dos conjuntos de variables diferentes, lo que hace que probablemente los poliedros asociados ni siquiera pertenezcan a la misma dimensión.

En la siguiente figura se pueden ver los poliedros asociados a dos formulaciones diferentes, siempre manteniéndose en dos dimensiones para poder visualizarlo correctamente.

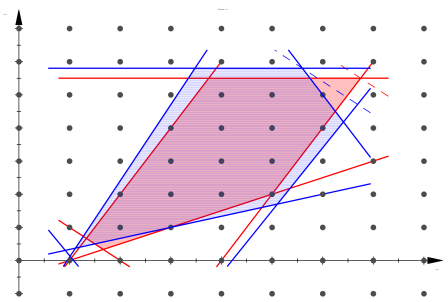

Fig. 2.2: Dos formulaciones y sus poliedros asociados

Por un lado se ve el poliedro rojo opaco, y por el otro el poliedro azul. Se puede observar que el conjunto de puntos enteros S es el mismo, pero el poliedro asociado P es diferente.

Se dice que una formulación $F1$ es *más ajustada* que otra, $F2$, cuando el poliedro asociado a $F1$ está incluido en el poliedro asociado a $F2$. En este caso $F1$ logra capturar todos los puntos enteros posibles y contiene menos puntos racionales que $F2$.

Esta noción conforma una relación de orden que no es total. En la figura anterior se ve que hay sectores coloreados de rojo, pero no de azul, y viceversa. Esto indica que ninguna de las dos formulaciones es más ajustada que la otra.

Bajo esta definición, la cápsula convexa de un conjunto de soluciones factibles representa la formulación más ajustada que se puede conseguir. En la siguiente figura se muestra la primera formulación visualizada, junto con la cápsula convexa de los puntos de S. En rojo opaco se encuentra la primera formulación, y con la recta gruesa azul se remarca la cápsula convexa.

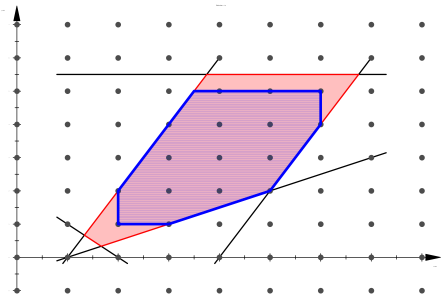

Fig. 2.3: Formulación y Cápsula convexa

En general, para un problema de Programación Lineal Entera se busca diseñar formulaciones lo más ajustadas posibles. Esta tarea en general resulta difícil, dado que la ayuda gráfica sólo sirve para los problemas en dos o tres dimensiones, y existen problemas que surgen en la práctica que suelen tener cientos de miles de variables.

2.1.3. Algoritmos para Problemas de Programación Lineal Entera

El procedimiento más simple para resolver un problema de Programación Lineal Entera pura es enumerar todas las posibilidades. Sin embargo, debido a la explosión combinatoria esta técnica sólo resulta aplicable a instancias sumamente pequeñas.

Los algoritmos más utilizados se encuadran dentro de algunos de estos esquemas básicos:

- Enumeración inteligente: algoritmos *Branch-and-Bound*.
- Caracterización de $conv(S)$ o ajuste de la relajación lineal: algoritmos de planos de corte.
- Una combinación de las dos técnicas anteriores: algoritmos *Branch-and-Cut*.

A continuación se describen los puntos más sobresalientes de cada uno.

Algoritmos *Branch-and-Bound*

Ya se mencionó que la enumeración de las soluciones factibles en busca de la solución óptima no es un procedimiento aconsejable para usar en la práctica. Para mejorar esta técnica básica muchas veces es posible eliminar algunas posibilidades mediante argumentos de dominancia o factibilidad. Es decir, argumentos que permiten afirmar que el óptimo no pertenece a un determinado subconjunto de soluciones sin la necesidad de enumerarlo.

Dentro de esta línea, en los años 60 fueron propuestos los algoritmos *Branch-and-Bound* [16], donde el *branching* se refiere a la parte enumerativa y el *bounding* al proceso de poda de posibles soluciones.

Estos algoritmos están asociados al concepto *divide y conquista*: si resulta difícil buscar el óptimo en un conjunto S, entonces es mejor buscar en partes de S y luego quedarse con la mejor solución.

Este esquema puede ser representado mediante un árbol cuya raíz corresponde al problema original y sus ramas resultan de la división en partes del espacio de búsqueda. A cada nodo del árbol le corresponde un subproblema que consiste en buscar el óptimo en una parte del espacio de soluciones. Los argumentos de dominancia y factiblidad son los que permitirán descartar ramas del árbol en el proceso de búsqueda.

Una forma de llevar a cabo la poda, *bounding*, es calcular en los nodos del árbol cotas inferiores del óptimo del problema restringido a esa parte del espacio de soluciones. Si la cota es peor que la mejor solución obtenida hasta el momento, no es necesario explorar dicha parte. El cálculo de estas cotas debe lograr un equilibrio entre calidad y esfuerzo en obtenerla. Una cota débil hará que se explore innecesariamente ramas del árbol, pero un procedimiento que brinde buenas cotas a un costo alto puede no justificarse.

Para obtener cotas inferiores, una posibilidad es relajar el problema de forma de obtener una relajación *fácil* de resolver. La idea es reemplazar un PEM difícil por un problema de optimización más simple cuyo valor óptimo sea menor o igual al óptimo del problema original. Obviamente, es deseable obtener relajaciones *ajustadas*, es decir, que la diferencia relativa (*gap*) entre el valor óptimo de la relajación y el valor óptimo del PEM sea chica. Hay dos posibilidades obvias para que el problema relajado tenga esta característica. Se puede agrandar el conjunto de soluciones factibles sobre el cual se optimiza o reemplazar la función objetivo por otra que tenga igual o menor óptimo. Dentro de la primera posibilidad se encuentra la relajación lineal y en la segunda se enmarca la relajación lagrangeana. Las relajaciones no sólo son útiles para obtener cotas inferiores, algunas veces permiten probar optimalidad.

La relajación lineal consiste en borrar del PEM la imposición de ser entera sobre las variables que la tengan, es lo que en los gráficos mostrados anteriormente se enunciaba como el poliedro P asociado a S. Es la relajación más natural y una de las más utilizadas. La relajación lagrangeana consiste en remover un subconjunto de las restricciones que no incluya las restricciones de no negatividad. La violación de las restricciones relajadas es penalizada incluyendo estas restricciones, con un multiplicador no negativo, en la función objetivo. Los multiplicadores son iterativamente actualizados para maximizar la cota in-

ferior obtenida del problema relajado.

El proceso de *branching* consiste en dividir la región factible en dos o más regiones factibles más pequeñas. Cada nueva región da origen a un nuevo subproblema o nodo hijo, originado por la adición de una nueva restricción al problema del nodo padre. Un requerimiento esencial es que cada solución entera factible del nodo padre pertenezca a al menos uno de los hijos. Estos nuevos subproblemas son agregados a la lista de nodos activos, es decir, aún no explorados. La regla de *branching* más simple consiste en considerar alguna variable entera que tiene valor fraccionario d en la solución actual. Se parte al problema en dos hijos, imponiendo en uno de ellos como cota superior de este variable el valor $\lfloor d \rfloor$ y en el otro como cota inferior $\lceil d \rceil$. Este procedimiento es aplicado recursivamente a cada nodo del árbol.

En la siguiente figura se puede ver cómo la formulación original es partida en dos subproblemas, tomando la variable x y separando cuando vale menor o igual a 3, y cuando vale mayor o igual a 4. Se puede ver que esta separación propuesta cumple con el requisito de que toda solución entera factible se encuentra en alguno de los dos subproblemas nuevos.

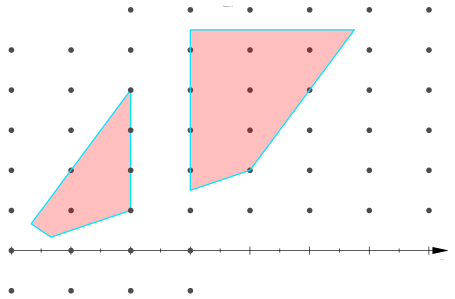

Fig. 2.4: Regla de Branching

Es importante notar que la idea de esta simple regla de *branching* tiene por objetivo también que el punto fraccionario óptimo actual no se encuentre en ninguno de los dos subproblemas. Tomando la imagen anterior, si el óptimo de la relajación lineal hubiese estado en el vértice superior izquierdo del poliedro donde la variable x toma un valor entre 3 y 4, el branching propuesto lograría dejar de tener en cuenta este punto.

La próxima decisión que se debe tomar es la selección del siguiente nodo a explorar. En la práctica hay varias alternativas que pueden ser utilizadas. Como sólo es posible podar significativamente el árbol de enumeración si se cuenta con buenas cotas superiores, entonces se debería descender lo más pronto posible en el árbol para encontrar rápidamente soluciones factibles. Esto sugiere el uso de una estrategia de *búsqueda en profundidad*. Otra estrategia sugiere elegir el nodo activo con mejor cota (más chica). De esta manera, nunca se dividiría un nodo con cota inferior mayor que el valor óptimo del problema. Esta estrategia es llamada *mejor cota primero*.

No hay una combinación de estos factores que resulte mejor para todos los problemas. Es necesario utilizar criterios basados en una combinación de teoría, sentido común y experimentación.

El esquema básico del algoritmo es el siguiente. Llámese $\mathcal{F}$ el problema entero mixto que se quiere resolver, $\mathcal{N}$ al conjunto de subproblemas o nodos del árbol de enumeración activos. Para cada nodo k, $PL(k)$ representa la relajación lineal del PEM asociado a este nodo y Z^k el valor óptimo de $PL(k)$. En Z^* se almacena el valor de la mejor solución obtenida.

---

1. **Inicialización:**
   $\mathcal{N} = \{\mathcal{F}\} \quad Z^* = \infty$

2. **Elección de próximo nodo:**
   Si $\mathcal{N} = \{\}$ el algoritmo termina. Si $Z^* \neq \infty$ entonces es óptimo. Si no, $\mathcal{F}$ es no factible
   Si $\mathcal{N} \neq \{\}$, elegir y borrar un nodo k de $\mathcal{N}$

3. **Evaluación:**
   Resolver $PL(k)$.

   a) Si no es factible, ir a **Elección**.

   b) *Bound*: si $Z^k > Z^*$, ir a **Elección**.

   c) Si la solución óptima cumple las condiciones de integralidad, actualizar $Z^* = \text{mín}\{Z^*, Z^k\}$ e ir a **Elección**.

4. **División:** (*Branch*) Particionar la región factible de $PL(k)$ en dos o más regiones, agregando un nuevo nodo a $\mathcal{N}$ por cada nueva región. Ir a **Elección**.

Algoritmos de planos de corte

Los algoritmos de planos de corte fueron originalmente propuestos por Gomory [10] en los 60's como un método general para resolver problemas de programación lineal entera.

Un algoritmo básico de planos de corte en un primer paso relaja las condiciones de integralidad sobre las variables y resuelve el programa lineal resultante. Si el programa lineal es infactible, el programa entero también lo es. Si la solución óptima del programa lineal cumple las condiciones de integralidad, se ha encontrado un óptimo del problema. En caso contrario, se busca identificar desigualdades lineales (*problema de separación*) que estén violadas por la solución fraccionaria del programa lineal y sean válidas para los puntos enteros factibles. Es decir, desigualdades que *separen* el óptimo fraccionario de $conv(S)$.

Siguiendo con el ejemplo gráfico que se presentó anteriormente, se puede visualizar la idea de este método. En la siguiente figura se muestra la formulación original con su óptimo en la relajación lineal, y una desigualdad válida que fortalece al modelo y deja afuera al anterior óptimo ya que era un punto fraccionario que no se quiere tener en cuenta.

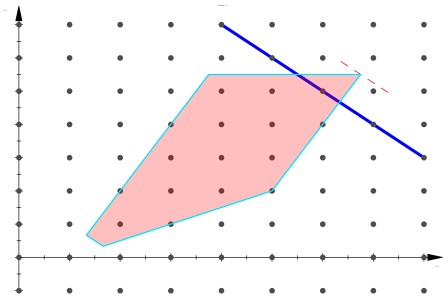

Fig. 2.5: Plano de corte

Se observa cómo la recta gruesa azul corta la solución fraccionaria óptima que existía hasta el momento, pero no corta ningún punto entero factible. De esta manera, si se vuelve a resolver la relajación lineal, pero ahora con el modelo fortalecido, dará un óptimo diferente.

El algoritmo continúa hasta que:

- una solución óptima entera es encontrada, en cuyo caso el problema es resuelto con éxito
- el programa lineal es infactible, lo que significa que el problema entero es infactible
- no se pudo identificar alguna desigualdad lineal que separe al óptimo de la relajación actual, ya sea porque no se conoce la descripción completa de la cápsula convexa o porque los algoritmos de separación no son exactos.

El éxito del algoritmo depende en gran medida de la posibilidad y la eficiencia de encontrar desigualdades violadas (*planos de corte*) que puedan ser agregadas a la formulación para separar las soluciones fraccionarias.

Los planos de corte pueden ser generados bajo dos enfoques:

- *Con herramientas generales aplicables a cualquier problema de programación lineal entera*

  El algoritmo original propuesto por Gomory [10] utiliza como planos de corte desigualdades derivadas del *tableau* óptimo de la relajación lineal, llamados *cortes de Gomory*. Aunque fue demostrado que este algoritmo, bajo ciertas condiciones, termina en un número finito de pasos, en la práctica su convergencia parece ser muy lenta. Por otro lado, la implementación computacional es numéricamente inestable, aunque en la actualidad han sido fortalecidos lográndose buenas implementaciones.

  Posteriormente, se han desarrollado algoritmos que utilizan una variedad de cortes aplicables a cualquier PEM, como por ejemplo los cortes *disyuntivos, clique, cover*, etc. Si bien estos algoritmos tienen propiedades teóricas de mucho interés, sólo son exitosos para cierta clase de problemas, mientras que para otros no lo son. Cualquiera de las técnicas mencionadas tiene la ventaja de poder ser utilizadas para

cualquier problema de Programación Lineal Entera, independientemente de su estructura. Si bien esto es una propiedad deseable en un algoritmo, no siempre brinda la herramienta más adecuada para casos particulares. Un estudio más específico del problema ayuda a obtener mejores procedimientos. Este es el sentido del próximo enfoque.

- *Explotando la estructura particular del problema.*

  En los 70's, resurgió el interés por los algoritmos de planos de corte debido al desarrollo de la teoría poliedral. Mediante el estudio de combinatoria poliedral, la intención es reemplazar el conjunto de restricciones de un programa de programación lineal entera mixta por la descripción de la cápsula convexa del conjunto de soluciones factibles. Las desigualdades lineales necesarias para describir a $conv(S)$ se denominan *facetas*. Si se conoce de forma completa esta descripción, el problema entero podría ser resuelto con los métodos utilizados para resolver un problema de programación lineal.

  Desafortunadamente, no es fácil tener esta descripción y no se conocen problemas pertenecientes a la clase NP-Difícil que tengan una cantidad polinomial de facetas. Es posible utilizar cualquier desigualdad válida para el conjunto de soluciones factibles como planos de corte, pero, en general, la eficiencia del algoritmo depende de la *fortaleza* de estas desigualdades, siendo las facetas las que suelen proveer los *mejores* cortes posibles.

  Con fines algorítmicos, el estudio poliedral debe estar acompañado de algoritmos de separación eficientes. En este sentido, hay un resultado muy importante de Grötschel, Lovász y Schrijver [14] que relaciona la complejidad del problema de separación con la complejidad del problema de optimización. Se establece que el problema de optimización $max\{cx : x \in conv(S)\}$ puede resolverse polinomialmente si y sólo si el problema de separación ($x \in conv(S)$ *ó encontrar una desigualdad válida violada*) es polinomial. Es decir que si el problema que se está tratando no es polinomial, existe al menos alguna familia de facetas que no puede separarse en tiempo polinomial. Esto de alguna manera implica el grado de dificultad de encontrar la descripción de todas las facetas de la cápsula convexa y del desarrollo de algoritmos de separación.

En general, para desarrollar un algoritmo de planos de corte, primero se busca una descripción parcial de la cápsula convexa del conjunto de las soluciones factibles enteras o desigualdades válidas *fuertes* para este conjunto. Luego es necesario el diseño de rutinas de separación para las familias de desigualdades encontradas. Estas rutinas toman como entrada una solución y retornan restricciones de estas familias violadas por este punto, si es que existe alguna. En algunos casos el problema de separación puede ser NP-difícil o tener complejidad alta, lo que lleva en la práctica a utilizar algoritmos heurísticos, o sea, que es posible que no sean capaces de encontrar una desigualdad violada aunque exista. La estrategia que se utilice para decidir la búsqueda en la diferentes familias es clave para la performance del algoritmo.

El esquema básico de un algoritmo de planos de corte es el siguiente. Llámese $\mathcal{F}$ al problema entero mixto que se quiere resolver, $PL(P)$ a la relajación lineal del problema P y x^P la solución óptima de esta relajación.

