DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

# Implementación de Cortes Generales

13 de diciembre de 2012 — Investigación Operativa

**Todos Tus Cortes**


| Integrante | LU | Correo electrónico |
|---|---|---|
| Brian Luis Curcio | 661/07 | bcurcio@gmail.com |
| Agustin Mosteiro | 125/07 | agustinmosteiro@gmail.com |
| Federico Javier Pousa | 221/07 | fedepousa@gmail.com |






**Facultad de Ciencias Exactas y Naturales**
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

# Índice

# 1. Introducción

En el presente trabajo se presenta la implementación de cortes generales para la resolución de modelos de Programación Lineal Entera, en particular para problemas con todas las variables binarias.

El trabajo tiene varios objetivos, a saber

**Interacción con CPLEX** CPLEX es un paquete de software comercial y académico para resolución de problemas de Programación Lineal y Programación Lineal Entera. CPLEX es un framework muy poderoso para la resolución de este tipo de problemas, siendo mundialmente reconocido como uno de los dos mejores y más completos softwares para este objetivo.

Uno de los objetivos de este trabajo es conocer el funcionamiento de este paquete no solo mediante la mera interacción para la resolución de problemas, sino intentando reemplazar parte del trabajo que este realiza para lograr un mejor entendimiento.

**Cortes de propósito general** Otro de los objetivos de este trabajo es poder entender los cortes de propósito general vistos en la materia. Si bien cada problema de Programación Lineal Entera merece un estudio particular para atacarlo, cada vez son más las herramientas generales que se desarrollan para tratar de resolver cualquier problema. Estas herramientas son la pieza fundamental para los paquetes de software de resolución general como CPLEX. Estas herramientas son el complemento ideal a la información particular del problema que puede proporcionar el usuario de CPLEX.

En lo que respecta a planos de corte, en la mayoría de los casos, lo mejor es buscar desigualdades válidas específicas para cada problema. Sin embargo, cuando estas no son halladas, resulta muy útil que CPLEX tenga formas de generar cortes sin importar el problema que se está tratando.

En nuestro caso, se vieron tres tipos de cortes de propósito general. Los cortes *Cover*, *Clique* y *Gomory*

**Implementación de cortes** Si bien en la materia se ve la teoría y la fundamentación de los cortes anteriormente mencionados, también es un objetivo de este trabajo toparse con las dificultades a la hora de implementar estos cortes. En varios de los cortes el preprocesamiento necesario y el algoritmo de separación no son triviales de implementar, incluyendo decisiones que tienen fuerte impacto en la performance de los mismos, tanto desde el punto de vista de la calidad del corte encontrado como desde el tiempo de ejecución consumido.

Por ejemplo, un caso claro donde se nota que la implementación no es trivial es en los cortes de Gomory. Estos utilizan información de la relajación lineal de cada nodo del arbol de branch-and-bound, que no es facilmente accesible ya que CPLEX trabaja con una versión reducida del problema original por cuestiones de eficiencia, por lo que no es simple realizar una correspondencia entre el problema reducido y el original. Es por esto que la implementación de los cortes de Gomory quedó fuera del trabajo ya que representaba un obstáculo solamente de traducción de las variables y no de algún punto interesante para analizar de la materia.

**Comparación entre diferentes métodos de resolución** Por último se buscará realizar una comparación entre diferentes métodos de resolución general.

- Branch-and-Bound: Resolución automática de CPLEX, quitando todo el preprocesamiento posible y la capacidad de generar cortes, para que la resolución sea simplemente por el arbol de Branch-and-Bound.
- Cut-and-Brach: Es un caso particular de aplicación de cortes, en donde los mismos solo se aplican al nodo raíz del arbol. La idea de esto es poner bastante esfuerzo computacional en

fortalecer el problema original de la mejor manera que se pueda, para luego lanzar un Branch-and-Bound clásico, pero que debería tomar menos tiempo al trabajar sobre una formulación más fuerte.

- Branch-and-Cut: Mezcla el arbol de Branch-and-Bound manejado por CPLEX con los cortes implementados en el trabajo. Los cortes se aplican en cada nodo del arbol para cortar soluciones fraccionarias óptimas.

# 2. Cortes Cover

## 2.1. Formulación

Los cortes cover son cortes generales que estan relacionados con restricciones mochila del problema a tratar.

Una restricción mochila es una desigualdad de la forma

$$\sum_{i=1}^{n} a_i.x_i \leq b \tag{1}$$

En donde el $b$ debe ser un número entero y los $a_i$ deben ser enteros y positivos.

Este tipo de desigualdades indican que al juntar varias variables con pesos positivos no puede ser mayor a cierto valor $b$. Esto induce a ciertas desigualdades válidas que se denominan desigualdades *cover*. Un *cover* es un conjunto de variables que, al tenerlos todas juntas con valor 1, violan una desigualdad mochila ya que se exceden de lado derecho permitido.

