ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE CIENCIAS

MÉTODOS EXACTOS PARA EL PROBLEMA DE *K*-PARTICIONAMIENTO CON RESTRICCIONES DE TAMAÑO Y PESO

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERA MATEMÁTICA

PROYECTO DE INVESTIGACIÓN

YOHANDRA RUBI ARIAS NAVARRETE
yohandra.arias@epn.edu.ec

ANA JULIA ESCOBAR ORTIZ
ana.escobar@epn.edu.ec

Director: RAMIRO DANIEL TORRES GORDILLO
ramiro.torres@epn.edu.ec

QUITO, AGOSTO 2018

DECLARACIÓN

Nosotras, Yohandra Rubi Arias Navarrete y Ana Julia Escobar Ortiz declaramos bajo juramento que el trabajo aquí escrito es de nuestra autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual, correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su reglamento y por la normatividad institucional vigente.

Rubi Arias

Yohandra Rubi Arias Navarrete

Ana Escobar.

Ana Julia Escobar Ortiz

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Yohandra Rubi Arias Navarrete y Ana Julia Escobar Ortiz, bajo mi supervisión.

Ramiro Daniel Torres Gordillo
Director

AGRADECIMIENTOS

Agradecemos a Dios por darnos la fuerza y voluntad para culminar este trabajo. A nuestros padres por estar siempre con nosotras. A nuestro tutor, Ramiro Torres, por su paciencia a todas las adversidades presentadas durante este camino, por su apoyo y conocimiento brindado durante toda la carrera; porque es un maestro que supo motivarnos para poder culminar con éxito nuestro proyecto.

DEDICATORIA

A mis padres, Javier y María, por brindarme su apoyo y amor incondicional en cada uno de mis pasos durante este camino. A mis hermanos, Erika, Jefferson y Adrián, por ser un buen ejemplo a seguir en cada diferente etapa de mi vida y por estar siempre presentes cuando más los he necesitado. A Xavier por nunca dejarme bajar los brazos a pesar de los problemas y por construir una hermosa historia a mi lado. A Migue, Pablo, David, Luis y Alexander, por demostrarme que la amistad puede ser verdadera, sincera y pura hasta en las peores circunstancias.

Rubi

A mi familia, Narcisa, Jorge, Berthita y Jorgito por estar siempre presentes. A Oscar por apoyarme y cuidarme en todo momento. A mis amigos que estuvieron presentes con consejos y enseñanzas. A mis primos y tíos que fueron parte importante en mi carrera universitaria.

Ana Julia

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Resumen

Sea $G := (V, E)$ un grafo no dirigido con función de costos sobre las aristas y pesos sobre los nodos. El problema de particionamiento sobre G, consiste en dividir a V en subconjuntos o cliques con el fin de minimizar la suma de los costos de las aristas que conectan los nodos de cada subconjunto.

En el presente trabajo se formulará un modelo de programación lineal entera para el problema de $k-$ particionamiento con restricciones de tamaño y peso, que será comparado con el modelo propuesto por Recalde *et al.* [2016]. Para las dos formulaciones lineales se usará un algoritmo tipo Branch & Cut como método exacto de solución, en los que se incluirán diferentes clases de planos cortantes como las desigualdades triangulares, 2-partición y de cubrimiento conocidas como cover. Finalmente, resultados computacionales basados en instancias reales y simuladas son reportadas.

Palabras clave: Problema de particionamiento de grafos, programación lineal entera, planos cortantes, algoritmo Branch & Cut.

Abstract

Let $G := (V, E)$ be a undirected graph with edge costs and node weigths. The partition problem over G is defined as finding a partition of V in a fixed number of subsets of nodes (or cliques), such that the sum of the cost of the edges with both ending nodes in the same subset is minimized.

In this work an integer lineal programing model for the $k-$ partition problem with size and weight restrictions is formulated, and this model is compared to the model proposed in Recalde *et al.* [2016]. For the two linear formulations, a Branch & Cut method will be used as an exact solving method. Furthermore, different types of cutting plane, such as triangular inequialities, 2-partition and cover planes will be included. Finally, computational results based upon both real and simulated instances are presented.