---

1. **Inicialización:**
   $P = \mathcal{F}$
2. **Evaluación:**
   Resolver $PL(P)$
   a) Si es no factible, entonces $\mathcal{F}$ es no factible y el algoritmo termina.
   b) Si x^P cumple las condiciones de integralidad, x^P es la solución óptima de $\mathcal{F}$ y el algoritmo termina.
   c) **Separación:** Caso contrario, resolver el problema de separación para x^P.
   - Si se encuentran cortes, agregarlos a P e ir a **Evaluación**.
   - Caso contrario, retornar el funcional evaluado en x^P como una cota inferior de $\mathcal{F}$.

El algoritmo de planos de corte puede no resolver el problema de forma óptima, ya sea por no encontrar desigualdades válidas violadas o porque el tiempo consumido excede el tiempo disponible. Sin embargo, puede ser utilizado para generar buenas cotas inferiores del valor óptimo del problema.

Algoritmos *Branch-and-Cut*

En muchas instancias, los dos algoritmos descriptos arriba fallan en la resolución del problema. A comienzos de los 80's se comenzó a aplicar una metodología mixta que conjuga las dos ideas dando origen a los llamados algoritmos *Branch-and-Cut*. De esta manera se lograron resolver exitosamente instancias de tamaño considerable de una gran cantidad de problemas de programación lineal entera, como por ejemplo el *Problema de Viajante de Comercio* [20], el *Problema de Ordenamiento Lineal* [11], el *Problema de Corte Máximo* [13], el *Problema de Coloreo de Grafos* [17] etc.

Uno de los factores que influye en el fracaso de los algoritmos *Branch-and-Bound* es la baja calidad de las cotas obtenidas mediante las relajaciones lineales. Esto significa que resulta crucial poder ajustar las formulaciones, por ejemplo con planos de corte.

Un algoritmo *Branch-and-Cut* es un *Branch-and-Bound* en el cual se generan planos de corte a través del árbol de enumeración. El objetivo de esto es reducir significativamente el número de nodos del árbol mejorando la formulación de los subproblemas. En un *Branch-and-Cut*, la enumeración y los planos de corte se benefician mutuamente. Generalmente, la cota producida en cada nodo del árbol de enumeración es mejor que en un *Branch-and-Bound*, debido a las nuevas desigualdades agregadas a la formulación del correspondiente subproblema.

Estos algoritmos no sólo son capaces de producir la solución óptima, también pueden brindar soluciones aproximadas al óptimo con certificado de calidad en un tiempo de

cómputo moderado. Es decir, aún sin conocer el valor real del óptimo, se puede acotar que tan lejos se está de éste.

En la implementación de un algoritmo *Branch-and-Cut* hay que tener en cuenta las estrategias propias de un algoritmo *Branch-and-Bound* sumadas a las de un algoritmo de planos de corte. Además, se agregan nuevas decisiones como ¿cuándo buscar planos de cortes?, ¿cuántos cortes agregar?, etc.

El esquema de un algoritmo *Branch-and-Cut* es el siguiente.

---

1. **Inicialización:**
   $\mathcal{N} = \{\mathcal{F}\} \quad Z^* = \infty$

2. **Elección de próximo nodo:**
   Si $\mathcal{N} = \{\}$ Z^* es óptimo y el algoritmo termina
   Si no, elegir y borrar un nodo k de $\mathcal{N}$

3. **Evaluación:**
   Resolver $PL(k)$.

   a) Si es no factible, ir a **Elección**.

   b) Si $Z^k > Z^*$, ir a **Elección**.

   c) Si la solución óptima cumple las condiciones de integralidad, poner $Z^* = \text{mín}\{Z^*, Z^k\}$ e ir a **Elección**.

4. **División vs Separación:**
   Decidir si se buscarán planos de corte:

   - **SI:** Ir a **Separación**
   - **NO:** Ir a **División**

5. **División:** Particionar la región factible de PL(k) en dos o más regiones, agregando un nuevo nodo a $\mathcal{N}$ por cada nueva región. Ir a **Elección**.

6. **Separación:** Resolver el problema de separación para la solución fraccionaria de $PL(k)$.

   - Si son encontrados cortes, agregarlos a la formulación e ir a **Evaluación**.
   - Si no se encuentran, ir a **División**.

2.2. Definiciones sobre grafos

Los problemas explicados en los próximos capítulos estarán planteados como problemas sobre grafos, por lo que se introducen algunos conceptos necesarios.

Un *grafo* $G = (V, A)$ consiste en un conjunto de vértices V y un conjunto de aristas A que relacionan dos vértices del conjunto V.

Un *camino* es una secuencia de vértices tal que existe una arista que relaciona a todo par de vértices consecutivos. Si el camino empieza y termina en el mismo vértice, se lo llama un *ciclo*. Si todos los vértices del *ciclo* son distintos entonces se lo denomina un *ciclo simple* y si además están todos los vértices del grafo se lo llama un *ciclo hamiltoniano*.

Si la relación entre vértices no es simétrica, se dice que las aristas son dirigidas y se dice que G es un *digrafo*. En este caso (i, j) representa un *arco* que sale de i y llega a j.

Un *camino dirigido* es una secuencia de vértices tal que para todo par de vértices consecutivos existe un arco que sale del primero y llega al próximo. Al igual que el caso no dirigido, un *camino dirigido* que empieza y termina en el mismo vértice, se lo llama un *ciclo dirigido*.

Se nota $D_n = (V_n, A_n)$ al digrafo completo con n vértices, es decir, $V_n = 1, 2, \ldots, n$ y para cada par de vértices distintos i y j existe un arco.

Un *torneo* T en A_n consiste en un subconjunto de los arcos de D_n tal que para cada par de vértices i y j aparece el arco (i, j) o (j, i), pero no ambos. Un *torneo acíclico* es un torneo sin ciclos dirigidos. Alternativamente se define un *ordenamiento lineal* de los vértices $1, 2, \ldots, n$ como un ranking de los vértices dados como una secuencia lineal, es decir, una permutación de los vértices.

Un *etiquetado* consiste en una asignación de etiquetas, usualmente representadas con números enteros, a los vértices de un grafo. Un *etiquetado lineal* de un grafo consiste en un etiquetado tal que las etiquetas son todos los números entre 1 y la cantidad de vértices, y todas son asignadas a algún vértice.

3

Modelo

Existen varios modelos en la literatura que tienen relación con el *POLP*. Los problemas de ordenenamiento lineal, etiquetado lineal mínimo y viajante de comercio sirvieron para motivar distintas formulaciones.

En la primera parte de este capítulo se presentan estos problemas, contar en qué consisten y de dónde surgen, junto con algunos de los modelos que los describen. En la segunda parte se plantean las modificaciones a los modelos expuestos previamente para adaptarlos al *POLP*. Se realizaron pruebas computacionales entre ellos comparando el rendimiento para seleccionar el modelo que será utilizado para el algoritmo *Branch-and-Cut*.

3.1. Modelos previos

El *POLP* no fue encontrado en la literatura tratado como un problema de programación lineal entera. Sin embargo se pueden encontrar problemas que se asemejan que fueron muy estudiados bajo este enfoque y podrían adaptarse para modelar *POLP*.

3.1.1. Ordenamiento lineal

Uno de los problemas que tienen como objetivo encontrar un orden total de vértices es el problema de ordenamiento lineal (*POL*). Este fue tratado por Grötschel [12] en 1985 y se encuentra clasificado como NP-difícil por Garey y Johnson [8] en 1979.

Dado un grafo completo dirigido $D_n = (V_n, A_n)$ con costo c_{ij} para cada $(i, j) \in A_n$, el objetivo del *POL* es computar un *torneo acíclico* T en A_n tal que $\sum_{(i,j)\in T} c_{ij}$ sea mínimo.

Para modelarlo con PEM se usa un modelo con una cantidad cuadrática de variables con respecto a la cantidad de vértices. Por cada par de vértices i y j existe una variable binaria x_{ij} que vale 1 si i está antes que j (corresponde a estar usando un arco que sale de i y llega a j). Para exigir que las soluciones factibles del modelo definan un torneo se debe restringir que para cada par de vértices tiene que valer 1 alguna de las variables x_{ij} o x_{ji} pero no las dos. Por último se debe verificar que no existan ciclos para tener bien definido el orden (es decir, que el torneo sea acíclico). Se puede ver que si existe algún ciclo entonces existe un ciclo de tres vértices por tratarse de un torneo, por lo que alcanza

con restringir que no existen tres vértices formando un ciclo. De esta manera los puntos factibles representan un torneo acíclico, que es lo que busca el problema.

$$\begin{aligned} \text{minimizar} \quad & \sum_{1 \leq i,j \leq N} c_{ij} x_{ij} & \\ \text{sujeto a} \quad & x_{ij} + x_{ji} = 1 & 1 \leq i < j \leq N \\ & x_{ij} + x_{jk} + x_{ki} \leq 2 & 1 \leq i < j, k \leq N, k \neq i, j \\ & x_{ij} \in \{0, 1\} & 1 \leq i, j \leq N, i \neq j \end{aligned}$$

Fig. 3.1: Formulación de *POL*

3.1.2. Etiquetado lineal mínimo

Existe un problema que busca minimizar las distancias entre pares de vértices llamado etiquetado lineal mínimo (*ELM*). Fue demostrado NP-difícil por Garey y Johnson [8].

El problema de etiquetado lineal mínimo se define como: Dado un grafo $G = (V, A)$ con n vértices, se busca encontrar un etiquetado de los vértices con los números de 1 a n (es decir un etiquetado lineal) tal que se minimice la suma de las distancias entre los vértices adyacentes en G. La distancia d_{ij} entre el vértice i y el vértice j está definida como la diferencia absoluta entre sus etiquetas.

Este es un caso particular del *POLP* ya que equivale a tener un costo de 1 entre aquellos vértices que tengan un arco en el grafo, y un cero entre aquellos que no. El problema es NP-difícil por lo que *POLP* también lo es.

El modelo propuesto para este problema por Caprara, Letchford y Salazar-González [2] tiene una cantidad cuadrática de variables y $\mathcal{O}(n^4)$ restricciones. Las variables d_{ij} representan la distancia entre los vértices i y j y las variables x_{ip} indican si el vértice i tiene etiqueta p. Por lo tanto las primeras dos restricciones indican que todas las etiquetas son usadas, y que todos los vértices tienen asignada una etiqueta, es decir x es una matriz de permutación. La última restricción impone una cota inferior a las variables de distancia, si el vértice i tiene etiqueta p y el vértice j tiene etiqueta q entonces d_{ij} será mayor o igual a $|q - p|$ (debido a la función objetivo, d_{ij} tendrá el menor valor posible, por lo que se cumplirá la igualdad).

$$
\begin{aligned}
\text{minimizar} \quad & \sum_{(i,j)\in A} d_{ij} & \\
\text{sujeto a} \quad & \sum_{i=1}^{N} x_{ij} = 1 & 1 \le j \le N \\
& \sum_{j=1}^{N} x_{ij} = 1 & 1 \le i \le N \\
& d_{ij} \ge (q-p)(x_{ip} + x_{jq} - 1) & 1 \le i, j, p, q \le N, p < q, i < j \\
& d_{ij} \ge (q-p)(x_{jp} + x_{iq} - 1) & 1 \le i, j, p, q \le N, p < q, i < j \\
& x_{ij} \in \{0,1\}, d_{ij} \ge 0, d_{ij} \in \mathbb{Z} & 1 \le i, j \le N, i \ne j
\end{aligned}
$$

Fig. 3.2: Formulación de *ELM*

3.1.3. Viajante de comercio

El problema del viajante del comercio es un problema clásico de optimización combinatoria. Fue introducido en 1954 por Dantzig, Fulkerson y Johnson [7]. Es muy estudiado en el área y ha sido base de una importante cantidad de avances teóricos y algorítmicos. Consiste en buscar el circuito hamiltoniano de menor costo en un grafo con costos en los arcos.

Existen muchas formulaciones para modelar este problema, entre ellas la de *Miller-Tucker-Zemlin* [18] donde introducen variables de posición que indican el orden en que los vértices son recorridos.

En este modelo las variables x_{ij} representan si está siendo usado el arco que sale de i y llega a j en el circuito hamiltoniano. Si c_{ij} representa el costo asociado al arco, la función objetivo es la suma de los costos de todos los arcos usados. Las primeras dos restricciones indican que todos los vértices tienen un arco entrante y uno saliente. Por otro lado las variables u_i representan la posición en la que se encuentra el vértice i en el circuito. La restricción asociada indica que si se usa el arco entre i y j entonces la diferencia entre u_i y u_j debe ser 1, salvo para el último vértice del recorrido.

$$
\begin{aligned}
\text{minimizar} \quad & \sum_{\substack{1 \le i,j \le N \\ i \ne j}} c_{ij} x_{ij} & \\
\text{sujeto a} \quad & \sum_{\substack{i=1 \\ i \ne j}}^{N} x_{ij} = 1 & 1 \le j \le N \\
& \sum_{\substack{j=1 \\ i \ne j}}^{N} x_{ij} = 1 & 1 \le i \le N \\
& u_i - u_j + (N-1)x_{ij} \le N - 2 & 2 \le i, j \le N \\
& u_1 = 0 & \\
& x_{ij} \in \{0,1\}, 1 \le u_i \le N-1, u_i \in \mathbb{Z} 1 \le i, j \le N &
\end{aligned}
$$

Fig. 3.3: Formulación de *TSP-MTZ*

Otro modelo del problema de viajante de comercio que resulta interesante es la formulación de Sherali-Driscol [22]. Es una reformulación del modelo *MTZ* utilizando una técnica de reformulación-linealización (RLT) [21] consiguiendo así mejores relajaciones. Además de las variables de orden del recorrido de los vértices se agregan al modelo las variables y_{ij} para indicar un orden de los arcos en el tour.

$$
\begin{aligned}
\text{minimizar} \quad & \sum_{1 \le i < j \le N+1} c_{ij} x_{ij} & \\
\text{sujeto a} \quad & \sum_{1 \le i \le N} x_{ij} = 1 & 1 \le j \le N \\
& \sum_{1 \le j \le N} x_{ij} = 1 & 1 \le i \le N \\
& \sum_{j=1}^{N} y_{ij} + N x_{i1} = u_i & 2 \le i \le N \\
& \sum_{i=1}^{N} y_{ij} + 1 = u_j & 2 \le j \le N \\
& x_{ij} \le y_{ij} \le (N-2) x_{ij} & 2 \le i, j \le N \\
& u_j + (N-1) x_{ij} - N(1 - x_{ji}) \le y_{ij} + y_{ji} & 2 \le i, j \le N \\
& y_{ij} + y_{ji} \le u_j - (1 - x_{ji}) & 2 \le i, j \le N \\
& 1 + (1 - x_{1j}) + (N-2) x_{j1} \le u_j & 2 \le j \le N \\
& u_j \le N - (N-2) x_{1j} - (1 - x_{j1}) & 2 \le j \le N \\
& x_{ij} \in \{0, 1\} & 1 \le i, j \le N, i \ne j \\
& y_{ij} \ge 0, y_{ij} \in \mathbb{Z} & 1 \le i, j \le N, i \ne j \\
& u_i \in \mathbb{Z}, 2 \le u_i \le N & 2 \le i \le N \\
& u_1 \in \mathbb{Z}, u_1 = 1 &
\end{aligned}
$$

Fig. 3.4: Formulación de *TSP-SD*

3.2. Modelos propuestos

A continuación se presentan los modelos para resolver el *POLP*, es decir, sea c la matriz de costos, determinar un ordenamiento lineal de los vértices tal que se minimice la suma de los productos entre el costo y la distancia para todo par de vértices. Se restringirá a estudiar el caso en el que la distancia entre pares de vértices se define como la cantidad de arcos que hay entre ellos, en este caso resulta que $d_{ij} = d_{ji}$. Con esta definición de distancia, se puede suponer que la matriz de costos es simétrica sin perdida de generalidad, ya que dada una instancia con matriz de costos asimétrica c se puede utilizar la matriz simétrica $c' = (c + c^T)/2$, en donde $d_{ij}c_{ij} + d_{ji}c_{ji} = d_{ij}(c_{ij} + c_{ji}) = d_{ij}(2c'_{ij})$.

3.2.1. Modelo 1 - Adaptación *POL*

El primer modelo propuesto es una adaptación del presentado en 3.1 para el *POL*. El problema de ordenamiento lineal no tiene variables para las distancias entre los vértices, pero cuenta con las variables que establecen que un vértice está antes que otro.