Por ejemplo, si se tiene la desigualdad

$$3x_1 + 5x_2 + x_3 \leq 7 \tag{2}$$

se observa que las variables $x_1$ y $x_2$ forman un cover, ya que no se podrían tener ambas en 1 en una solución factible. Obviamente, al tener un cover y agregar cualquier otra variable, se tiene nuevamente un cover porque ya de antemano se estaba excediendo el valor de $b$. Se dice que un cover es *minimal*, cuando no se puede sacar ninguna variable del conjunto y seguir teniendo un cover. En este caso los covers son {1,2,3} y {1,2}, pero solamente el segundo conforma un cover minimal.

Luego, lo que dice intuitivamente esta situación es que dado un cover $C$, en cualquier solución factible del problema no se pueden tener todas las variables pertenecientes a $C$ con valor 1. Al menos una de las variables debe estar *apagada*.

Formalmente, un cover $C$ para una desigualdad mochila induce la siguiente desigualdad válidad para el modelo.

$$\sum_{i \in C} x_i \leq \mid C \mid -1 \tag{3}$$

## 2.2. Algoritmo de separación

Para lograr introducir cortes cover se guardan todas las restricciones mochila desde un principio para luego ver que desigualdades válidas están violadas en cada nodo repasando cada una de las desigualdades mochilas originales.

Si bien no todas las restricciones del problema original son restricciones mochilas con las características pedidas, hay formas de reformular las restricciones para que se conviertan en desigualdades mochilas a ser revisadas para generar potencialmente desiguadades cover.

Existen varias formas de reformulación más y menos complejas para convertir las desigualdades originales. En este caso se utilizó un preprocesamiento simple que consiste en invertir el orden de las desigualdades que en el problema original esten con signo $\geq$. Se notó que en varias de las instancias utilizadas, muchas de las restricciones se encuentran formuladas por mayor igual y con un simple cambio de signo se logra reformular la desigualdad de forma que si cumpla con los requisitos necesarios para ser desigualdad mochila candidata a generadora de cortes cover.

Una vez que se tienen todas las desigualdades mochila del problema original, el algoritmo de separación consiste en ver en cada nodo, cada una de las desigualdades originales, y buscar si la solución de la relajación lineal del nodo actual viola o no la desigualdad cover inducida por la mochila.

Para ver si existe un cover violado se utiliza el siguiente razonamiento, donde se denomina $x^*$ a la solución del nodo actual.

Una desigualdad cover es de la forma

$$\sum_{i \in C} \leq \mid C \mid -1 \tag{4}$$

Despejando la desigualdad anterior se puede obtener la siguiente expresión

$$1 \leq \sum_{i \in C} (1 - x_i) \tag{5}$$

Luego, ver si existe un cover violado es análogo a ver si existe un cover que cumpla la siguiente desigualdad

$$1 > \sum_{i \in C} (1 - x_i^*) \tag{6}$$

Resolver esta cuestión es análogo a resolver el siguiente problema de optimización

$$minimizar \sum_{i=1}^{n} (1 - x_i^*).y_i \tag{7}$$

Sujeto a:

$$\sum_{i=1}^{n} a_i.y_i > b \tag{8}$$

Donde $y$ representa el vector característico de pertenencia a un cover.

Se esta buscando minimizar la función objetivo, pero cumpliendo la desigualdad presentada. Si el valor de la minimización da menor a 1, entonces se encontro el vector característico para un cover violado ya que, por la restricción, los pesos suman más que el $b$ permitido y, por la función objetivo, violan fehacientemente la desigualdad cover asociada.

Algo importante a notar es que como todos los valores que se manejan son enteros, la restricción del anterior modelo se puede reemplazar por una en donde se utilice un mayor igual, en vez de un mayor estricto.

$$\sum_{i=1}^{n} a_i.y_i \geq b + 1 \tag{9}$$

El problema de separación original se tradujo en resolver el problema de optimización recientemente presentado. A continuación se presentan las diferentes formas de resolver o aproximar este problema.

### 2.2.1. Greedy

Una primer idea posible para resolver este problema es hacerlo mediante un algoritmo goloso.

Si bien esta idea no resuelve el problema hasta la optimalidad, puede dar una muy buena aproximación a la solución real. Además es importante destacar que no se necesita necesariamente el mínimo del problema presentado, con obtener cualquier $y$ factible que su función objetivo valga menos que 1, se estará encontrando un cover violado por el óptimo de la relajación lineal del nodo actual.

En el problema hay que ver cuales variables se eligen para hacer lo más chico posible la función objetivo, pero a la vez es importante poder hacer válida la restricción, eligiendo variables que tengan costos asociados altos.

El algoritmo goloso diseñado trata de encontrar una solución teniendo en cuenta los dos factores mencionados. Lo que se hizo fue ir eligiendo las variables golosamente, para esto se ordenó las variables de forma decreciente respecto del valor $a_i/(1 - x_i^*)$. Este valor asociado tiene el doble objetivo de tratar de tomar variables con costos asociados altos, pero que tengan un pequeño valor asociado en la función objetivo.

Luego de tener ordenadas las variables por el criterio expuesto, se van tomando las variables golosamente hasta que se cumpla la restricción. Cuando sucede esto solo resta saber si la función objetivo tiene valor por debajo de 1 o no.

### 2.2.2. Programación Dinámica

Una segunda opción para resolver el problema de optimización propuesto es utilizar un algoritmo exacto mediante la técnica de programación dinámica.

Esta técnica se basa en obtener el óptimo para un problema, asumiendo que ya se tienen los óptimos para los subproblemas que lo conforman y que el óptimo del problema original esta necesariamente conformado por algunos de los óptimos de los subproblemas.

En este caso, el problema que se irá resolviendo será el de obtener la mínima función objetivo si se usa hasta el item $i$ de los $n$ disponibles, y se quiere sobrepasar el peso $\tilde{b}$.

Cuando $i$ sea igual a $n$ y $\tilde{b}$ sea igual a b, se estará obteniendo la mínima función objetivo utilizando todos los items y teniendo como restricción la desigualdad original del problema que se necesitaba resolver. Luego, si este valor es menor a 1 significa que hay una desigualdad cover violada para introducir como plano de corte.

Para resolver un problema con un $i$ y un $\tilde{b}$ particulares la idea es basarse en los dos siguientes casos.

- Se usa el item $i$ en la solución, entonces la mejor función objetivo es la que era la mejor solución cuando se utilizaba hasta el item $i$-1 y se quería sobrepasar $\tilde{b}$-$a_i$, sumado al valor por utilizar el item $i$ (o sea 1-$x_i^*$).
- No se utiliza el item $i$ en la solución, entonces la mejor función objetivo es la que era la mejor solución cuando se utilizaba hasta el item $i$-1 y se quería sobrepasar $\tilde{b}$.

Luego, el óptimo para el problema actual resulta de tomar la mejor opción entre las dos propuestas.

Es importante notar que en ambas opciones se utilizan óptimos de problemas que incluyen una menor cantidad de items por lo que ya se suponen resueltos si los mismos se van resolviendo en un orden adecuado.

A diferencia del algoritmo greedy, el algoritmo por programación dinámica siempre encuentra el óptimo al problema planteado por lo que si existe desigualdad violada, va a ser hallada por este algoritmo, mientras que podía suceder que la versión greedy no lo lograse.

Sin embargo, se nota que el algoritmo de programación dinámica tiene una complejidad temporal perteneciente a $\mathcal{O}(N * b)$, siendo $N$ la cantidad de variables y $b$ el rhs de la única desigualdad del modelo. Este tipo de algoritmos, en los que la complejidad depende del valor numérico de la entrada y no de su tamaño, se denominan algoritmo pseudopolinomiales y su tiempo de ejecución podría resultar prohibitivo sí se cuenta con un $b$ muy grande; mientras que el algoritmo greedy si resulta polinomial.

## 2.3. Lifteo

Algo interesante a realizar con una desigualdad válida que se quiere ingresar como plano de corte, es analizar la posibilidad de *liftearla* para fortalecerla. *Liftear* una desigualdad por menor o igual significa hacer crecer los coeficientes que acompañan a las variables, obviamente siempre y cuando la desigualdad siga siendo válida. Cuando se habla de hacer crecer los coeficientes puede ser

tanto incrementando los de las variables que ya se encuentran en la desigualdad, como agregando variables a la desigualdad con coeficientes positivos.

En el caso de las desigualdades cover existe un lifteo que es simple de demostrar su validez y a la vez es simple de implementar. Si ya se cuenta una desigualdad cover, se puede agregar a la desigualdad cualquier variable que tenga un $a_i$ asociado mayor o igual a todos los $a_i$ de las variables que ya se encuentran en la desigualdad.

# 3. Cortes Clique

## 3.1. Formulación

Lo cortes clique como el nombre lo indica requieren algún grafo subyacente del cual se puedan inferir cortes. Antés de explicar los cortes, se comenta el grafo asociado al problema.