Por lo tanto, se puede extender agregando variables de distancias en donde la distancia entre un vértice i y un vértice j es la diferencia entre la cantidad de vértices que preceden a j y la cantidad de vértices que preceden a i. Esta extensión se puede ver en la siguiente formulación.

$$
\begin{aligned}
\text{minimizar} \quad & \sum_{\substack{1 \le i < j \le N \\ i \neq j}} c_{ij} d_{ij} & \\
\text{sujeto a} \quad & x_{ij} + x_{ji} = 1 & 1 \le i < j \le N \\
& x_{ij} + x_{jk} + x_{ki} \le 2 & 1 \le i < j, k \le N, k \neq i, j \\
& d_{ij} \ge \sum_{\substack{k=1 \\ k \neq i}}^{N} x_{ki} - \sum_{\substack{k=1 \\ k \neq j}}^{N} x_{kj} & 1 \le i < j \le N \\
& d_{ij} \ge \sum_{\substack{k=1 \\ k \neq j}}^{N} x_{kj} - \sum_{\substack{k=1 \\ k \neq i}}^{N} x_{ki} & 1 \le i < j \le N \\
& x_{ij} \in \{0, 1\} & 1 \le i, j \le N, i \neq j \\
& d_{ij} \ge 0, d_{ij} \in \mathbb{Z} & 1 \le i < j \le N
\end{aligned}
$$

Fig. 3.5: Adaptación de la formulación de *POL*

3.2.2. Modelo 2 - Adaptación *ELM*

El segundo modelo es una extensión del modelo de etiquetado lineal mínimo (3.1.2). La diferencia de este problema con el *POLP* es que el modelo anterior no incluye un costo asociado a las distancias, por lo que se agrega el costo a la función objetivo. Además la función objetivo suma sobre todos los pares de vértices i y j y no sólo un subconjunto de ellos. Las variables y restricciones de este modelo son iguales a las del modelo presentado en 3.1.2.

$$
\begin{aligned}
\text{minimizar} \quad & \sum_{1 \le i < j \le N} c_{ij} d_{ij} & \\
\text{sujeto a} \quad & \sum_{i=1}^{N} x_{ij} = 1 & 1 \le j \le N \\
& \sum_{j=1}^{N} x_{ij} = 1 & 1 \le i \le N \\
& d_{ij} \ge (q-p)(x_{ip} + x_{jq} - 1) & 1 \le i, j, p, q \le N, p < q, i < j \\
& d_{ij} \ge (q-p)(x_{jp} + x_{iq} - 1) & 1 \le i, j, p, q \le N, p < q, i < j \\
& x_{ij} \in \{0, 1\} & 1 \le i, j \le N \\
& d_{ij} \ge 0, d_{ij} \in \mathbb{Z} & 1 \le i, j \le N, i \ne j
\end{aligned}
$$

Fig. 3.6: Adaptación de la formulación de *ELM*

3.2.3. Modelo 3 - Adaptación *MTZ*

El tercer modelo está basado en la formulación del problema del viajante de comercio propuesta por Miller-Tucker-Zemlin (3.3). En dicha formulación existe un orden de recorrido de los vértices, sin embargo supone que es indistinto cuál es el primer vértice porque busca encontrar un ciclo. Por lo tanto, hace falta extender el modelo anterior para tener en cuenta que cualquier vértice puede estar en la primera posición.

Además se debe cambiar la función objetivo: mientras que antes la matriz c definía un costo por utilizar entre cada par de vértices, ahora se sabe que la distancia entre un par de vértices es exactamente la diferencia entre sus posiciones, y ésta tiene un costo asociado. Como en el modelo original de *MTZ* se definen variables de posición, es necesario agregar variables de distancia entre pares de vértices, donde éstas se calcularán como la diferencia entre las posiciones de los respectivos vértices.

$$
\begin{aligned}
\text{minimizar} \quad & \sum_{1 \le i < j \le N} c_{ij} d_{ij} & \\
\text{sujeto a} \quad & \sum_{1 \le i \le N} x_{ij} = 1 & 1 \le j \le N \\
& \sum_{1 \le j \le N} x_{ij} = 1 & 1 \le i \le N \\
& \sum_{1 \le i \le N} r_i = 1 & \\
& u_i + r_i N \le N + 1 & 1 \le i \le N \\
& u_i - u_j + x_{ij} N - r_j N \le N - 1 & 1 \le i, j \le N, i \ne j \\
& u_j - u_i \le d_{ij} & 1 \le i < j \le N \\
& u_i - u_j \le d_{ij} & 1 \le i < j \le N \\
& x_{ij} \in \{0, 1\} & 1 \le i, j \le N, i \ne j \\
& d_{ij} \ge 0, d_{ij} \in \mathbb{Z} & 1 \le i < j \le N, i \ne j \\
& u_i \in \mathbb{Z}, r_i \in \{0, 1\} & 1 \le i \le N
\end{aligned}
$$

Fig. 3.7: Adaptación de formulación MTZ de TSP

La diferencia entre la adaptación y el modelo original de *MTZ* (3.3) radica en las variables binarias r_i que indican si el i-ésimo vértice es el primero del recorrido. Las variables originales son las variables x_{ij} que indican que el arco que va de i a j se está usando y las variables u_i que representan en qué posición se encuentra el vértice i.

Las primeras dos restricciones definen que cada vértice tiene un arco de entrada y uno de salida. De esta manera se podría conseguir un conjunto de ciclos con vértices disjuntos, pero esto se evita utilizando las variables de posición.

La tercera restricción impone que sólo un vértice puede ser el primero del recorrido y la cuarta restricción fuerza a que el vértice que tiene la variable r_i en 1 esté en la posición 0.

Luego está la restricción del modelo de *MTZ* que asegura que si se usa el arco entre i y j, entonces la diferencia de las posiciones es 1. Esto vale para todos los casos salvo cuando j es el primer vértice, ya que en la solución de TSP se usa el arco que va del vértice N a 1. En el modelo original no hacía falta tener en cuenta esto porque se asumía que el primer vértice estaba en la primera posición, pero en este caso la desigualdad debe valer independientemente de cúal es el primer vértice. Por último la sexta restricción define la distancia entre dos vértices como la diferencia entre sus posiciones.

3.2.4. Modelo 4 - Adaptación *SD*

Por último se adaptó el modelo de Sherali-Driscol para el problema del viajante de comercio (3.4). Para esto se agregaron variables de distancia d con las respectivas restricciones, que determinan la distancia como diferencia entre las posiciones, y se modificó la función objetivo para incluirlas.

Dado que en el modelo original se considera que en la primera posición siempre estaba el vértice 1, también se adaptó para que cualquier vértice pueda estar primero. El modelo original de Sherali-Driscol tiene restricciones más complejas que el modelo original de

MTZ, resultando muy complejo incorporar las N variables indicando cuál es el primer vértice como en la adapatación de *MTZ*. En cambio, se decidió crear un vértice ficticio de manera que siempre se encontrara en la primera posición, pero se excluye de la función objetivo. De esta manera, los vértices ocupan las posiciones entre 2 y $N+1$. Como el interés para el problema está en la distancia entre pares de vértices esto no afecta al problema.

$$
\begin{aligned}
\text{minimizar} \quad & \sum_{2 \le i < j \le N+1} c_{ij} d_{ij} & \\
\text{sujeto a} \quad & \sum_{1 \le i \le N+1} x_{ij} = 1 & 1 \le j \le N+1 \\
& \sum_{1 \le j \le N+1} x_{ij} = 1 & 1 \le i \le N+1 \\
& \sum_{j=2}^{N+1} y_{ij} + N x_{i1} = u_i & 2 \le i \le N+1 \\
& \sum_{i=2}^{N+1} y_{ij} + 1 = u_j & 2 \le j \le N+1 \\
& x_{ij} \le y_{ij} \le (N-1) x_{ij} & 2 \le i, j \le N+1 \\
& u_j + (N-1) x_{ij} - N(1 - x_{ji}) \le y_{ij} + y_{ji} & 2 \le i, j \le N+1 \\
& y_{ij} + y_{ji} \le u_j - (1 - x_{ji}) & 2 \le i, j \le N+1 \\
& 1 + (1 - x_{1j}) + (N-2) x_{j1} \le u_j & 2 \le j \le N+1 \\
& u_j \le N - (N-2) x_{1j} - (1 - x_{j1}) & 2 \le j \le N+1 \\
& u_i - u_j \le d_{ij} & 1 \le i < j \le N, i \ne j \\
& u_j - u_i \le d_{ij} & 1 \le i < j \le N, i \ne j \\
& x_{ij} \in \{0, 1\} & 1 \le i, j \le N+1, i \ne j \\
& d_{ij} \ge 0, d_{ij} \in \mathbb{Z} & 1 \le i < j \le N+1, i \ne j \\
& y_{ij} \ge 0, y_{ij} \in \mathbb{Z} & 1 \le i, j \le N+1, i \ne j \\
& u_i \in \mathbb{Z}, 1 \le u_i \le N & 2 \le i \le N+1 \\
& u_1 \in \mathbb{Z}, u_1 = 0 &
\end{aligned}
$$

Fig. 3.8: Adaptación de formulación SD de TSP

3.3. Comparación y elección de modelo

El objetivo en esta etapa es seleccionar el modelo más prometedor para utilizar como base para explotar la estructura particular del problema. Utilizando los modelos presentados anteriormente se usó el algoritmo *Branch-and-Cut* genérico provisto por el paquete *CPLEX 12.4*[1] utilizando todos los parámetros y opciones que vienen por defecto. Los experimentos computacionales se realizaron en una computadora con procesador i7 a 3.4 Ghz con 8 núcleos y 16 Gb de memoria RAM.

Para realizar la comparación se ejecutaron todos los modelos sobre múltiples instancias de distintos tamaños, obteniendo el tiempo de ejecución y el gap alcanzado para cada una de ellas en un tiempo máximo de 10 minutos. Las instancias fueron generadas a partir de una matriz simétrica con coeficientes enteros positivos, en donde cada posición toma un valor equiprobable en un rango de valores. El propósito es identificar el modelo que resuelve mayor cantidad de instancias en menor tiempo y/o obtiene el menor gap al alcanzar el tiempo límite.

Las primeras cuatro tablas muestran la performance de cada modelo. Por cada tamaño se corrieron ocho instancias de prueba de las cuales se indica el gap obtenido. Además se muestra el tiempo máximo, mínimo y el promedio que se utilizó en alcanzar el óptimo para aquellas instancias en las que se logra resolver el problema.

La última tabla indica un detalle de cuánto tardó el algoritmo en obtener el óptimo para cada instancia si lo hizo, o el gap alcanzado al llegar al límite de tiempo.

El modelo que mejores resultados obtuvo fue la adaptación del *POL*. En las tablas se puede observar que en los 10 minutos se logran resolver algunas instancias de tamaño 14. En las instancias de mayor tamaño al terminar de procesar el nodo raíz se obtiene un gap de aproximadamente 70 %. Es importante notar que fue el único modelo que logró resolver instancias de tamaño mayor a 10.

La adaptación de *ELM* consiguió buenos resultados en algunas de las instancias más pequeñas pero para instancias medianas tiene peores tiempos que el anterior. Llega a resolver algunas instancias de tamaño 10 en el límite de tiempo. Se puede ver que para los primeros nodos procesados se tiene un gap de 100 %. Esto sucede porque la relajación lineal encuentra una asignación de variables de manera tal que la función objetivo vale 0. Una observación intersante, que no sucede con el modelo anterior, es que para aquellas instancias que logra resolver llega a la solución óptima teniendo entre 40 % y 50 % de gap y consume mucho tiempo en demostrar optimalidad.

Los otros dos modelos tampoco dieron buenos resultados. Mientras la adaptación de *MTZ* logró resolver todas las instancias hasta de tamaño 10, la adaptación de *SD* solo resolvió 7 de ellas. Es importante notar que mientras la adaptación de *MTZ* logra conseguir el óptimo en menor tiempo para las instancias resueltas, la adaptación de *SD* consigue un gap considerablemente menor para las instancias no resueltas. Esto se debe a que el modelo de Sherali-Driscol tiene relajaciones más fuertes, por lo que se pueden observar mejores cotas inferiores, aunque en ambos casos el gap en el nodo raíz es de 100 %. Lo que sucede en la adaptación de *MTZ* es que llega a un valor de gap cercano al 30 % en el cual alcanza una cota inferior global y el algoritmo termina descartando el árbol de búsqueda restante.

[1] `http://www-01.ibm.com/software/commerce/optimization/linear-programming/`

El primer modelo es el que reporta mejor desempeño y por lo tanto, fue el seleccionado para realizar el estudio poliedral y desarrollar el algoritmo *Branch-and-Cut*.

N	Min	Max	$\bar{t}$(s)	I1	I2	I3	I4	I5	I6	I7	I8
9	0.17	1.52	0.87	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
10	0.81	4.22	2.26	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
11	3.20	14.28	7.91	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
12	17.16	71.77	40.14	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
13	98.61	259.10	179.71	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
14	558.06	586.17	572.12	10.52 %	0.00 %	9.73 %	11.11 %	3.25 %	0.00 %	8.23 %	3.91 %

Tab. 3.1: Resultados por tamaño de entrada para la adaptación del *POL*

N	Min	Max	$\bar{t}$(s)	I1	I2	I3	I4	I5	I6	I7	I8
9	40.91	60.82	51.03	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
10	562.62	569.73	566.17	4.02 %	0.00 %	0.00 %	4.87 %	5.55 %	5.18 %	4.27 %	2.09 %
11				51.93 %	55.09 %	47.49 %	53.90 %	51.39 %	60.64 %	53.02 %	51.69 %
12				81.97 %	77.73 %	81.05 %	81.84 %	80.99 %	81.17 %	81.45 %	80.12 %
13				90.56 %	91.39 %	90.56 %	92.40 %	92.81 %	90.90 %	90.54 %	90.15 %
14				95.91 %	96.33 %	96.34 %	94.58 %	95.81 %	92.17 %	95.48 %	93.28 %

Tab. 3.2: Resultados por tamaño de entrada para la adaptación de *ELM*

N	Min	Max	$\bar{t}$(s)	I1	I2	I3	I4	I5	I6	I7	I8
9	16.71	17.48	17.11	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
10	229.60	235.08	232.82	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
11				74.97 %	73.06 %	74.27 %	75.25 %	75.36 %	76.17 %	73.83 %	73.02 %
12				88.14 %	87.31 %	86.52 %	88.05 %	88.40 %	87.58 %	88.29 %	88.07 %
13				91.58 %	90.87 %	91.62 %	91.82 %	91.73 %	91.03 %	91.48 %	91.55 %
14				92.78 %	93.88 %	94.00 %	94.10 %	93.54 %	93.84 %	93.67 %	93.76 %

Tab. 3.3: Resultados por tamaño de entrada para la adaptación de *MTZ*

N	Min	Max	$\bar{t}$(s)	I1	I2	I3	I4	I5	I6	I7	I8
9	22.41	35.17	28.77	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
10	255.94	427.83	329.05	3.93 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
11				35.53 %	27.54 %	29.60 %	33.41 %	36.68 %	38.71 %	36.44 %	35.82 %
12				47.68 %	47.50 %	46.89 %	49.46 %	48.24 %	49.63 %	49.32 %	49.88 %
13				54.07 %	53.41 %	55.53 %	54.22 %	52.15 %	51.96 %	55.43 %	53.35 %
14				58.23 %	58.24 %	60.15 %	59.87 %	58.49 %	61.74 %	60.63 %	59.30 %

Tab. 3.4: Resultados por tamaño de entrada para la adaptación de *SD*

N	#	Modelo 1	Modelo 2	Modelo 3	Modelo 4
9	1	0.95	52.16	16.89	28.66
9	2	0.17	47.73	16.91	22.41
9	3	0.61	57.00	16.71	32.27
9	4	1.42	58.29	16.92	32.55
9	5	1.00	48.47	17.39	35.17
9	6	1.35	60.82	17.39	28.99
9	7	0.44	40.91	17.18	25.28
9	8	0.46	42.85	17.48	24.82
10	1	4.22	(4.02 %)	231.24	(3.93 %)
10	2	2.16	569.73	229.60	328.55
10	3	1.31	562.62	233.02	255.94
10	4	1.33	(4.87 %)	235.08	287.95
10	5	1.73	(5.55 %)	232.13	337.21
10	6	2.71	(5.18 %)	234.71	342.70
10	7	3.84	(4.27 %)	234.91	427.83
10	8	0.81	(2.09 %)	231.86	323.20
11	1	10.42	(51.93 %)	(74.97 %)	(35.53 %)
11	2	3.20	(55.09 %)	(73.06 %)	(27.54 %)
11	3	5.82	(47.49 %)	(74.27 %)	(29.60 %)
11	4	9.07	(53.90 %)	(75.25 %)	(33.41 %)
11	5	8.49	(51.39 %)	(75.36 %)	(36.68 %)
11	6	14.28	(60.64 %)	(76.17 %)	(38.71 %)
11	7	5.25	(53.02 %)	(73.83 %)	(36.44 %)
11	8	6.71	(51.69 %)	(73.02 %)	(35.82 %)
12	1	61.28	(81.97 %)	(88.14 %)	(47.68 %)
12	2	19.00	(77.73 %)	(87.31 %)	(47.50 %)
12	3	17.16	(81.05 %)	(86.52 %)	(46.89 %)
12	4	51.55	(81.84 %)	(88.05 %)	(49.46 %)
12	5	43.02	(80.99 %)	(88.40 %)	(48.24 %)
12	6	22.96	(81.17 %)	(87.58 %)	(49.63 %)
12	7	71.77	(81.45 %)	(88.29 %)	(49.32 %)
12	8	34.36	(80.12 %)	(88.07 %)	(49.88 %)
13	1	205.40	(90.56 %)	(91.58 %)	(54.07 %)
13	2	144.60	(91.39 %)	(90.87 %)	(53.41 %)
13	3	183.40	(90.56 %)	(91.62 %)	(55.53 %)
13	4	259.05	(92.40 %)	(91.82 %)	(54.22 %)
13	5	98.61	(92.81 %)	(91.73 %)	(52.15 %)
13	6	156.43	(90.90 %)	(91.03 %)	(51.96 %)
13	7	259.10	(90.54 %)	(91.48 %)	(55.43 %)
13	8	131.11	(90.15 %)	(91.55 %)	(53.35 %)
14	1	(10.52 %)	(95.91 %)	(92.78 %)	(58.23 %)
14	2	558.06	(96.33 %)	(93.88 %)	(58.24 %)
14	3	(9.73 %)	(96.34 %)	(94.00 %)	(60.15 %)
14	4	(11.11 %)	(94.58 %)	(94.10 %)	(59.87 %)
14	5	(3.25 %)	(95.81 %)	(93.54 %)	(58.49 %)
14	6	586.17	(92.17 %)	(93.84 %)	(61.74 %)
14	7	(8.23 %)	(95.48 %)	(93.67 %)	(60.63 %)
14	8	(3.91 %)	(93.28 %)	(93.76 %)	(59.30 %)
	Total	40	0	0	

Tab. 3.5: Resultados para intancias de tamaño 9-14

4

Estudio Poliedral

Un modelo de programación lineal entera tiene asociado un poliedro que contiene su conjunto de soluciones factibles. Teniendo un modelo de PEM para el *POLP*, un algoritmo *Branch-and-Cut* puede calcular una cota inferior del valor óptimo de la función objetivo calculando la relajación lineal del modelo.