Las instancias a resolver fueron restringidas a instancias con todas las variables binarias. Existen grafos que sirven para representar este tipo de problemas y son los *grafos de conflictos*. Lo que dice el grafo de conflicto es si existen combinaciones de pares de nodos que no pueden ocurrir, es decir, dadas las restricciones de un problema pueden existir variables $x_2$ y $x_3$ tal que que si las dos estan en 1, el conjunto de restricciones se vuelve infactible(como en la desigualdad 10). De la misma manera puede ocurrir que $x_2$ este en 1, y $x_1$ esta en 0 y que también así el conjunto de resticciones sea infactible (como en la desigualdad 11).

$$-3x_1 + 3x_2 + 5x_3 \leq 3 \tag{10}$$

$$4x_1 - 1x_2 + 3x_3 \geq 3 \tag{11}$$

Por lo tanto, existen combinaciones de pares de nodos que no pueden ocurrir en una solución del problema, cada combinación que no pueda ocurrir es un eje en el grafo de conflictos. Para representar este grafo, se tiene un conjunto $\mathcal{V}$ con dos nodos por cada variable, $x_i$ representando la variable $x_i$ en 1, y $\tilde{x}_i$ para la variable $x_i$ en 0. Por lo tanto, el conjunto $\mathcal{E}$ de ejes son los pares de nodos que no pueden ocurrir en el problema, entre ellos, los que conectan a $x_i$ con $\tilde{x}_i$ como muestra la figura 1.

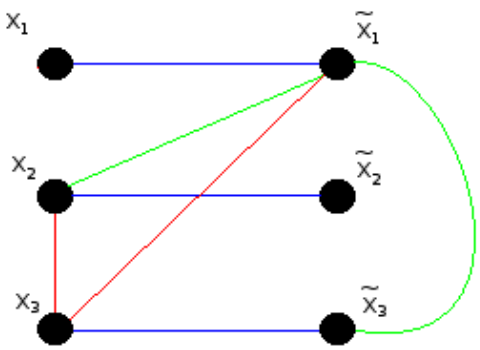


Figura 1: Grafo de conflictos del las restricciones anteriores

Una vez que se tiene armado el grafo, se puede ver que las soluciones factibles, son en realidad *conjuntos independientes* en el grafo (ie: no existen dos nodos conectados con un eje, porque están conectadas las que hacen al problema infactible). Por lo tanto cada eje representa también una restricción en el problema dependiendo la combinación variables.

- Si los nodos $x_i$ y $x_j$ son vecinos se tiene una restricción del estilo $x_i + x_j \leq 1$
- Si los nodos $x_i$ y $\tilde{x}_j$ son vecinos se tiene una restricción del estilo $x_i + 1 - x_j \leq 1$
- Si los nodos $\tilde{x}_i$ y $x_j$ son vecinos se tiene una restricción del estilo $1 - x_i + x_j \leq 1$
- Si los nodos $\tilde{x}_i$ y $\tilde{x}_j$ son vecinos se tiene una restricción del estilo $1 - x_i + 1 - x_j \leq 1$

Una idea ingenua podría ser insertar todos estas restricciones a nuestra formulación del problema lineal, reduciendo así la posibilidad de que la solución sea no entera. Pero al aumentar la cantidad de resticciones considerablemente, se tarda más en resolver el problema por el método simplex. Por lo que se va a insertar las restricciones a medida que resulten convenientes.

¿Cuando resulta conveniente? Si en la relajación lineal se viola alguna restricción, va a ser conveniente cortar soluciones de ese tipo. Como la solución será un conjunto independiente en el grafo de conflictos, entonces, todas las cliques en nuestro grafo tienen a lo sumo un nodo como parte de la solución. Entonces si al encontrar la relajación lineal se encuenta una clique que viole esta condición, entonces la resticción de que en la clique debe haber a lo sumo un nodo elegido sera útil para cortar soluciones.

$$\sum_{i \in \mathcal{K}} x_i \leq 1 \tag{12}$$

## 3.2. Construcción del grafo

Lo primero a realizar es la construcción del grafo de conflictos que se menciona en la sección 3.1. Si la instancia del problema lo permite, es posible recorrer todas las posibilidades por fuerza bruta. Es decir, para cada par de variables y para cada combinación de variable en 1 y variable en 0, se evalua si el conjunto de restricciones es infactible.

Para ver si el conjunto de restricciones es factible dado que dos variables tienen un valor fijo, se busca todas las restricciones que contengan a estas dos variables y ver si existe una valuación tal que sea factible. Esto se puede hacer de manera golosa, ya que la resticciones que son por menor o igual, basta poner en 1 todas las variables con coeficiente negativo, y en 0 todas las variables que tengan coeficiente positivo. Si usando esta valuación la restricción resulta infactible se puede probar que el conjunto de restricciones es infactible para esta combinación de nodos. De manera analoga para las restricciones por mayor o igual, se intenta que las variables que tengan coeficiente positivo esten en 1, y las demás en 0 (recordando que hay 2 nodos cuyo valor es fijo).

Para los casos donde no sea posible realizar fuerza bruta (ya sea porque hay demasiados ejes en el grafo de conflictos, o porque tarda demasiado), es posible quedarse con un conjunto al azar de variables, y probar sobre estas si existen ejes.

## 3.3. Algoritmo de separación

Una vez que tenemos la relajación lineal, lo que buscamos es la clique de mayor peso en el grafo de conflictos. Dicho problema es **NP-completo**, por lo que se realiza un heurística para encontrar algunas cliques candidatas ya que no es posible resolverlo de forma exacta.

Teniendo la clique de mayor peso, se puede determinar si esta clique esta violando la restriccione que indica si la suma de sus variables es mayor a 1. En caso de que así sea, se inserta la restricción clique (desigualdad 12) al problema.

A continuación, se presenta el pseudocódigo para generar la clique

**function** GenerarClique(k,s) ▷ k es una clique,s es el primer nodo no usado
  **for** $i \leftarrow s$ to $n$ **do**
    **if** $i$ tiene eje con todos en $k$ **then**
      $GenerarClique(k+i, i+1)$ ▷ Agregamos el nodo i a la clique
    **end if**
  **end for**
  **if** Ningún nodo pudo entrar a la clique **then**
    **if** Suma de relajaciones de k mayor a 1 **then**$MeterRestriccion(k)$
    **end if**
  **end if**
**end function**

La heurística presentada consiste en encontrar cliques maximales que sean candidatas a cortar el óptimo fraccionario. Primero se ordena a los nodos por su valor en la relajación lineal. Luego se intenta insertar cada nodo en la clique como indica el algoritmo GenerarClique. De esta manera el resultado es una clique maximal, y se puede evaluar si viola el conjunto de resticciones.

Como la cantidad de cliques en un grafo puede ser muy grande, la heurística debe dejar de buscar cliques en algún momento, por lo que se puede fijar en algún parámetro cuantas cliques se decide explorar. Una desventaja de esto es que a medida que se acerca a la solución, puede pasar que el orden de los nodos no varíe mucho, resultando en explorar siempre las mismas cliques.

# 4. Resultados

En esta sección se presentan los resultados obtenidos en las diferentes combinaciones de métodos de resolución y cortes utilizados. Dichas combinaciones fueron

- Branch-and-Bound por defecto de CPLEX. (B&B)
- Cut-and-Branch con cortes Cover. (C&B-Cov)
- Cut-and-Branch con cortes Clique. (C&B-Cli)
- Cut-and-Branch con ambos cortes. (C&B-CC)
- Branch-and-Cut con cortes Cover. (B&C-Cov)
- Branch-and-Cut con cortes Clique. (B&C-Cli)
- Branch-and-Cut con ambos cortes. (B&C-CC)

Para cada una de estas instancias se resolvió reportar y analizar los tiempos de ejecución, que suele ser la medida más importante a la hora de analizar este tipo de algoritmos; la cantidad de nodos explorados y la calidad de la cota dual conseguida en el nodo raíz.

La idea original del trabajo era realizar las corridas pertinentes sobre instancias de la MIPLIB, pero dado el tamaño de las instancias se decidió generar instancias propias con tamaños manejables por los algoritmos utilizados, con el objetivo de poder visualizar en un tiempo razonable las diferencias entre las diferentes opciones.

Todas las corridas se realizaron sobre una CPU INTEL CORE 2 DUO 2 GHZ 2GB RAM, utilizando la versión 12.4 de CPLEX.

En la siguiente tabla se muestra la información de las instancias utilizadas, mostrando la cantidad de columnas del problema, la cantidad de filas, la cantidad de desigualdades que ya son cover (dado que por como estan generadas las instancias todas son desigualdades mochila); y por último se encuentra la cantidad de ejes presentes en el grafo de conflicto utilizado para las desigualdades clique.

| Método / Instancia | CantCol | CantFil | Cover | Ejes Grafo Conflicto |
|---|---|---|---|---|
| Inst1 | 23 | 14 | 3 | 126 |
| Inst2 | 50 | 20 | 0 | 290 |
| Inst3 | 102 | 49 | 4 | 626 |
| Inst4 | 255 | 124 | 12 | 1554 |
| Inst5 | 342 | 165 | 12 | 2062 |
| Inst6 | 359 | 171 | 17 | 2236 |
| Inst7 | 407 | 197 | 18 | 2496 |
| Inst8 | 450 | 212 | 16 | 2716 |
| Inst9 | 484 | 236 | 19 | 2852 |
| Inst10 | 550 | 256 | 14 | 3334 |
| Inst11 | 637 | 308 | 26 | 3822 |
| Inst12 | 1497 | 717 | 59 | 9104 |

Cuadro 1: Información de las instancias

A continuación, se presentan los tiempos de ejecución para cada una de las instancias en cada una de las combinaciones mencionadas. Para cada corrida se fijó un tiempo límite de 20 minutos. En el caso de que se alcance el tiempo máximo establecido, lo que se reporta es el gap alcanzando.