Sea $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ el poliedro asociado a la relajación lineal, es decir todos los puntos reales que cumplen las restricciones del modelo. Se define $S = P \cap \{x \in \mathbb{Z}^n\}$ y $conv(S)$ como el menor poliedro que contiene a S. Si la relajación lineal del modelo de PEM es $conv(S)$ entonces calculando el óptimo de esta relajación lineal se podría determinar el óptimo del modelo de PEM; pero como se ha dicho anteriormente, no es fácil tener esta descripción. Por lo tanto, se estudian características de $conv(S)$ para ser incluidas al modelo y así tener relajaciones lineales que estén más cerca de puntos factibles para el problema, proveyendo mejores cotas inferiores.

En el caso del modelo seleccionado, no todos los elementos de S constituyen puntos factibles del problema ya que las variables de distancia no se encuentran ajustadas. Por lo tanto el conjunto de puntos que interesa estudiar es el definido por
$R = \{(x, d) \ \in \ S \ | \ \forall \ (x, \bar{d}) \in \ S \ (x, d) <= \ (x, \bar{d})\}$.

El objetivo de este capítulo es presentar las características de $conv(R)$ que se pudieron encontrar y que se utilizarán para realizar planos de corte en el algoritmo propuesto.

4.1. Modelo elegido

Considerando la experimentación computacional analizada, la adaptación del *POL* resultó ser el modelo que mostró mejor performance al ser resuelto mediante el algoritmo de *Branch-and-Cut* provisto por defecto por CPLEX.

Recordemos que en este modelo se consideran variables x_{ij} para todo par de vértices que indican si el vértice i esta ubicado en una posición anterior al vértice j. También existen variables d_{ij} con $i < j$ que indican la distancia entre el vértice i y el vértice j, donde la distancia se calcula como la diferencia entre sus posiciones en el ordenamiento lineal. Además se cuenta con un costo c_{ij} con $i < j$ que indica el costo entre los vértices i y j. Se considera que los costos son positivos y simétricos.

$$
\begin{aligned}
\text{minimizar} \quad & \sum_{1 \le i < j \le N} c_{ij} d_{ij} & \\
\text{sujeto a} \quad & x_{ij} + x_{ji} = 1 & 1 \le i < j \le N \\
& x_{ij} + x_{jk} + x_{ki} \le 2 & 1 \le i < j, k \le N \\
& d_{ij} \ge \sum_{\substack{k=1 \\ k \ne i}}^{N} x_{ki} - \sum_{\substack{k=1 \\ k \ne j}}^{N} x_{kj} & 1 \le i < j \le N \\
& d_{ij} \ge \sum_{\substack{k=1 \\ k \ne j}}^{N} x_{kj} - \sum_{\substack{k=1 \\ k \ne i}}^{N} x_{ki} & 1 \le i < j \le N \\
& x_{ij} \in \{0, 1\} & 1 \le i, j \le N, i \ne j \\
& d_{ij} \ge 0, d_{ij} \in \mathbb{Z} & 1 \le i < j \le N
\end{aligned}
$$

Fig. 4.1: Adaptación de la formulación de *POL*

4.2. Igualdades del Sistema Minimal

Un aspecto importante del estudio poliedral es encontrar un sistema minimal de ecuaciones, esto es un conjunto de igualdades linealmente independientes que valen para todos los puntos factibles. Teniendo un conjunto de igualdades es posible reducir la dimensión del problema ya que es posible determinar el valor de algunas variables en función de otras.

En toda solución factible del modelo, independientemente de qué vértices ocupen cada posición, la suma de las distancias es constante, esto es expresado en la ecuación (4.1). Esto es así porque el conjunto $\{d_{ij} | 1 \le i < j \le N - 1\}$ de valores que pueden tomar las variables de distancia es único.

$$\sum_{1 \le i < j \le N} d_{ij} = (N-1)N(N+1)/6 \tag{4.1}$$

Computacionalmente se puede ver que la igualdad anterior, junto con las igualdades presentes en el modelo, son las únicas válidas para instancias de hasta 10 vértices (es decir: no existe otra igualdad válida linealmente independiente con estas) y se conjetura que no existen otras igualdades válidas linealmente independientes a la anterior.

4.3. Desigualdades Válidas

El objetivo de esta sección es presentar desigualdades válidas que caracterizan parcialmente $conv(S)$, para luego ser introducidas como planos de corte en el algoritmo *Branch-and-Cut*.

Mediante la herramienta *PORTA* [1] (POlyhedron Representation Transformation Algorithm) es posible obtener en instancias de tamaños chicos todas las desigualdades que describen completamente al poliedro, luego caracterizarlas y evaluar cómo funcionan las desigualdes para tamaños más grandes. Sólo se pudo entontrar la descripción completa

[1] `http://typo.zib.de/opt-long_projects/Software/Porta/`

del poliedro en instancias de tamaño 4 en donde la cápsula convexa es descripta con 327 desigualdades, mientras que el caso de tamaño 5 contaba con más de 10^9 desigualdades. A partir de dichas descripciones, se logró caracterizar varias familias de desigualdades válidas. Cabe señalar que en todas las instancias en las que fue posible correr el software resultaron ser facetas y se verificó computacionalmente que existen puntos que pueden distinguir cada una de ellas.

Las desigualdades encontradas fueron las siguientes.

Desigualdad tipo 1 Para todo par de vértices i y j, el máximo valor que puede tomar $\sum_{\substack{1\leq k\leq N\\ k\neq i,k\neq j}}(x_{ik}+x_{jk})$ es $2(N-2)$, suponiendo que los vértices i y j estan al comienzo del orden. Si en un orden la distancia entre i y j es d_{ij}, existirán $d_{ij}-1$ vértices que están antes que uno y después que el otro, por lo tanto
$\sum_{\substack{1\leq k\leq N\\ k\neq i,k\neq j}}(x_{ik}+x_{jk}) \leq 2(N-2)-(d_{ij}-1) = 2N-3-d_{ij}$. Es posible hacer el mismo análisis considerando los vértices que quedan depués de i y j en lugar de antes.

$$d_{ij}+\sum_{\substack{1\leq k\leq N\\ k\neq i,k\neq j}}(x_{ik}+x_{jk}) \leq 2N-3 \qquad 1\leq i<j\leq N \tag{4.2}$$

$$d_{ij}+\sum_{\substack{1\leq k\leq N\\ k\neq i,k\neq j}}(x_{ki}+x_{kj}) \leq 2N-3 \qquad 1\leq i<j\leq N \tag{4.3}$$

Desigualdad tipo 2 Las desigualdades (4.4) y (4.5) establecen que la suma de las distancias con respecto a un vértice depende del valor de algunas variables de precedencia. Existe una cantidad exponencial de este tipo de desigualdades ya que para cada subconjunto de vértices, existe una desigualdad.

$$\sum_{\substack{1\leq j\leq N\\ j\neq i}} d_{ij}+\sum_{k=1}^{\lfloor\frac{(N-1)}{2}\rfloor}((N-2k)(x_{S_{2k-1}i}-x_{S_{2k}i})) \leq \frac{N(N-1)}{2} \qquad \begin{array}{c}1\leq i\leq N, S\subset\{1,\ldots,N\}\\ i\notin S, |S|=2\lfloor\frac{N-1}{2}\rfloor\end{array} \tag{4.4}$$

$$\sum_{\substack{1\leq j\leq N\\ j\neq i}} d_{ij}+\sum_{k=1}^{\lfloor\frac{(N-1)}{2}\rfloor}((N-2k)(x_{iS_{2k-1}}-x_{iS_{2k}})) \leq \frac{N(N-1)}{2} \qquad \begin{array}{c}1\leq i\leq N, S\subset\{1,\ldots,N\}\\ i\notin S, |S|=2\lfloor\frac{N-1}{2}\rfloor\end{array} \tag{4.5}$$

Se puede demostrar la validez de dicha desigualdad para toda solución factible del conjunto R. Para cualquier vértice i existe un conjunto U de vértices que aparecen antes y un conjunto V que aparecen después, es decir, no existe intersección entre U y V.

Se puede ver que la suma de todas las distancias con respecto a i es $\frac{|U|(|U|+1)+|V|(|V|+1)}{2}$.

Por otro lado, tomando a r como la diferencia entre $|U|$ y $|V|$, y un conjunto ordenado $S\subset\{1,\ldots,N\}\setminus\{i\}$, el máximo valor que puede tomar $\sum_{k=1}^{\lfloor\frac{(N-1)}{2}\rfloor}((N-2k)(x_{S_{2k-1}i}-x_{S_{2k}i}))$ es $r(N-r-1)$, ubicando la mayor cantidad posible de índices de los vértices de U en las posiciones impares de S.

Suponiendo que $|U| \geq |V|$ entonces $r = |U| - |V|$. Por lo tanto

$$\sum_{\substack{1\leq j\leq N\\ j\neq i}} d_{ij} + \sum_{k=1}^{\lfloor\frac{(N-1)}{2}\rfloor} ((N-2k)(x_{S_{2k-1}i} - x_{S_{2k}i})) \leq \frac{|U|(|U|+1)+|V|(|V|+1)}{2} + (|U|-|V|)(N-(|U|-|V|)-1)$$

que alcanza el valor máximo cuando $|U| = N-1$ y $|V| = 0$ y en ese caso

$$\sum_{\substack{1\leq j\leq N\\ j\neq i}} d_{ij} + \sum_{k=1}^{\lfloor\frac{(N-1)}{2}\rfloor} ((N-2k)(x_{S_{2k-1}i} - x_{S_{2k}i})) \leq \frac{(N-1)((N-1)+1)}{2} + (N-1)(N-(N-1)-1) = \frac{(N-1)N}{2}.$$

Desigualdad tipo 3 Las desigualdades (4.6) y (4.7) muestran que la suma de las distancias de un punto a todos los demás está acotada inferiormente. Es decir, hay un posición que minimiza la distancia con el resto, ésta es cuando el vértice está en el medio del orden. En las instancias probadas (4.6) sólo resultó ser faceta cuando la cantidad de vértices es par. Por otro lado, en el caso impar no es suficiente, porque en estos casos la posición del medio es única, por lo que la desigualdad (4.7) agrega la información de que cuando el vértice está en la posición de mínima distancia con el resto, además tiene $(N-1)/2$ vértices adelante.

$$\sum_{\substack{1\leq j\leq N\\ i\neq j}} d_{ij} \geq \lfloor\frac{N}{2}\rfloor\lceil\frac{N}{2}\rceil \qquad 1 \leq i \leq N \tag{4.6}$$

$$\sum_{\substack{1\leq j\leq N\\ i\neq j}} d_{ij} + \sum_{\substack{1\leq j\leq N\\ i\neq j}} x_{ij} \geq \lfloor\frac{N}{2}\rfloor\lceil\frac{N}{2}\rceil + \frac{(N-1)}{2} \qquad 1 \leq i \leq N \tag{4.7}$$

Desigualdad tipo 4 La desigualdad (4.8) impone una cota superior a la suma de las distancias entre tres vértices. Para cada terna de vértices i,j y k se cumple que alguno de los tres se encuentra posicionado entre los otros dos. Sea j el que se encuentra después de i y antes que k, vale que $d_{ij} + d_{jk} = d_{ik}$. Como $d_{ik} \leq N-1$ entonces vale que la suma de las tres distancias esta acotada por dos veces la máxima distancia. Existe una cantidad cúbica respecto a la cantidad de vértices de este tipo de desigualdades.

$$d_{ij} + d_{jk} + d_{ik} \leq 2N - 2 \qquad 1 \leq i < j < k \leq N \tag{4.8}$$

Desigualdad tipo 5 La desigualdad (4.9) indica una cota superior para ciertos conjuntos de variables. Estos son conjuntos maximales de variables de distancias entre vértices tal que no existen dos de ellas que correspondan a un mismo vértice. En el caso de cantidad de vértices impar, existirá un vértice que no es considerado por el conjunto anterior, y en este caso toda solución factible verifica (4.10) y (4.11).

Para esta familia también existe una cantidad exponencial de desigualdades.

$$\sum_{i=1}^{N/2} d_{S_{2i-1}S_{2i}} \leq \frac{N^2}{4} \qquad \forall S\ permutacion\{1,\ldots,N\} \tag{4.9}$$

$$\sum_{i=1}^{(N-1)/2} d_{S_{2i-1}S_{2i}} + \sum_{i=1}^{N-1} x_{S_iS_N} \leq \frac{(N-1)(N+3)}{4} \qquad \forall S\ permutacion\{1,\ldots,N\} \tag{4.10}$$

$$\sum_{i=1}^{(N-1)/2} d_{S_{2i-1}S_{2i}} + \sum_{i=1}^{N-1} x_{S_N S_i} \leq \frac{(N-1)(N+3)}{4} \qquad \forall S \; permutacion\{1,\ldots,N\} \quad (4.11)$$

Desigualdad tipo 6 La desigualdad (4.12) indica que la distancia entre dos vértices está acotada inferiormente por la cantidad de vértices que aparecen antes que alguna y después que otra. Se nota $\sum_k(\pm x_{ki} \mp x_{kj})$ como $+x_{ki} - x_{kj}$ o $-x_{ki} + x_{kj}$. Es decir, que para cada par de vértices i y j existe una cantidad de vértices $d_{ij} - 1$ que verifican que $x_{ki} \neq x_{kj}$ (ya que si son iguales $\pm x_{ki} \mp x_{kj} = 0$). Se puede ver que para cualquier combinación existe una solución factible que verifique la desigualdad. Existe una cantidad exponencial de desigualdades de este tipo.

$$d_{ij} + \sum_{\substack{1 \leq k \leq N \\ i \neq k, j \neq k}} (\pm x_{ki} \mp x_{kj}) \geq 1 \qquad 1 \leq i < j \leq N \quad (4.12)$$

Desigualdad tipo 7 La desigualdad (4.13) indica que si existe un vértice k entre i y j, entonces se verifica que $d_{ik} + d_{jk} = d_{ij}$ y además vale que k aparece después de i y antes que j o vice versa. Hay una cantidad cúbica de este tipo de desigualdades.

$$d_{ij} - d_{ik} - d_{jk} \pm 2x_{ik} \mp 2x_{kj} \leq 0 \qquad \begin{matrix} 1 \leq i < j \leq N, 1 \leq k \leq N \\ k \neq i, k \neq j \end{matrix} \quad (4.13)$$

Para verificar la validez separamos en tres casos:

- El vértice k esta ubicado entre los vértices i y j
  En este caso los vértices pueden estar ordenados (i,k,j) o (j,k,i), en ambos casos $d_{ij} - d_{ik} - d_{jk} = 0$, $2x_{ik} - 2x_{kj} = 0$ y $-2x_{ik} + 2x_{kj} = 0$. De esta manera la desigualdad (4.13) se cumple por igualdad.

- El vértice i esta ubicado entre los vértices j y k
  En este caso los vértices estan ordenados (j,i,k) o (k,i,j), en estos casos vale

  $$d_{ij} + d_{ik} = d_{jk}$$

  $$d_{ij} = d_{jk} - d_{ik}$$

  y debido a la que la distancia entre dos vértices es mayor o igual a uno vale:

  $$d_{ij} - d_{ik} - d_{jk} = d_{jk} - d_{ik} - d_{ik} - d_{jk} = -2d_{ik} \leq -2 \quad (4.14)$$

  Por otro lado

  $$\begin{cases} +2x_{ik} - 2x_{kj} = 2 & \text{en el caso } (j,i,k) \\ +2x_{ik} - 2x_{kj} = -2 & \text{en el caso } (k,i,j) \end{cases}$$

  En ambos casos por 4.14 vale

  $$d_{ij} - d_{ik} - d_{jk} + 2x_{ik} - 2x_{kj} \leq 0$$

  Además, alternando los signos se cumple:

  $$\begin{cases} -2x_{ik} + 2x_{kj} = -2 & \text{en el caso } (j,i,k) \\ -2x_{ik} + 2x_{kj} = 2 & \text{en el caso } (k,i,j) \end{cases}$$

Que junto con 4.14 verifica

$$d_{ij} - d_{ik} - d_{jk} - 2x_{ik} + 2x_{kj} \leq 0$$

- El vértice j esta ubicado entre los vértices i y k
  En este caso los vértices pueden estar ordenados (i, j, k) o (k, j, i). El análisis para este caso es análogo al caso anterior.

Como vimos que la desigualdad es válida en los tres casos entonces para cualquier solución factible vale la desigualdad.

Desigualdad tipo 8 La desigualdad (4.15) es similar a la anterior, pero agregando el hecho de que si k aparece después de i y antes que j entonces i aparece antes que j. También existe una de estas desigualdades para cualquier tres vértices, por lo que hay una cantidad cúbica con respecto a la cantidad de vértices. Se puede realizar la demostración de la validez utilizando argumentos similares a los de la desigualdad anterior.

$$d_{ij} - d_{ik} - d_{jk} + 2x_{ik} + 2x_{kj} - 4x_{ij} \leq 0 \qquad \begin{matrix} 1 \leq i, j, k \leq N \\ i \neq j, i \neq k, j \neq k \end{matrix} \tag{4.15}$$

Desigualdad tipo 9 La desigualdad (4.16) indica que si i está antes que j, entonces tiene que haber $d_{ij} - 1$ vértices que aparezcan antes que j y no aparezcan antes que i. Hay una cantidad cuadrática de este tipo de desigualdades.

$$d_{ij} + (2N-4)x_{ij} + \sum_{\substack{1 \leq k \leq N \\ i \neq k, j \neq k}} (x_{ki} - x_{kj}) \leq 2N - 3 \qquad 1 \leq i < j \leq N \tag{4.16}$$

Para mostrar la válidez se muestra que para cualquier solución factible valen todas las desigualdades. Para toda desigualdad se divide en dos casos:

- El vértice i esta antes que el vértice j.
  En este caso se tiene que
  $$(2N-4)x_{ij} = 2N - 4$$
  y todos los vértices que están antes que i también están antes que j. La cantidad de vértices distintos a i y j que estan entre ellos dos es $d_{ij} - 1$.
  $$\sum_{\substack{1 \leq k \leq N \\ i \neq k, j \neq k}} (x_{ki} - x_{kj}) = -(d_{ij} - 1)$$
  Sumando las ecuaciones con d_{ij} se cumple
  $$d_{ij} + (2N-4)x_{ij} + \sum_{\substack{1 \leq k \leq N \\ i \neq k, j \neq k}} (x_{ki} - x_{kj}) = d_{ij} + (2N-4) - (d_{ij} - 1) = 2N - 3$$

- El vértice j esta antes que el vértice i.
  $$d_{ij} \leq N - 1$$
  $$(2N-4)x_{ij} = 0$$

Además, en este caso todos los vértices que estan antes que el vértice j también estan antes que i.

$$\sum_{\substack{1\leq k\leq N\\ i\neq k, j\neq k}} (x_{ki} - x_{kj}) = d_{ij} - 1 \leq N - 2$$

Sumando las tres ecuaciones se cumple

$$d_{ij} + (2N-4)x_{ij} + \sum_{\substack{1\leq k\leq N\\ i\neq k, j\neq k}} (x_{ki} - x_{kj}) \leq 2N - 3$$

4.4. Desigualdades del poliedro del *POL*

Dado que nuestro modelo está basado en el modelo propuesto por Grötschel [12], las desigualdades válidas derivadas para el *POL* también lo serán para el *POLP*.

Desigualdades anti-ciclo Estas desigualdades son las que se utilizan en el modelo del *POL*. Establecen que no puede existir un ciclo entre las variables.

$$x_{ij} + x_{jk} + x_{ki} \leq 2 \tag{4.17}$$

Desigualdades k-fence Fueron estudiadas por Grötschel [12] como facetas del *POL*.

Sean $U = \{u_1, \ldots, u_k\}$, $W = \{w_1, \ldots, w_k\} \subseteq V$ conjuntos de vértices disjuntos de cardinalidad $3 \leq k \leq N/2$. Se denomina *k-fence* al conjunto de arcos

$$A = \{(u_i, w_i) \mid i = 1, \ldots, k\} \cup \{(w_i, u_j) \mid i, j \in \{1, \ldots, k\}, i \neq j\}.$$

La desigualdad *k-fence* es:

$$\sum_{(i,j)\in A} x_{ij} \leq k^2 - k + 1. \tag{4.18}$$

5

Algoritmo Branch-and-Cut

El propósito de este capítulo es describir las características principales del algoritmo *Branch-and-Cut* para resolver el *POLP*.

La performance del algoritmo depende de muchos factores. Una buena relajación lineal y planos de corte con procedimientos de separación adecuados, ayuda a mejorar las cotas inferiores. Esto es clave para podar ramas del árbol y disminuir el espacio de búsqueda y es aquí donde el estudio poliedral es fundamental para considerar buenas desigualdades válidas.

Por otro lado, las estrategias de recorrido del árbol, selección de variables y algunos otros detalles de implementación aportan al conjunto del comportamiento del algoritmo. No hay una elección de cada una de las alternativas del algoritmo óptima para cualquier instancia. Las características propias del problema son las que ayudan a determinar una buena elección.

En las próximas secciones se describe cada uno los factores que conforman nuestro algoritmo. La presentación del algoritmo está dividida en dos partes. En la primera se desarrollan las características asociadas al esquema de *Branch-and-Bound*. En la segunda parte, se muestran los algoritmos de separación y otros detalles vinculados a la etapa de *Cut*.

5.1. Branch-and-Bound

5.1.1. Selección de variable de *branching*

El modelo tiene dos tipos de variables: las variables de distancia, que son variables enteras entre 1 y $N-1$, y las variables de precedencia, que son variables binarias. El paquete CPLEX permite asignar prioridades que serán utilizadas para decidir cuál será la variable a utilizar, teniendo mayor peso aquellas que tengan mayor prioridad. Se decidió explorar tres posibilidades para seleccionar la variable por la cual se realiza el *branching*.

- Asignar prioridad a las variables x_{ij} de manera que se seleccione primero las variables binarias. La idea de esto es que si una variable binaria cuenta con un valor fraccional

en la relajación lineal el nodo se divide en dos de manera tal que en uno de los nodos se fuerza a la variable a valer 0 y en el otro 1.

- Asignar prioridad a las variables d_{ij} de manera que se seleccione primero las variables enteras generales. La idea es que si en la relajación lineal asociada al nodo actual existe una variable d_{ij} con valor fraccionario, d^*_{ij}, se divide en los casos $d_{ij} \leq \lfloor d^*_{ij} \rfloor$ y $d_{ij} \geq \lceil d^*_{ij} \rceil$.
- Dejar que la librería CPLEX decida automáticamente. Este procedimiento no se encuentra documentado.

5.1.2. Estrategias de recorrido del árbol

Existen dos estrategias principales para recorrer el árbol.

- **Depth-first search** Elige el nodo que fue creado más recientemente. El objetivo es poder encontrar soluciones buenas rápido, consiguiendo así mejores cotas superiores.
- **Best-first search** Elige el nodo con el mejor valor de la función objetivo de la relajación lineal asociada al nodo.

5.1.3. Heurística Inicial

Antes de comenzar el *Branch-and-Bound* se ejecuta una heurística inicial para proveer una solución factible. El objetivo de esto es tener una cota superior para el problema, lo cual permitirá cerrar nodos si el valor de la función objetivo en la relajación lineal asociada provee un valor superior a la solución encontrada.

La heurística consiste en armar un orden de los vértices. Comienza fijando las primeras posiciones (notamos k la cantidad de posiciones fijas) y asigna al resto de los vértices golosamente. Para asignar un vértice a una posición se procede de la siguiente manera: para cada posición se busca aquel que no fue utilizado y tiene el menor costo acumulado, donde el costo acumulado se calcula como la suma de los costos respecto a los vértices ya posicionados. Se prueban todas las posibilidades de ubicar los vértices en las primeras posiciones es $\frac{n!}{(n-k)!} \in \mathcal{O}(n^k)$ y en cada posición se realiza una búsqueda lineal para determinar cuál es el mejor vértice, para la cual determinar el costo asociado a dicho vértice también es $\mathcal{O}(n)$, ya que recorre todos los vértices ya posicionados. Por lo tanto, la heurística inicial tiene complejidad temporal $\mathcal{O}(n^{(k+2)})$. Mediante pruebas experimentales los mejores resultados se obtuvieron con $k = 3$.

Una vez fijado el orden se realiza una búsqueda local para mejorar la solución. Se define la vecindad de una solución como todas aquellas que resultan al aplicar reglas del siguiente estilo:

$$\langle 1, 2, \ldots, i{-}1, i, i{+}1, \ldots, j{-}1, j, j{+}1, \ldots, n\rangle \rightarrow \langle 1, 2, \ldots, i{-}1, i{+}1, \ldots, j{-}1, i, j, j{+}1, \ldots, n\rangle$$

Para realizar la búsqueda local se explora toda la vecindad de una solución y se conserva la que tiene mejor valor de la función objetivo. Si existe una solución en la vecindad que mejore el valor de la función objetivo, se reemplaza la solución anterior y se vuelve a realizar una búsqueda local. Sino la búsqueda local termina. El tamaño de la vecindad

es del orden de $\mathcal{O}(n^2)$, mientras que la complejidad temporal de obtener el valor de la función objetivo es $\mathcal{O}(n)$, por lo que el tiempo utilizado en cada iteración de la búsqueda es del orden de $\mathcal{O}(n^3)$.

5.1.4. Heurística Primal

La heurística primal se ejecuta en los nodos del árbol de búsqueda, el objetivo es poder encontrar soluciones para el problema a partir de la información obtenida en la relajación lineal asociada al nodo. En caso de encontrar una mejor solución se encontrará una mejor cota superior que servirá para podar nodos del árbol *Branch-and-Bound*.

Para construir dicho orden primero se busca un par de vértices que se encuentren lo más alejados en la relajación lineal asociada al nodo. Esto se obtiene utilizando vértices que corresponden a los índices del máximo d^*_{ij}. Sea u uno de dichos vértices y sea $D = \{d_{ui} \mid 1 \leq i \leq N, i \neq u\}$. Se ordenan las variables de D por su valor, d^*_{ij}, en forma creciente y se utilizan los índices de las variables como el orden que corresponde a la solución.

Una vez obtenida una solución factible se realiza una búsqueda local como la descripta para la heurística inicial 5.1.3, cuya solución finalmente es comparada con la solución actual del problema.

5.2. Planos de Corte

El objetivo de esta sección es explicar los algoritmos de separación desarrollados para las desigualdades válidas presentadas en el capítulo anterior.

Dado que en cada nodo se resuelve la relajación lineal y que el tiempo que tarda dicha resolución depende en parte de la cantidad de restricciones, se decidió que todas aquellas familias de desigualdades que tienen cardinalidad de al menos $\mathcal{O}(n^3)$ (donde n es la cantidad de vértices) no estarán explícitamente incorporadas al modelo y serán introducidas como planos de corte.

5.2.1. Algoritmos de separación

A continuación se describen los algoritmos de separación por cada familia de desigualdades.

- **Desigualdades anti-ciclo**

$$x_{ij} + x_{jk} + x_{ki} \leq 2 \tag{5.1}$$

Dichas desigualdades son las desigualdades originales del problema de ordenamiento lineal, éstas impiden que existan ciclos entre las variables de precedencia. El algoritmo de separación consiste en enumerar todas las posibilidades y evaluar si están siendo violadas en la relajación lineal. La complejidad temporal del algoritmo es $\mathcal{O}(n^3)$.

- **Desigualdades tipo 2**

$$\sum_{\substack{1\leq j\leq N\\ j\neq i}} d_{ij} + \sum_{k=1}^{\lfloor\frac{(N-1)}{2}\rfloor} ((N-2k)(x_{s_{2k-1}i} - x_{s_{2k}i})) \leq \frac{N(N-1)}{2} \qquad \begin{array}{c} 1 \leq i \leq N, s \subset \{1,\ldots,N\} \\ i \notin s, |s| = 2\lfloor\frac{N-1}{2}\rfloor \end{array} \tag{5.2}$$

$$\sum_{\substack{1\leq j\leq N\\ j\neq i}} d_{ij} + \sum_{k=1}^{\lfloor\frac{(N-1)}{2}\rfloor} ((N-2k)(x_{is_{2k-1}} - x_{is_{2k}})) \leq \frac{N(N-1)}{2} \qquad \begin{array}{c} 1 \leq i \leq N, s \subset \{1,\ldots,N\} \\ i \notin s, |s| = 2\lfloor\frac{N-1}{2}\rfloor \end{array} \tag{5.3}$$

Para esta familia de desigualdades se utiliza una heurística para encontrar alguna desigualdad violada. La desigualdad es más ajustada cuando el vértice i se encuentra en una de las puntas del orden, ya que en estos casos la distancia contra todos los demás vértices es máxima. Entonces primero se encuentran los vértices que corresponden a las puntas del orden en la relajación lineal. Para esto se buscan los vértices i y j que cumplan que d^*_{ij} sea máximo.

Luego se debe encontrar un conjunto ordenado $S \subseteq \{1,\ldots,N\} \setminus i$ que viole la desigualdad. Hay $2\lfloor\frac{(N-1)}{2}\rfloor$ términos acompañando variables x, la mitad de ellos con coeficiente positivo y la otra mitad con coeficiente negativo. Como los coeficientes que acompañan las variables x son todos distintos y se quiere maximizar la suma para que no valga la desigualdad, entonces se ordenan las variables x_{ij}, las de menor valor acompañan a los coeficientes negativos, y las de mayor valor acompañan a los coeficientes positivos. Se puede ver que una vez fijado el i, esta asignación golosa encuentra el máximo valor que puede tomar el lado izquierdo.

Puede pasar que para algún i se viole la desigualdad pero este no cumpla que en la relajación lineal tenga asociada la distancia de mayor valor, por lo que este procedimiento no garantiza que de existir un corte posible se identificará. La complejidad temporal del algoritmo de separación es de $\mathcal{O}(n^2)$ ya que se evalúan todas las distancias entre pares de vértices para decidir los vértices de los extremos.

- **Desigualdades tipo 4**

$$d_{ij} + d_{jk} + d_{ik} \leq 2N - 2 \qquad 1 \leq i < j < k \leq N \tag{5.4}$$

Al igual que las desigualdades anti-ciclo, existe una cantidad cúbica respecto a la cantidad de vértices. El procedimiento para encontrar es exhaustívamente probar todas las desigualdades, por lo que el algoritmo de separación tiene complejidad temporal $\mathcal{O}(n^3)$.

- **Desigualdades tipo 5**

$$\sum_{i=1}^{N/2} d_{s_{2i-1}s_{2i}} \leq \frac{N^2}{4} \qquad \forall S \; permutacion\{1,\ldots,N\} \tag{5.5}$$

$$\sum_{i=1}^{(N-1)/2} d_{s_{2i-1}s_{2i}} + \sum_{i=1}^{N-1} x_{s_i s_N} \leq \frac{(N-1)(N+3)}{4} \qquad \forall S \; permutacion\{1,\ldots,N\} \tag{5.6}$$

$$\sum_{i=1}^{(N-1)/2} d_{s_{2i-1}s_{2i}} + \sum_{i=1}^{N-1} x_{s_N s_i} \leq \frac{(N-1)(N+3)}{4} \qquad \forall S \; permutacion\{1,\ldots,N\} \tag{5.7}$$

El algoritmo de separación ejecuta una heurísitica para determinar si existe alguna violada, es decir no asegura encontrarla en caso de que exista. La desigualdad contiene una sumatoria de distancias entre vértices disjuntos. Para encontrar una desigualdad violada se deberá maximizar dicha suma.

En el caso par se ordenan todas las distancias por el valor en la relajación y se recorren, de mayor a menor, acumulando aquellas variables de distancia tales que no posean índices en común con las acumuladas previamente.

Para el caso impar, luego de aplicar este procedimiento, existirá un vértice que no se consideró en la acumulación anterior. Este será el vértice contra el que se acumulará el segundo término de la desigualdad. Dado que se ordenan todas las distancias, la complejidad temporal es de $\mathcal{O}(n^2 log(n))$.