| Mét / Ins | B&B | C&B-Cov | C&B-Cli | C&B-CC | B&C-Cov | B&C-Cli | B&C-CC |
|---|---|---|---|---|---|---|---|
| Inst1 | 0.028 | 0.02 | 0.01 | 0.02 | 0.02 | 0.02 | 0.03 |
| Inst2 | 0.35 | 0.04 | 0.03 | 0.03 | 0.04 | 0.04 | 0.03 |
| Inst3 | 439.36 | 0.07 | 22.34 | 0.06 | 0.07 | 0.78 | 0.07 |
| Inst4 | (34.30 %) | 5.22 | (10.40 %) | 0.61 | 1.66 | (8.26 %) | 0.67 |
| Inst5 | (49.52 %) | 62.92 | (32.17 %) | 5.54 | 40.85 | (25.20 %) | 7.18 |
| Inst6 | (39.19 %) | 21.41 | (24.47 %) | 1.04 | 6.16 | (16.02 %) | 1.01 |
| Inst7 | (45.05 %) | 253.80 | (27.39 %) | 19.56 | 207.82 | (19.51 %) | 55.54 |
| Inst8 | (46.75 %) | 1013.47 | (35.44 %) | 151.50 | 824.36 | (23.35 %) | 313.26 |
| Inst9 | (49.42 %) | 123.65 | (35.69 %) | 13.14 | 8.16 | (21.34 %) | 4.51 |
| Inst10 | (50.12 %) | 1175.68 | (30.95 %) | 16.73 | 297.48 | (22.90 %) | 72.96 |
| Inst11 | (49.78 %) | 626.17 | (31.50 %) | 9.29 | 235.80 | (19.64 %) | 15.00 |
| Inst12 | (64.27 %) | (5.55 %) | (45.55 %) | (2.77 %) | (4.12 %) | (28.67 %) | (1.87 %) |

Cuadro 2: Tiempos de ejecución

De la anterior tabla, hay varios resultados interesantes a analizar.

En primer lugar es claro que el Branch-and-Bound solo es la peor de las combinaciones, siendo este un resultado esperado de antemano.

Luego se observa que la inclusión de las desigualdades clique es beneficiosa para el algoritmo, ya que los gaps conseguidos son sustancialmente más bajos que los conseguidos solo con el Branch-and-Bound, también se observa que hay una notoria diferencia entre la inclusión de los cortes solo en el nodo raíz y en todos los nodos del algoritmo, de hecho la reducción del gap entre estos dos casos es casí tan grande como la reducción del gap entre el primer caso y el branch-and-bound en la mayoría de los casos. Sin embargo, se nota que la sola inclusión de este corte no logra resolver casi ninguna instancia de las presentadas.

Por otro lado, observando los cortes cover, se puede visualizar un gran incremento en la calidad del algoritmo en cuanto al tiempo de ejecución se refiere. La sola inclusión de las desigualdades cover en el nodo raíz ya hace que la mayoría de las instancias sean resueltas en el tiempo fijado. Nuevamente, al igual que en las desigualdad Clique, se puede observar que el método de Branch-and-Cut es notoriamente superior a incluir cortes solo en el primer nodo.

Luego, se observan los mejores resultados cuando se combinan los dos cortes propuestos. Incluir los dos cortes propuestos resulta en mejoras considerables del tiempo de ejecución en todas las instancias presentadas y por grandes márgenes. Un resultado interesante cuando se combinan los cortes es que en casi todas las instancias ahora la mejor combinación resulta a ser el Cut-and-Branch en vez del Branch-and-Cut, es decir que resulta más conveniente fortalecer el nodo raíz y luego lanzar un Branch-and-Bound sobre el problema fortalecido, que ir buscando cortes en todos los nodos del arbol recorrido.

Una posible explicación para esto proviene del hecho de que cuando solo se tienen en cuenta dos variables, una desigualdad clique es también una desigualdad cover y por lo tanto se utiliza mucho tiempo ejecutando los algoritmos de separación en cada uno de los nodos, muchas veces encontrando la misma desigualdad violada.

A continuación se observan la cantidad de nodos procesados por cada método en cada instancia.

| Instancia \ Método | B&B | C&B-Cov | C&B-Cli | C&B-CC | B&C-Cov | B&C-Cli | B&C-CC |
|---|---|---|---|---|---|---|---|
| Inst1 | 121 | 9 | 26 | 9 | 11 | 26 | 11 |
| Inst2 | 1713 | 21 | 64 | 5 | 10 | 52 | 5 |
| Inst3 | 1784716 | 21 | 79305 | 13 | 11 | 943 | 7 |
| Inst4 | 2597655 | 7338 | 2614682 | 399 | 1275 | 456508 | 116 |
| Inst5 | 2218211 | 83018 | 2198464 | 6541 | 29007 | 342221 | 1716 |
| Inst6 | 2133152 | 24488 | 2123171 | 549 | 3781 | 318212 | 85 |
| Inst7 | 1970321 | 292520 | 1949330 | 20163 | 127452 | 281831 | 11483 |
| Inst8 | 1823566 | 1035270 | 1795256 | 159417 | 457015 | 259647 | 58220 |
| Inst9 | 1700896 | 121732 | 1721985 | 12070 | 3722 | 240533 | 648 |
| Inst10 | 1507069 | 1025320 | 1572922 | 11010 | 132581 | 204558 | 11557 |
| Inst11 | 1349057 | 493905 | 1435534 | 5587 | 94046 | 177712 | 1937 |
| Inst12 | 628987 | 519323 | 649291 | 547822 | 222469 | 72630 | 62321 |

Cuadro 3: Cantidad de nodos

Al igual que en los tiempos de ejecución, la anterior tabla muestra tendencias marcadas que se condicen en todas las instancias.