- **Desigualdades tipo 6**

$$d_{ij} + \sum_{\substack{1 \leq k \leq N \\ i \neq k, j \neq k}} (\pm x_{ki} \mp x_{kj}) \geq 1 \qquad 1 \leq i < j \leq N \tag{5.8}$$

El algoritmo de separación intentará encontrar por cada par de vértices i y j la combinación que más violada dejará a la desigualdad. Dado que por cada vértice k distinto a i y j se suma un término, se desea que este sea lo menor posible para violar la desigualdad. Entonces se recorren todos los vértices k y se acumula el menor entre $x_{ki} - x_{kj}$ y $x_{kj} - x_{ki}$. Si al resultado acumulado se le suma d_{ij} y el resultado es menor a 1, entonces la desigualdad está siendo violada y puede ser ingresada como corte. Si existe alguna desigualdad de esta familia violada por la relajación lineal, este algoritmo la puede identificar e ingresar como corte. La complejidad temporal del algoritmo de separación es de $\mathcal{O}(n^3)$.

- **Desigualdades tipo 7**

$$d_{ij} - d_{ik} - d_{jk} \pm 2x_{ik} \mp 2x_{kj} \leq 0 \qquad \begin{matrix} 1 \leq i < j \leq N, 1 \leq k \leq N \\ k \neq i, k \neq j \end{matrix} \tag{5.9}$$

Esta familia de desigualdades posee una cantidad cúbica de desigualdades con respecto a la cantidad de vértices. El procedimiento para identificar aquellas que son violadas consiste en recorrer todas las combinaciones de vértices i, j y k y verificar su validez para la relajación lineal.

- **Desigualdades tipo 8**

$$d_{ij} - d_{ik} - d_{jk} + 2x_{ik} + 2x_{kj} - 4x_{ij} \leq 0 \qquad \begin{matrix} 1 \leq i, j, k \leq N \\ i \neq j, i \neq k, j \neq k \end{matrix} \tag{5.10}$$

Al igual que al resto de las familias con cantidad de desigualdades cúbicas, éstas se resuelven evaluando todas ellas.

6

Experimentación Computacional

En el capítulo anterior se describieron distintas características del algoritmo *Branch-and-Cut*. El objetivo de esta sección es presentar los experimentos computacionales que permiten evaluar la performance del algoritmo desarrollado.

En la primera parte del capítulo se explica de qué manera fueron realizados los experimentos, luego se evalúan las características principales del algoritmo y finalmente se compara el algoritmo desarrollado contra un algoritmo de *Branch-and-Cut* general.

6.1. Implementación del algoritmo y entorno de experimentación

La implementación del algoritmo *Branch-and-Cut* se hizo en *C++* utilizando el paquete CPLEX 12.4[1].

Las instancias de prueba se corrieron sobre una computadora con un procesador i7 de 3.4 GHz y 16Gb de memoria ram. Para el algoritmo de *Branch-and-Cut* implementado se utilizó solo un núcleo para correr las pruebas por limitaciones de CPLEX.

Las instancias de prueba fueron generadas al azar de la misma manera que las pruebas realizadas para seleccionar el modelo.

6.2. Evaluación de heurísticas

Para relizar la evaluación de heurísticas se ejecutó el programa sin la incorporación de planos de corte con todas las combinaciónes entre usar heurística inicial o no y usar heurística primal o no.

A continuación se muestran los resultados obtenidos para algunas instancias de tamaños 14 y 15. Se detalla el valor óptimo de la función objetivo, el tiempo de ejecución en segundos, la cantidad de nodos del árbol explorados, si se utilizaron heurísticas y en el caso de las heurísticas iniciales el valor de la solución inicial.

[1] `http://www-01.ibm.com/software/commerce/optimization/linear-programming/`

N	I	Óptimo	Tiempo	# Nodos	H.P	H.I
14	1	5959	279.64	99455	No	-
14	1	5959	315.12	110168	No	5980
14	1	5959	327.87	110550	Si	-
14	1	5959	317.71	107093	Si	5980
14	2	5915	79.25	25651	No	-
14	2	5915	76.1	27441	No	5915
14	2	5915	77.53	26617	Si	-
14	2	5915	84.06	27368	Si	5915
14	3	6271	225.59	76210	No	-
14	3	6271	217.87	76961	No	6279
14	3	6271	239.16	84454	Si	-
14	3	6271	237.23	84014	Si	6279
14	4	7030	333.12	150619	No	-
14	4	7030	375.78	160664	No	7034
14	4	7030	350.73	153606	Si	-
14	4	7030	351.78	149209	Si	7034
14	5	5952	207.17	86688	No	-
14	5	5952	228.95	92366	No	5952
14	5	5952	237.6	91690	Si	-
14	5	5952	224.88	93845	Si	5952
14	6	6138	153.07	51131	No	-
14	6	6138	144.4	48156	No	6138
14	6	6138	155.37	50427	Si	-
14	6	6138	146.88	49941	Si	6138
14	7	6202	122.51	41970	No	-
14	7	6202	109.97	37179	No	6202
14	7	6202	105.36	33991	Si	-
14	7	6202	105.26	33991	Si	6202
14	8	6166	96.05	34014	No	-
14	8	6166	89.09	30664	No	6166
14	8	6166	105.18	35845	Si	-
14	8	6166	93.9	31751	Si	6166

N	I	Óptimo	Tiempo	# Nodos	H.P	H.I
15	1	8395	535.14	122746	No	-
15	1	8395	573.24	136400	No	8434
15	1	8395	507.37	123385	Si	-
15	1	8395	541.21	133049	Si	8434
15	2	8418	792.12	185309	No	-
15	2	8418	751.12	186617	No	8418
15	2	8418	712.4	169655	Si	-
15	2	8418	690.5	166033	Si	8418
15	3	7348	1367.91	374950	No	-
15	3	7348	1509.94	400448	No	7348
15	3	7348	1158.71	302717	Si	-
15	3	7348	1392.15	358965	Si	7348
15	4	7639	1348.72	370675	No	-
15	4	7639	1242.85	351416	No	7639
15	4	7639	1296.07	351913	Si	-
15	4	7639	1240.33	340769	Si	7639
15	5	7380	932.51	272800	No	-
15	5	7380	854.02	258870	No	7380
15	5	7380	909.58	289726	Si	-
15	5	7380	901.66	272427	Si	7380
15	6	8228	1375.74	356789	No	-
15	6	8228	1313.13	347369	No	8254
15	6	8228	1362.82	358030	Si	-
15	6	8228	1560.6	373711	Si	8254
15	7	7828	635.53	169021	No	-
15	7	7828	762.72	185211	No	7832
15	7	7828	839.49	204367	Si	-
15	7	7828	828.73	203795	Si	7832
15	8	7755	483.07	110477	No	-
15	8	7755	473.88	105836	No	7759
15	8	7755	541.69	117428	Si	-
15	8	7755	512.61	109565	Si	7759

Una primera observación es que el valor obtenido por la heurística inicial es muy cercano al valor óptimo para muchas instancias. También se puede ver que existe mucha varianza en el tiempo de ejecución entre las distintas instancias, pero dentro de las pruebas realizadas para cada instancia no se aprecian diferencias muy signficativas.

Tampoco existe una clara opción que resulte mejor que el resto. Esto se puede deber a que en el *POLP* es muy fácil obtener una solución factible, ya que cualquier orden total de vértices es factible. Esto implica que el algoritmo sin heurísticas logre encontrar alguna solución razonable en poco tiempo.

6.3. Evaluación de cortes

Para realizar la evaluación de los cortes primero se observó la cota inferior alcanzada en el nodo raíz del árbol *Branch-and-Bound* y cuántos planos de corte fueron insertados al modelo por cada familia de desigualdades. Dependiendo del valor de la cota inferior y conociendo el valor del óptimo en cada instancia se puede calcular el gap, que permite realizar una comparación entre los distintos cortes.

Para aquellas desigualdades que no se cumplen se define el *excedente* como la diferencia entre el valor del lado izquierdo y la cota. En algunos casos, en vez de incluir todas las desigualdades violadas, sólo se agregan como planos de cortes, aquellas que tengan un excedente mayor a cierto margen de tolerancia. Se encontró que el valor adecuado de este margen de tolerancia depende de cada familia de desigualdades y se realizaron diversas pruebas buscando los que obtienen mejores resultados.

Otro factor importante es la similitud entre desigualdades pertencientes a una misma familia. Si los planos de corte son muy similares entre sí, puede resultar poco efectivo agregar todas las desigualdades violadas, por lo que en algunos casos se busca una forma de evitar agregar desigualdades similares.

Para determinar los factores anteriores se realizó una gran cantidad de experimentos preliminares cuyos resultados no son presentados ya que no aportan conclusiones significativas. A continuación se explican algunas determinaciones realizadas y se presentan los resultados de los cortes ingresados para el nodo raíz en algunas instancias.

6.3.1. Sin cortes

Primero se evaluó el valor del gap al salir del nodo raíz sin la incorporación de planos de corte. Las desigualdades anti-ciclo son necesarias para la correctitud del modelo, por lo que siempre se agregan dichos planos de corte al modelo. En la tabla a continuación se muestra para instancias de 20 y 21 vértices el valor de la cota inferior y cuán alejada se encuentra con respecto al óptimo de la instancia (gap). También se muestra la cantidad de cortes anti-ciclos que fueron agregados.

N	I	x^*	Gap	#AC
20	1	10314.00	65.44	424
20	2	12285.00	56.24	411
20	3	11613.00	58.5	428
20	4	11167.00	67.6	413
20	5	10828.00	64.34	438
20	6	11551.00	61.91	426
20	7	11177.00	61.09	441
20	8	10752.00	66.33	402
Promedio			62.56	422.87
21	1	13415.00	67.88	488
21	2	12754.00	67.79	505
21	3	12245.00	63.97	471
21	4	13571.00	58.43	490
21	5	12271.00	68.64	493
21	6	12098.00	73.84	479
21	7	12830.00	62.46	463
21	8	12501.00	70.17	461
Promedio			66.65	481.25

Tab. 6.1: Resultados en nodo raíz sin utilizar planos de cortes

6.3.2. Desigualdades de tipo 2

Mediante experimentos preliminares se determinó que el algoritmo de separación incluya como planos de corte todas aquellas desigualdades que tengan un excedente mayor a 1. Siempre que se encuentre una de estas desigualdades violada en la relajación lineal será agregado un plano de corte.

N	I	x^*	Gap	#Cortes	#AC	GapOrig
20	1	10314.00	65.44	9	464	65.44
20	2	12285.00	56.24	10	458	56.24
20	3	11622.00	58.38	17	835	58.5
20	4	11167.00	67.6	1	414	67.6
20	5	10835.81	64.22	9	847	64.34
20	6	11551.00	61.91	3	426	61.91
20	7	11177.00	61.09	0	441	61.09
20	8	10752.00	66.33	1	402	66.33
Promedio			62.56	6.25	535.88	62.56
21	1	13415.00	67.88	0	488	67.88
21	2	12754.00	67.79	3	537	67.79
21	3	12341.00	62.7	6	528	63.97
21	4	13571.00	58.43	3	489	58.43
21	5	12271.00	68.64	14	527	68.64
21	6	12098.00	73.84	2	466	73.84
21	7	12830.00	62.46	16	508	62.46
21	8	12501.00	70.17	7	492	70.17
Promedio			66.65	6.38	504.38	66.65

Tab. 6.2: Cortes agregados en el nodo raíz para desigualdades de tipo 2

Los resultados indican que esta desigualdad no aporta mejoras a los resultados ya que muestran las mismas cotas obtenidas sin usar planos de cortes.

6.3.3. Desigualdades de tipo 4

Las desigualdades del tipo 4 presentan una mejora con respecto a no usar cortes, reduciendo el gap en un 37 % en promedio. El método en el que se agregan las desigualdades de esta familia como planos de corte es igual al caso anterior.

N	I	x^*	Gap	#Cortes	#AC	GapOrig
20	1	12315.00	38.56	154	525	65.44
20	2	14221.50	34.97	185	504	56.24
20	3	13796.50	33.41	146	526	58.5
20	4	13340.03	40.3	159	496	67.6
20	5	12661.33	40.54	103	520	64.34
20	6	13247.50	41.18	134	484	61.91
20	7	12985.77	38.65	155	496	61.09
20	8	12467.17	43.44	165	508	66.33
Promedio			38.88	150.13	507.38	62.56
21	1	16364.67	37.62	184	639	67.88
21	2	15858.50	34.94	190	569	67.79
21	3	14038.50	43.02	175	543	63.97
21	4	16016.89	34.23	206	569	58.43
21	5	14227.25	45.45	170	554	68.64
21	6	15139.29	38.92	209	579	73.84
21	7	14944.74	39.47	178	545	62.46
21	8	14747.50	44.25	155	562	70.17
Promedio			39.74	183.38	570.00	66.65

Tab. 6.3: Cortes agregados en el nodo raíz para desigualdades de tipo 4

6.3.4. Desigualdades de tipo 5

Las desigualdades de tipo 5 reducen aún más el gap con respecto a no utilizar ninguna otra desigualdad. También se agregan todas las desigualdades que se encuentran tal que la diferencia absoluta entre pares de vértices sea menor a 0,0001.

N	I	x^*	Gap	#Cortes	#AC	GapOrig
20	1	13462.70	26.75	108	764	65.44
20	2	15467.16	24.1	154	772	56.24
20	3	14408.76	27.74	80	714	58.5
20	4	15020.44	24.61	115	772	67.6
20	5	14368.61	23.84	162	855	64.34
20	6	14955.85	25.05	151	783	61.91
20	7	14033.20	28.31	121	663	61.09
20	8	14220.37	25.76	144	814	66.33
Promedio			25.77	128.38	767.13	62.56
21	1	17959.84	25.4	274	949	67.88
21	2	16922.80	26.46	288	941	67.79
21	3	15434.88	30.08	300	885	63.97
21	4	17202.85	24.98	266	1008	58.43
21	5	16232.67	27.48	334	1014	68.64
21	6	16614.86	26.58	358	925	73.84
21	7	16082.10	29.6	240	840	62.46
21	8	17233.41	23.44	292	967	70.17
Promedio			26.75	294.00	941.13	66.65

Tab. 6.4: Cortes agregados en el nodo raíz para desigualdades de tipo 5

6.3.5. Desigualdades de tipo 6

Para las desigualdades de tipo 6 se definió un margen de tolerancia de 0.5. Todas aquellas que se excedan son agregadas como planos de corte.

N	I	x^*	Gap	#Cortes	#AC	GapOrig
20	1	10314.00	65.44	83	629	65.44
20	2	12285.00	56.24	95	711	56.24
20	3	11613.00	58.5	97	657	58.5
20	4	11167.00	67.6	95	625	67.6
20	5	10828.00	64.34	99	652	64.34
20	6	11551.00	61.91	99	672	61.91
20	7	11177.00	61.09	105	677	61.09
20	8	10752.00	66.33	102	661	66.33
Promedio			62.68	96.88	660.5	62.56
21	1	13415.00	67.88	118	703	67.88
21	2	12754.00	67.79	107	679	67.79
21	3	12245.00	63.97	118	664	63.97
21	4	13571.00	58.43	99	730	58.43
21	5	12271.00	68.64	107	755	68.64
21	6	12098.00	73.84	120	698	73.84
21	7	12830.00	62.46	108	620	62.46
21	8	12501.00	70.17	119	671	70.17
Promedio			66.65	112.00	690.00	66.65

Tab. 6.5: Cortes agregados en el nodo raíz para desigualdades de tipo 6

Al igual que las desigualdades de tipo 2, las cotas inferiores obtenidas no muestran que dichos cortes aporten para resolver el problema.

6.3.6. Desigualdades de tipo 7

Estas desigualdades son una de las que mejores resultados obtuvieron en el nodo raíz. La cota inferior logra alcanzar valores cercanos al óptimo lo cual será de gran ayuda para demostrar la optimalidad de la solución encontrada.

Debido a la gran cantidad de cortes de este tipo que se incorporan en el nodo raíz se decidió diferenciar cuáles son las desigualdades utilizadas para los planos de corte entre el nodo raíz y el resto de los nodos. Luego de varias pruebas, resultó mejor utilizar distintos valores de margen de tolerancia en distintos nodos del árbol de búsqueda. Se seleccionaron valores de manera tal que en el nodo raíz se incluyan aquellas que tengan un excedente mucho mayor que en el resto de los nodos.

Por otro lado, debido a que aún existía una gran cantidad de desigualdades que no son válidas en la relajación lineal asociada a cada nodo, también se decidió agregar sólo algunas de ellas como planos de corte. Dado que varias de ellas resultan ser similares por compartir las mismas variables, se agregan aquellas que resultaran diferentes entre sí. Como este tipo de desigualdades utiliza el hecho que existen dos vértices i y j con un vértice k entre ellos se agregaron todas las desigualdades con i y j distintos.