En primer lugar se nota que este caso la cantidad de nodos que procesan el Branch-and-Bound y la versión clique en Cut-and-Branch están en el mismo orden, esto se debe a que en el segundo caso solo se está poniendo un poco más de esfuerzo en el nodo raíz pero no alcanza para resolver las instancias en los tiempos dados por lo que en ambos casos se está cortado la ejecución por tiempo límite y ambas combinaciones terminan procesando una cantidad similar de nodos.

Luego, se puede ver que si bien el Branch-and-Cut con cortes clique también cortaba por tiempo, se observa que, logicamente, la cantidad de nodos procesados en este es bastante menor, porque se está poniendo más esfuerzo computacional en cada nodo para encontrar mejores cortes que eviten menos *brancheo* del arbol de resolución.

Nuevamente las desigualdades cover por si solas ya son una muy buena mejora a la opción sin cortes, ya solo en Cut-and-Branch se puede observar como casi todas las instancias se resuelven hasta la optimalidad explorando una cantidad de nodos muchisimo menor a las combinaciones analizadas recientemente. Al igual que en las desigualdad clique, se puede observar que la cantidad de nodos necesitados por el Branch-and-Cut para llegar hasta la optimalidad es mucho menor a solamente fortalecer la formulación en el nodo raíz.

Por último, se observa nuevamente como la combinación de las dos familias de cortes muestran una mejoría notable. La cantidad de nodos procesados por las combinaciones de los cortes esta en menos del 10 % de los nodos utilizados por el resto de los casos en la mayoría de las instancias, excepto por las instancias muy pequeñas que se resuelven casi instantaneamente.

Si bien en la tabla anterior se pudo apreciar que la mejor combinación en cuanto a tiempos de ejecución resulto ser el Cut-and-Branch con las dos familias de cortes, es lógico que ahora esa tendencia sea la inversa en cuanto a la cantidad de nodos se refiere. La cantidad de nodos explorados por el algoritmo de Branch-and-Cut es casi siempre menor a los explorados por el Cut-and-Branch. Nuevamente esta tendencia se explica por el esfuerzo realizado en cada nodo para obtener mejores cotas duales.

A continuación, se presentan las cotas duales que se cuentan al salir del nodo raíz en cada instancia para cada una de las combinaciones procesadas.

Dado que solo se realiza el estudio de cortes sobre el nodo raíz, no tiene sentido hacer la distinción entre Cut-and-Branch y Branch-and-Cut

| Instancia \ Método | B&B | Cov | Cli | CC |
|---|---|---|---|---|
| Inst1 | 63.102984 | 50.718318 | 57.290728 | 50.718318 |
| Inst2 | 131.694127 | 96.169751 | 105.196265 | 91.543583 |
| Inst3 | 241.328893 | 177.003756 | 233.579693 | 176.221233 |
| Inst4 | 657.440072 | 475.573907 | 636.544978 | 468.910325 |
| Inst5 | 941.943356 | 632.583498 | 920.353932 | 626.323623 |
| Inst6 | 909.175963 | 655.330574 | 883.845455 | 644.841459 |
| Inst7 | 1111.512719 | 789.187803 | 1089.283174 | 778.825366 |
| Inst8 | 1111.091109 | 799.345685 | 1074.689065 | 793.736217 |
| Inst9 | 1337.292364 | 923.283656 | 1321.944489 | 916.926981 |
| Inst10 | 1461.620012 | 1027.316771 | 1416.152038 | 1012.057732 |
| Inst11 | 1689.316530 | 1190.200534 | 1654.273710 | 1185.898751 |
| Inst12 | 4016.096160 | 2793.529570 | 3911.505935 | 2748.371351 |

Cuadro 4: Relajación inicial

Dado que las instancias utilizadas son todas problemas de maximización, lo deseable sería obtener valores lo más pequeños posibles para las cotas duales.

En esta tabla la tendencia que se venía dando con los anteriores resultados es irrefutable, en absolutamente todas las instancias se puede ver que el Branch-and-Bound obtiene las peores cotas duales, lo cual es lógico porque todos los demás métodos comienzan con esta solución como punto de partida.

Luego se encuentra la inclusión de las desigualdades clique que en todas las instancias mejora las cotas del Branch-and-Bound, pero los resultados obtenidos se encuentran en el mismo orden de magnitud.