N	I	x^*	Gap	#Cortes	#AC	GapOrig
20	1	16247.86	5.02	478	1306	65.44
20	2	18424.95	4.17	492	1269	56.24
20	3	17300.47	6.39	397	1102	58.5
20	4	17916.93	4.46	481	1278	67.6
20	5	16857.46	5.56	447	1172	64.34
20	6	17902.05	4.47	498	1345	61.91
20	7	17376.04	3.62	520	1355	61.09
20	8	16774.21	6.61	415	1212	66.33
Promedio			5.04	466.00	1254.88	62.56
21	1	21353.91	5.47	518	1450	67.88
21	2	20434.69	4.72	565	1436	67.79
21	3	18774.56	6.94	541	1391	63.97
21	4	20707.08	3.83	581	1534	58.43
21	5	19766.34	4.69	541	1505	68.64
21	6	20140.86	4.42	540	1437	73.84
21	7	19954.69	4.45	542	1467	62.46
21	8	20613.93	3.2	551	1552	70.17
Promedio			4.72	547.38	1471.5	66.65

Tab. 6.6: Cortes agregados en el nodo raíz para desigualdades de tipo 7

6.3.7. Desigualdades de tipo 8

Las desigualdades de tipo 8 obtuvieron resultados similares a las desigualdades de tipo 7 y surgieron las mismas dificultades, por lo tanto, fueron incluidas de la misma forma que las anteriores.

N	I	x^*	Gap	#Cortes	#AC	GapOrig
20	1	16295.52	4.71	707	956	65.44
20	2	18456.68	4.0	706	1034	56.24
20	3	17370.64	5.96	632	971	58.5
20	4	17967.41	4.17	664	1040	67.6
20	5	16821.76	5.78	660	905	64.34
20	6	17971.88	4.06	735	1051	61.91
20	7	17445.02	3.21	763	1140	61.09
20	8	16855.05	6.1	644	931	66.33
Promedio			4.75	688.88	1003.5	62.56
21	1	21390.62	5.28	761	1270	67.88
21	2	20488.99	4.45	835	1214	67.79
21	3	18789.72	6.86	772	1176	63.97
21	4	20706.93	3.83	840	1232	58.43
21	5	19832.60	4.34	835	1180	68.64
21	6	20226.13	3.98	840	1131	73.84
21	7	20011.43	4.16	775	1223	62.46
21	8	20680.63	2.86	811	1194	70.17
Promedio			4.47	808.63	1202.5	66.65

Tab. 6.7: Cortes agregados en el nodo raíz para desigualdades de tipo 8

6.3.8. Todas las desigualdades válidas

En esta sección se presenta cómo resulta la cota inferior al salir del nodo raíz utilizando todas las desigualdades válidas como planos de corte. Se puede ver que se consiguen cotas que mejoran todas las demás obtenidas. También se muestra la cantidad de planos de corte incorporados de cada familia.

N	I	x^*	Gap	#AC	D2	D4	D5	D6	D7	D8	GapOrig
20	1	16495.28	3.44	972	12	149	7	90	320	627	65.44
20	2	18615.27	3.11	958	23	144	7	87	343	669	56.24
20	3	17861.30	3.05	1020	17	130	6	93	296	578	58.5
20	4	18024.20	3.84	918	20	125	6	93	338	655	67.6
20	5	17193.33	3.49	913	18	110	11	95	341	645	64.34
20	6	17988.71	3.97	971	15	121	7	89	370	701	61.91
20	7	17549.15	2.6	1022	26	133	9	85	358	679	61.09
20	8	17386.83	2.85	965	21	143	8	94	311	598	66.33
Promedio			3.29	967.38	19.00	131.88	7.63	90.75	334.63	644.00	62.56
21	1	21776.23	3.42	1079	21	150	14	110	379	756	67.88
21	2	20771.44	3.03	1067	32	154	12	98	394	772	67.79
21	3	19353.92	3.74	1086	24	174	12	108	372	736	63.97
21	4	20886.05	2.94	1173	32	156	14	114	381	730	58.43
21	5	19990.54	3.51	1150	14	157	14	97	385	772	68.64
21	6	20462.94	2.78	1084	24	158	17	110	394	753	73.84
21	7	20098.96	3.7	1042	16	164	14	119	361	689	62.46
21	8	20789.66	2.32	1106	13	141	27	109	383	757	70.17
Promedio			3.18	1098.38	22.00	156.75	15.5	108.13	381.13	745.63	66.65

Tab. 6.8: Cortes agregados en el nodo raíz utilizando todos los cortes

Por último se realizó una evaluación de aquellas familias que no mostraron mejorar la cota inferior en el nodo raíz. A continuación se muestran los tiempos de ejecución para resolver todas las instancias, utilizando todos los cortes, ignorando las desigualdades del tipo 2, del tipo 6 y ambas. Se indica con un asterisco la mejor alternativa.

N	I	Con	Sin 2	Sin 6	Sin 2 ni 6
20	1	120.35*	131.09	187.45	128.27
20	2	172.03*	278.71	249.64	248.03
20	3	52.49*	57.98	65.17	63.13
20	4	259.92	292.67	231.5*	327.18
20	5	35.28	34.51	44.67	31.28*
20	6	447.71*	662.52	660.59	610.60
20	7	75.89*	88.66	90.42	90.88
20	8	38.08	31.04*	33.6	31.84
21	1	102.35	96.23	85.28*	99.92
21	2	180.4	128.23	304.57	102.72*
21	3	280.94	173.39*	304.31	379.86
21	4	210.6	227.68	137.21*	144.68
21	5	506.83*	515.54	650.79	638.62
21	6	110.12	86.13*	120.29	88.56
21	7	156.0*	178.27	237.73	316.45
21	8	40.72*	80.28	51.75	105.45
Promedio		174.36	191.43	215.94	212.98

Tab. 6.9: Comparación de tiempos de ejecución ignorando ciertas familias de desigualdades

Si bien no existe una alternativa que domine a las demás, se puede ver que utilizar todos los cortes tiene mejor tiempo en mayor cantidad de casos. La última fila, que muestra los promedios de tiempos de ejecución sobre todas las instancias, también indica que utilizar ambos cortes resulta en un mejor desempeño del algoritmo.

Basados en estos resultados, la estrategia elegida para la incorporación de planos de corte para continuar con la experimentación es utilizar todos los planos de corte.

6.4. Evaluación de variables de branching

La evaluación de variable de branching se realizó utilizando heurística inicial e incorporando planos de corte. Se realizaron pruebas para instancias con 18 y 19 vértices con un límite de tiempo de 30 minutos. Se evaluaron las tres opciones descriptas en la Sección 5.1.1. A continuación se pueden ver los tiempos de ejecución y cantidad de nodos explorados para cada uno de los casos. Se indica con asterisco el caso con mejor desepeño.

		Distancia		Precedencia		CPLEX	
N	I	Tiempo	#Nodos	Tiempo	#Nodos	Tiempo	#Nodos
18	1	1800.00	45078	20.07*	148	25.89	188
18	2	1800.00	46831	6.17*	37	7.03	31
18	3	1800.00	33895	45.28*	460	49.81	512
18	4	1800.01	28003	10.57*	61	12.12	67
18	5	1800.00	27542	69.10*	519	78.11	672
18	6	1800.00	42459	36.12*	301	37.41	307
18	7	1800.01	40361	144.83	1545	99.70*	993
18	8	1800.00	43246	36.19*	360	38.59	380
19	1	1800.00	14470	22.75	138	19.41*	99
19	2	1800.01	14196	27.87	217	26.11*	185
19	3	1800.00	46317	10.60*	45	11.98	46
19	4	1800.00	26380	35.52*	233	36.42	231
19	5	1800.00	25358	69.93	529	48.33*	300
19	6	1800.00	25060	109.03*	755	122.58	868
19	7	1800.00	36052	22.94*	144	29.34	235
19	8	1800.00	45751	44.33*	288	48.44	287
Promedio		1800.00	33812.44	44.46	361.25	43.20	337.56

Tab. 6.10: Opciones de *branching* priorizando distintos tipos de variables

De los resultados obtenidos se observa claramente el pobre desempeño que obtuvo priorizar las variables de distancia.

En los demás casos los resultados son comparables sin diferencia marcada. Se decidió utilizar la prioridad por variables de precedencia debido a que existe una mayor cantidad de casos en los que obtuvo resultado favorable.

6.5. Reevaluación de heurísticas

Debido a que las heurísticas iniciales no habían mostrado diferencias significativas antes de ingresar los planos de corte, se volvieron a realizar experimentos comparando la performance de éstas después de haber decidido otros factores, consiguiendo resolver instancias de mayor tamaño.

De esta manera, fue posible realizar la evaluación sobre instancias más grandes, en donde se espera que sea más difícil mediante heurísticas encontrar la solución óptima.

A continuación se muestran los tiempos de ejecución en instancias de mayor tamaño, el valor de la solución y de la heurística inicial. Se indica con asterisco aquellas que obtuvieron mejor tiempo.

N	I	Óptimo	Tiempo	# N	H.P	H.I
20	1	17064	147.31	703	No	-
20	1	17064	120.35	741	No	17070
20	1	17064	118.79	612	Si	17070
20	2	19195	366.57	1584	No	-
20	2	19195	172.03	1013	No	19211
20	2	19195	321.16	1568	Si	19211
20	3	18407	74.77	301	No	-
20	3	18407	52.49	229	No	18469
20	3	18407	66.76	285	Si	18469
20	4	18717	323.95	1408	No	-
20	4	18717	259.92	1273	No	18745
20	4	18717	367.79	1670	Si	18745
20	5	17795	42.78	137	No	-
20	5	17795	35.28	153	No	17827
20	5	17795	38.0	133	Si	17827
20	6	18703	1037.63	6211	No	-
20	6	18703	447.71	2457	No	18737
20	6	18703	1285.85	7125	Si	18737
20	7	18006	93.99	608	No	-
20	7	18006	75.89	462	No	18011
20	7	18006	78.37	529	Si	18011
20	8	17884	37.82	181	No	-
20	8	17884	38.09	191	No	17884
20	8	17884	39.39	201	Si	17884

N	I	Óptimo	Tiempo	# N	H.P	H.I
21	1	22522	148.38	621	No	-
21	1	22522	102.35	473	No	22522
21	1	22522	121.4	521	Si	22522
21	2	21401	170.61	759	No	-
21	2	21401	180.4	757	No	21436
21	2	21401	152.01	719	Si	21436
21	3	20079	241.05	979	No	-
21	3	20079	280.94	1163	No	20156
21	3	20079	230.1	989	Si	20156
21	4	21501	233.09	778	No	-
21	4	21501	210.6	730	No	21501
21	4	21501	216.43	830	Si	21501
21	5	20694	601.9	2368	No	-
21	5	20694	506.83	2501	No	20742
21	5	20694	692.5	2909	Si	20742
21	6	21032	118.22	439	No	-
21	6	21032	110.12	377	No	21049
21	6	21032	91.09	350	Si	21049
21	7	20844	134.46	603	No	-
21	7	20844	156.0	817	No	20852
21	7	20844	145.75	632	Si	20852
21	8	21274	86.06	374	No	-
21	8	21274	40.72	177	No	21282
21	8	21274	67.45	313	Si	21282

Si bien sigue existiendo mucha varianza entre los tiempos de las distintas instancias, se puede ver que utilizar heurísticas en general ayuda a resolver en menos tiempo las instancias, en particular utilizar sólo la inicial, que obtuvo mejores resultados en más de la mitad de los casos. Mientras que no utilizar heurísticas tiene un tiempo promedio de aproximadamente 241 segundos sobre todas las instancias, y utilizar ambas 252 segundos, utilizar sólo la heurística inicial tiene un promedio de 174 segundos, aproximadamente un 30 % menos.

6.6. Experimentación contra CPLEX

Por último se realizó la comparación entre el algoritmo propuesto y el *Branch-and-Cut* que provee CPLEX por defecto.

Se realizaron pruebas para instancias de diversos tamaños fijando un límite en el tiempo de ejecución de 30 minutos. Al comparar con CPLEX por defecto se probaron 4 alternativas que podrían obtener mejores que el algoritmo desarrollado.

1. Utilizar sólo las desigualdades válidas de anti-ciclo, para agregar restricciones cúbicas al modelo. Al agregar planos de cortes, CPLEX no permite utilizar multiples núcleos, por lo que se ejecuta en un proceso.

2. No utilizar planos de cortes ni heurísticas y no utilizar el estudio poliedral. De esta manera se utilizan los 8 núcleos.

3. No utilizar planos de cortes ni heurísticas y agregar las desigualdades 1,3 y 9 al modelo original (aquellas que tienen cardinalidad menos de $\mathcal{O}(n^3)$). De esta manera se utilizan los 8 núcleos.

4. No utilizar planos de cortes ni heurísticas y agregar las desigualdades 1,3,4,7,8 y 9 al modelo original. De esta manera se utilizan los 8 núcleos.

Los resultados obtenidos muestran que el algoritmo de *Branch-and-Cut* implementado obtiene mejores resultados que las demás alternativas. Además se puede ver la gran ayuda que proporciona el estudio poliedral al incluir las restricciones al modelo utilizando el algoritmo general de *Branch-and-Cut* de CPLEX, que si bien para los casos pequeños este provee resultados decentes, para instancias de mayor tamaño existe una diferencia muy notoria.

Las siguientes tablas muestran la performance para cada una de las alternativas, indicando la cantidad de nodos explorados y el tiempo en obtener la solución óptima. En caso de que el algoritmo no termine al cabo de los 30 minutos, se indica entre paréntesis el gap obtenido al llegar al límite de tiempo.

		B&C		CPLEX 1		CPLEX 2		CPLEX 3		CPLEX 4	
N	I	t(s)	#N	t(s)	#N	t(s)	#N	t(s)	#N	t(s)	#N
10	1	0.20	7	4.85	23808	3.26	43072	1.60	3328	1.13	458
10	2	0.16	3	1.03	4432	0.45	5043	0.58	1902	1.19	158
10	3	0.15	3	1.22	5436	1.14	13144	0.51	1588	1.43	333
10	4	0.19	11	1.07	4200	0.67	6832	0.39	1260	0.70	1
10	5	0.23	12	2.19	10304	1.77	17215	1.14	4618	1.03	113
10	6	0.18	6	2.80	13271	1.81	27587	0.51	2226	0.66	41
10	7	0.22	8	1.50	6639	2.02	19333	0.31	1246	1.39	474
10	8	0.21	17	1.67	7857	0.45	5102	0.62	1862	0.76	116
11	1	0.35	10	14.95	58022	6.10	80090	2.63	6049	1.24	180
11	2	0.24	3	6.37	22082	2.43	23246	1.73	3707	1.54	145
11	3	0.34	8	9.41	29969	2.97	29268	0.90	2219	1.28	63
11	4	0.30	3	12.37	41635	5.11	50091	1.65	5188	1.71	269
11	5	0.36	10	10.49	35077	3.64	36151	2.53	7414	1.66	116
11	6	0.42	11	13.03	53563	6.94	75417	2.47	7514	1.58	308
11	7	0.30	3	4.94	16857	5.46	61933	2.17	5095	0.90	69
11	8	0.32	11	15.50	58824	5.68	64296	5.39	9835	1.58	242
12	1	0.71	23	142.18	424560	20.57	221558	9.73	18726	7.80	1067
12	2	0.48	4	32.68	78646	9.84	79017	3.40	6465	3.85	294
12	3	0.70	25	15.16	36444	10.86	93201	11.40	22628	3.49	362
12	4	1.12	65	23.67	68890	29.08	325443	8.65	13771	5.60	752
12	5	0.63	19	88.13	262728	18.30	215010	8.08	10878	5.77	884
12	6	0.87	33	21.91	65385	14.25	132962	10.46	16227	3.04	330
12	7	0.66	17	127.64	369325	33.31	368057	6.57	8783	5.77	1259
12	8	1.05	45	33.80	99137	24.65	247296	9.10	18818	6.06	892
13	1	1.24	25	232.37	531242	88.51	788775	21.11	28426	7.82	504
13	2	2.09	68	187.38	388781	90.32	703971	36.10	63595	8.51	1461
13	3	1.62	37	165.51	359419	31.60	239685	26.63	41279	10.88	927
13	4	1.45	39	357.73	862353	89.33	760541	26.67	34269	7.43	847
13	5	1.12	25	149.00	332189	39.87	285166	19.50	25166	8.24	548
13	6	1.01	11	263.63	581205	77.23	634558	30.95	42288	4.85	271
13	7	2.05	57	455.33	1116316	98.57	775121	23.17	31365	14.16	1415
13	8	1.77	45	362.14	864752	90.19	727250	19.28	23052	12.47	1528

		B&C		CPLEX 1		CPLEX 2		CPLEX 3		CPLEX 4	
N	I	t(s)	#N	t(s)	#N	t(s)	#N	t(s)	#N	t(s)	#N
20	1	120.35	741	(45.21)	441683	(40.00)	1430370	(26.33)	417031	1760.67	6956
20	2	172.03	1013	(45.57)	492616	(32.05)	1474731	(21.75)	385205	(0.44)	6208
20	3	52.49	229	(42.82)	449582	(37.49)	1392340	(21.59)	311974	829.20	3563
20	4	259.92	1273	(49.89)	442102	(30.49)	1522300	(27.26)	373163	(0.72)	7161
20	5	35.28	153	(53.76)	463804	(38.83)	1512948	(22.76)	335294	597.34	3572
20	6	447.71	2457	(45.52)	499589	(33.76)	1539612	(27.12)	378519	(0.96)	6006
20	7	75.89	462	(42.13)	465725	(41.98)	1453232	(24.78)	403505	984.07	5760
20	8	38.09	191	(50.79)	426659	(43.42)	1396571	(26.56)	371949	279.52	1415
21	1	102.35	473	(60.60)	369389	(44.04)	1190950	(33.14)	231357	(0.66)	5702
21	2	180.40	757	(60.75)	347178	(43.27)	1210114	(29.26)	203943	896.47	3642
21	3	280.94	1163	(52.59)	376518	(45.13)	1178573	(33.23)	206710	(2.17)	4509
21	4	210.60	730	(50.61)	348253	(37.85)	1212030	(29.10)	214532	(0.86)	4293
21	5	506.83	2501	(54.61)	392100	(52.99)	1163634	(34.75)	202921	1464.95	6025
21	6	110.12	377	(61.88)	378885	(42.57)	1134061	(32.92)	258782	(0.47)	6020
21	7	156.00	817	(55.46)	381594	(40.61)	1108704	(31.25)	201827	(0.10)	7057
21	8	40.72	177	(62.49)	370978	(54.06)	1101353	(31.27)	220891	978.81	3895
22	1	869.46	1702	(60.40)	297211	(41.36)	948622	(35.01)	140548	(1.23)	3553
22	2	(0.56)	2972	(51.09)	313922	(42.06)	898312	(32.37)	132890	(2.69)	3406
22	3	170.35	528	(63.68)	312262	(59.03)	883561	(41.92)	175334	(0.16)	5029
22	4	103.02	394	(53.11)	313216	(42.12)	898240	(34.27)	170940	956.14	3207
22	5	494.10	1798	(58.25)	277801	(49.69)	825029	(43.05)	181100	(0.30)	4989
22	6	90.26	290	(56.01)	305352	(43.82)	868991	(37.32)	145301	(0.16)	4215
22	7	141.63	473	(55.25)	311221	(48.00)	870271	(36.89)	163601	(0.40)	3911
22	8	1024.39	2954	(64.13)	271851	(51.00)	847297	(40.79)	153968	(2.26)	3323
23	1	848.87	2089	(66.61)	259952	(61.28)	615439	(51.37)	34766	(1.62)	2180
23	2	691.45	1923	(69.28)	245923	(59.21)	687503	(48.96)	57178	(1.09)	2743
23	3	1762.71	3883	(69.93)	235136	(64.72)	703060	(49.22)	88213	(1.52)	2321
23	4	113.51	230	(60.72)	213722	(50.74)	659623	(41.37)	98834	(0.85)	2210
23	5	(0.38)	3345	(70.23)	243691	(53.62)	658831	(45.89)	78316	(2.74)	2228
23	6	516.10	1692	(60.96)	245655	(47.69)	669661	(43.67)	49555	(1.75)	2487
23	7	222.61	611	(58.61)	234203	(46.50)	656583	(43.22)	64764	(0.90)	2404
23	8	309.15	868	(65.38)	250536	(54.90)	658439	(44.85)	91839	(0.84)	2591
24	1	(1.69)	2668	(74.45)	182142	(66.38)	502006	(51.74)	61012	(2.52)	1564
24	2	(2.50)	2306	(77.67)	205626	(66.23)	510225	(55.97)	48941	(4.18)	1372
24	3	(1.93)	1873	(72.63)	200152	(62.28)	540402	(48.75)	42232	(5.23)	996
24	4	(0.84)	2490	(71.20)	190330	(57.36)	538408	(52.78)	39559	(2.88)	1405
24	5	89.61	187	(65.54)	176855	(55.16)	516416	(46.45)	50614	(2.24)	1453
24	6	82.68	181	(73.90)	163608	(56.95)	501209	(50.95)	67536	1106.93	1836
24	7	(1.99)	2160	(60.56)	180903	(53.58)	502551	(44.80)	74987	(4.37)	1545
24	8	(0.32)	3019	(60.98)	195237	(56.66)	500411	(42.55)	54375	(4.51)	1274

Se puede ver que los resultados obtenidos con el algoritmo *Branch-and-Cut* desarrollado son muy positivos. Para las instancias de tamaño mediano se logró resolver todas ellas en una cantidad de tiempo órdenes de magnitud menor que el resto, mientras que para las

grandes se resolvieron instancias que las demás opciones no pudieron.

A continuación se comparan los tiempos de ejecución promedio para las instancias de tamaño mediano.

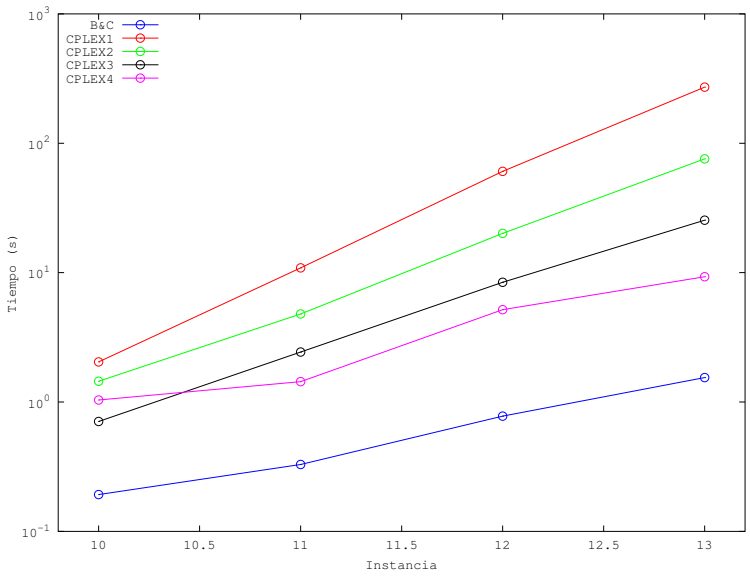

Fig. 6.1: Promedios de tiempo de ejecución para las instancias de tamaño 10, 11, 12 y 13

Bajo las configuraciones 2,3 y 4, CPLEX permite utilizar algunas técnicas y herramientas del paquete que no están disponibles cuando se utiliza la opción de incorporar planos particulares del problema.

Una de las opciones, por ejemplo, es la de permitir ejecutar el algoritmo en multiples núcleos, permitiendo paralelizar el recorrido del árbol. Existen varias ventajas como éstas que se pudieron aprovechar en estos casos.

Aún así el algoritmo desarrollado logra una mejor perfomance en todas las instancias. Sólo la opción 4 logró resolver alguna de las instancias grandes en el tiempo permitido pero en tiempos que difieren en el orden de magnitud.

Por otro lado, también se puede apreciar la ventaja de haber realizado el estudio poliedral. Esto se ve particularmente al comparar las opciones 2, 3 y 4. La opción 2 corresponde al modelo original mientras que en la opción 3 se agregan las desigualdades cuadráticas, ajustando así las relajaciones lineales asociadas. Finalmente, la opción 4 además agrega la familia de desigualdades de orden cúbico. Lo que se observa es que la cantidad de nodos explorados decrece abruptamente cuanto mayor es la cantidad de desigualdades agregadas al modelo.

Sin embargo la cantidad de nodos explorados no garantiza una mejora en el tiempo de ejecución debido a que el costo de resolver cada relajación lineal es mucho mayor. En este caso, se observa que el tiempo de ejecución en aquellas instancias que fueron resueltas también decrece al incorporar mayor cantidad de desigualdades. La siguiente tabla muestra los promedios sobre cada tamaño de la cantidad de nodos y tiempos para estas opciones.

	CPLEX2		CPLEX3		CPLEX4	
N	t(s)	# Nodos	t(s)	# Nodos	t(s)	# Nodos
10	1.45	17166.0	0.71	2253.75	1.04	211.63
11	4.79	52561.5	2.43	5877.625	1.44	174.0
12	20.11	210318.0	8.42	14537.0	5.17	730.0
13	75.70	614383.38	25.43	36180.0	9.29	937.63
20	-	1465263.0	-	372080.0	890.16	5080.13
21	-	1162427.38	-	217620.38	1113.41	5142.88
22	-	880040.38	-	157960.25	956.14	3954.13
23	-	663642.38	-	70433.13	-	2395.5
24	-	513953.5	-	54907.0	1106.93	1430.63

Tab. 6.11: Promedios de tiempos de ejecución y cantidad de nodos explorados por cada tamaño

Esto indica que el estudio poliedral realizado, más allá de los algoritmos de separación utilizados, permitieron la resolución de instancias de mayor tamaño. Es importante notar que en aquellas instancias en las que el algoritmo no logró terminar en el tiempo límite, el gap obtenido en un mismo margen de tiempo es menor.

7

Conclusiones y Trabajo Futuro

A lo largo del trabajo se realizaron distintas observaciones y conclusiones específicas de cada sección. A continuación se enfatizan algunas conclusiones generales sobre el trabajo realizado.

- **Modelo**
  El modelo elegido finalmente resultó estar basado en el problema de ordenamiento lineal. Si bien esto era lo esperado, fue importante analizar los distintos problemas que se pueden relacionar con el problema tratado ya que podía suceder que la calidad del modelo no fuera extrapolable al *POLP* debido a que fue necesario agregar más variables al modelo original.

- **Estudio poliedral**
  Una de las conclusiones más importantes del trabajo es el gran impacto que tuvo el estudio poliedral del modelo. Aún utilizando el *Branch-and-Cut* genérico provisto por CPLEX, agregar las desigualdades válidas al modelo permitió resolver las instancias de prueba con mucho mejores resultados. La utilización de estas desigualdades como planos de corte permitió resolver instancias que al comenzar el trabajo resultaban muy desafiantes. Si bien no se presentan resultados teóricos, como la demostración del sistema minimal o que las desigualdades válidas sean en realidad facetas, no cabe duda de que fueron muy importantes para el desarrollo del algoritmo y esto es un importante trabajo pendiente.
  Otro aspecto importante a destacar fue la interacción con el software PORTA que permitió: encontrar desigualdades que no hubieran sido posible de obtener, reforzar aquellas desigualdades que no proporcionaban buenos resultados inicialmente, y ganar confianza en las caracterizaciones obtenidas.

- **Heurísticas**
  Las heurísticas iniciales proveyeron muy buenas soluciones para el problema, resultando en soluciones muy cercanas al óptimo. Esto permitió que luego de realizar una iteración de planos de corte en el nodo raíz el algoritmo *Branch-and-Cut* comenzará con menos de 5 % de gap en las instancias evaluadas.

Por otro lado, las heurísticas primales no mostraron ser muy efectivas, esto en parte se debe a la buena calidad de las heurísticas iniciales.

- **Inserción de planos de corte**
  Un punto en donde hubo que realizar mucha experimentación fue en la evaluación de las distintas configuraciones de incorporación de los planos de corte en el problema. Determinar cuántas y cuáles de las desigualdades violadas por cada familia de desigualdades deberán ser ingresadas como cortes, son algunos de los factores que pueden ayudar a reducir el tiempo de ejecución del algoritmo y entender mejor la estructura del problema.

- **CPLEX**
  El paquete CPLEX se mostró muy efectivo en todos los análisis realizados, pudiendo resolver sin estudios previos del problema instancias que hubieran sido imposibles de resolver mediante una mera fuerza bruta. Los mecanisimos que provee para ingresar cortes y heurísticas fueron utilizados exitosamente y resultaron claves para explotar la estructura particular del problema. Otra ventaja de CPLEX son las herramientas generales que provee para explorar el espacio de búsqueda, permitiendo de una manera sencilla probar todos los aspectos posibles, como selección de variables y estrategias de recorrido entre otras, resultando en mejores tiempos de ejecución.

Debido a la gran cantidad de decisiones que se tomaron en cada una de las secciones es imposible analizar cada una de las combinaciones de opciones posibles. A continuación se mencionan algunas líneas de trabajo futuro.

- **Demostraciones teóricas**
  En el estudio poliedral realizado se conjeturó que existe sólo una igualdad en el sistema minimal, y se encontró que para varios tamaños de instancias las desigualdades válidas resultan ser facetas. Estos aspectos deben ser demostrados rigurosamente. Conocer estas características permite tener un mejor conocimiento de la cápsula convexa, pudiendo hallar más desigualdades válidas.

- **Desigualdades válidas**
  En todas las instancias se logró conseguir rápidamente una solución muy cercana al óptimo mediante el uso de las heurísticas, y gracias a las desigualdades válidas encontradas se pudo determinar que el valor de la función objetivo en la configuración óptima no se encuentra muy lejano. Sin embargo, para las instancias muy grandes se debe invertir mucho tiempo para demostrar optimalidad. Generando más familias de desigualdades válidas se podría fortalecer el modelo resultando en un espacio de soluciones más acotado y obteniendo relajaciones lineales más ajustadas a la cápsula convexa.

- **Otros modelos**
  La primer decisión tomada en este trabajo fue la elección del modelo a utilizar, en el que se basó el estudio poliedral y el desarrollo del algoritmo propuesto. Como mencionamos anteriormente, los resultados obtenidos en el estudio poliedral fueron muy fructíferos y permitieron resolver las instancias más desafiantes. Esto hace notar que si bien este modelo mostró tener un mejor desempeño en las pruebas en las que se basó su elección, para asegurar que es el modelo más apto para el desarrollo de un algoritmo *Branch-and-Cut*, es necesario un estudio similar al realizado para cada uno de los modelos. Esto puede constituir un trabajo futuro de importante envergadura.

Bibliografía

[1] G. S. Bloom y S. W. Golomb. Applications of numbered undirected graphs. *Proceedings of the IEEE*, 65(4):562–570, 1977.

[2] A. Caprara, A. N. Letchford, y J. J. Salazar-González. Decorous lower bounds for minimum linear arrangement. *INFORMS Journal on Computing*, 23(1):26–40, 2011.

[3] I. Charon y O. Hudry. A branch-and-bound algorithm to solve the linear ordering problem for weighted tournaments. *Discrete Applied Mathematics*, 154(15):2097–2116, 2006.

[4] H. B. Chenery y T. Watanabe. International comparisons of the structure of production. *Econometrica: Journal of the Econometric Society*, pages 487–521, 1958.

[5] F. R. K. Chung. Labelings of graphs. *Selected topics in graph theory*, 3:151–168, 1988.

[6] V. Chvatal. *Linear programming*. Macmillan, 1983.

[7] G. Dantzig, R. Fulkerson, y S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, pages 393–410, 1954.

[8] M. R. Garey y D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[9] F. Glover, T. Klastorin, y D. Kongman. Optimal weighted ancestry relationships. *Management Science*, 20(8):1190–1193, 1974.

[10] R. E. Gomory. Outline of an Algorithm for Integer Solutions to Linear Programs. *Bulletin of the American Society*, 64:275–278, 1958.

[11] M. Grötschel, M. Jünger, y G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations research*, 32(6):1195–1220, 1984.

[12] M. Grötschel, M. Jünger, y G. Reinelt. Acyclic subdigraphs and linear orderings: Polytopes, facets, and a cutting plane algorithm. In Ivan Rival, editor, *Graphs and Order. The Role of Graphs in the Theory of Ordered Sets and its Applications, Proceedings of NATO ASI, Ser. C 147, Banff/Canada 1984*, pages 217–264. D. Reidel Publishing Company, 1985.

[13] M. Grötschel, M. Jünger, y G. Reinelt. Calculating exact ground states of spin glasses: A polyhedral approach. *Heidelberg Colloquium on Glassy Dynamics*, pages 325–353, 1987.

[14] M. Grötschel, L. Lovász, y A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[15] J. G. Kemeny. Mathematics without numbers. *Daedalus*, 88(4):577–591, 1959.

[16] A. H. Land y A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.

[17] I. Méndez-Díaz y P. Zabala. A branch-and-cut algorithm for graph coloring. *Discrete Applied Mathematics*, 154(5):826–847, 2006.

[18] C. E. Miller, A. W. Tucker, y R. A. Zemlin. Integer Programming Formulation of Traveling Salesman Problems. *J. ACM*, 7(4):326–329, October 1960.

[19] J. E. Mitchell y B. Borchers. Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm. In *High performance optimization*, pages 349–366. Springer, 2000.

[20] M. Padberg y G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Rev.*, 33(1):60–100, February 1991.

[21] H. D Sherali y W. P Adams. *A reformulation-linearization technique for solving discrete and continuous nonconvex problems*, volume 31. Springer, 1998.

[22] H. D. Sherali y P. J. Driscoll. On Tightening the Relaxations of p Miller-Tucker-Zemlin Formulations for Asymmetric Traveling Salesman Problems. *Oper. Res.*, 50(4):656–669, July 2002.

[23] L. A. Wolsey. *Integer programming.* Wiley-Interscience, New York, NY, USA, 1998.