Ya si con una diferencia notoria se encuentra la utilización de la desigualdades cover, que en los casos más interesantes logran cotas que están en el orden de los dos tercios de las conseguidas por las desigualdades clique.

Por último, la mejor columna en todos los casos es la perteneciente a la combinación de las dos familias, reforzando una vez más el hecho de que es importante la inclusión de este tipo de cortes generales. Si bien esta es la mejor columna de las 4, cabe notar que no muestra mucha diferencia con los resultados arrojados por las desigualdades cover.

Como análisis general, se observa que la inclusión de las dos familias de cortes generales resulto beneficioso en todos los aspectos presentados, mostrando grandes diferencias con el método que se contaba originalmente, el Branch-and-Bound.

# 5. Conclusiones

Luego del trabajo realizado y de los resultados obtenidos y analizados, se desprenden varias conclusiones generales sobre lo estudiado

**Cortes generales** La conclusión más evidente de todo el trabajo, pero que no por eso se puede dejar de mencionar, es el hecho de la gran utilidad que se le encontro a los cortes implementados.

Trabajando con instancias pequeñas se observó que si bien el algoritmo de Branch-and-Bound es teoricamente correcto, resulta muy lento y necesita explorar una cantidad altísima de nodos para, no solo tratar de llegar al óptimo, si no también lograr el certificado de optimilidad.

La inclusión de los cortes generales hizo que sin explotar la estructura particular de cada problema, muchas de las instancias propuestas se pudieran resolver en el tiempo límite fijado.

**Cut-and-Branch y Branch-and-Cut** Si bien de antemano se podría pensar que el Cut-and-Branch no marcaría una gran diferencia con el Branch-and-Bound, al solo efectivizar los cortes en el nodo raíz. Se encontró que en la mayoría de los casos estudiados el Cut-and-Branch no solo mejora ampliamente los tiempos de ejecución del Branch-and-Bound, si no que también mejora, en un nivel mucho menor, los tiempos de ejecución del Branch-and-Cut. Por lo que parece suceder que el problema con muchas de las instancias es que la formulación inicial es muy débil cuando se compara el conjunto de puntos enteros que se quiere analizar y la relajación lineal obtenida por la formulación.

**Implementación** Si bien el desarrollo teórico de un corte general es un punto más que necesario, para poder demostrar que el corte sirve como tal y para poder encarar el problema de separación; la implementación práctica de los cortes es un tema para nada menor.

El hecho de que las desigualdades cover se hayan comportado mejor que las clique en estas instancias, no indica que esto vaya a suceder siempre. Un posible problema en las desigualdades clique es que la implementación, tanto a la hora de armar el grafo de conflicto, como al buscar la clique, puede haber sido demasiado trivial, y se necesiten mejores algoritmos para resolver los dos problemas centrales de estas desigualdades.

Otras cosas que han sido notadas durante la implementación y la experimentación, es como fluctuaron los tiempos de ejecución no por cuestiones algorítmicas, sino por cuestiones meramente de codificación. El ejemplo más claro se dió en la elección del momento para asignar y liberar memoria dinámica al algoritmo, un ejemplo que tiene el agravante de que no puede ser controlado en cualquier lenguaje de programación que se utilice.

**MIPLIB** Si bien la idea original del trabajo era realizar las pruebas sobre instancias de la MIPLIB, no se logró trabajar con estas ya que los tiempos de ejecución para resolver las instancias más pequeñas eran muy elevados, y también resultaba complicado encontrar instancias donde se pudiese observar la inclusión de los cortes clique.

Este problema puede ser generado por el hecho de que las dos familias de desigualdades no sean lo suficientemente fuertes como para poder resolver instancias grandes, como así también puede haber sido generado por implementaciones muy ingenuas de los algoritmos de separación. Algo que si es importante destacar, es que al deshabilitar todos las ayudas de CPLEX, dejando solamente el manejo del arbol, no hay ningún tipo de búsqueda de soluciones factibles, a menos que se consiga una al resolver la relajación de un nodo. Es decir, la inclusión de cortes generales ataca un solo lado del problema, solamente busca ajustar las cotas duales, pero el no tener ningún abordaje por el lado primal puede ser uno de los principales motivos para que los tiempos de ejecución sean elevados.

**CPLEX** Otra de las conclusiones obvias pero que no dejan de ser notables, es el gran poder que tiene CPLEX como software de resolución general. CPLEX tiene muchisimos años de desarrollo, y presenta muchisimos cortes y heurísticas con implementaciones muy evolucionadas. Todas las instancias que fueron resueltas en este trabajo, CPLEX las resuelve casi instantaneamente si se le deja usar todas las ayudas que vienen incluidas. Es notorio que se logre hacer tanta diferencia en la resolución de los problemas, sin tener ningún tipo de información sobre la estructura del mismo como para poder entender mejor el conjunto de puntos con el que se está trabajando.