Keywords: Graph partitioning problem, integer linear programming, cutting planes, Branch & Cut algorithm

Notaciones

$G = (V, E)$	Grafo no dirigido
$K_n = (V_n, E_n)$	Grafo completo
V	Conjunto de nodos
E	Conjunto de aristas
k	Número de grupos a formarse
n	Número de nodos
$\mathbb{R}^{E_n}$	Espacio de vectores indexados por el conjunto de aristas E_n del grafo K_n
$\delta(i) := \{\{i, j\} \in E\}$	Conjunto de aristas incidentes al nodo $i \in V$
S	Subconjunto de V
$E(S) := \{\{u, v\} : u, v \in S\}$	Conjunto de aristas en G con ambos nodos en S

Capítulo 1

Introducción

El problema de particionamiento de grafos ha sido ampliamente estudiado desde 1970 (Bichot y Siarry [2011]), pues es un problema transversal para campos como la informática, ingeniería, deportes y otros. Este consiste en encontrar una partición del conjunto de nodos de un grafo no dirigido, considerando restricciones de balance sobre el número de nodos y de peso sobre cada partición, tal que se minimice el costo total de las aristas dentro de cada partición. A pesar de que el problema ha sido estudiado a lo largo del tiempo por Grötschel y Wakabayashi [1989], Ferreira *et al.* [1998], Labbé y Ozsoy [2010], Jaehn y Pesch [2013] y recientes aportes descritos por Recalde *et al.* [2016], se considera que aún existe trabajo por hacer en el diseño de nuevas formulaciones y en la construcción de nuevos métodos de solución que permitan resolver este problema de manera eficiente. Además, la existencia de aplicaciones con instancias cada vez más grandes, como se puede observar en problemas derivados de control de tráfico aéreo reportado por Bichot [2007] , simulaciones en Schloegel *et al.* [2003], circuitos VLSI en Kahng *et al.* [2011], redes móviles en Fairbrother *et al.* [2017], programación de equipos deportivos en Mitchell [2003], entre otros; hace que el problema de particionamiento de grafos se vuelva cada vez más importante, polifacético y desafiante.

Podemos iniciar el estudio del problema de particionamiento de grafos citando el trabajo de Bichot y Siarry [2011], quienes clasifican al problema en dos grupos: el primero conocido como particionamiento restringido, donde los tamaños de cada partición deben ser similares; y el segundo considerando particiones sin restricciones, es decir, cada partición puede ser de cualquier tamaño. Ambos presentan formulaciones similares, sin embargo, los algoritmos utilizados para resolverlos son diferentes en su implementación y en su propósito.

El área de investigación más estudiada se enfoca en el particionamiento restringido, grupo en el que se encuentra el problema central del presente trabajo. El problema de k-particionamiento con restricciones de tamaño y peso consiste en construir subconjuntos (cliques), donde el número de nodos en cada clique difiere en máximo 1 unidad y la suma total de los pesos en los nodos en cada clique respete cotas superiores e inferiores, tal que el costo total de las aristas con nodos finales en el mismo subconjunto sea minimizado. La motivación para plantear este proyecto se deriva de la aplicación en la segunda categoría del fútbol ecuatoriano estudiado en Recalde *et al.* [2016], donde los equipos de fútbol deben ser divididos en k grupos de acuerdo con algunas regulaciones impuestas por la Federación Ecuatoriana de Fútbol, con el objetivo de minimizar la distancia total de los viajes por carretera que todos los equipos deben realizar para jugar un Torneo Doble Round Robin.

Se ha observado en el artículo antes mencionado que los tiempos de resolución para algunas instancias son altos, por lo que el objetivo principal del presente proyecto es mostrar que, gracias a la inclusión de planos cortantes en los modelos estudiados, se obtienen mejores soluciones o incluso soluciones óptimas en tiempos razonables.

El presente trabajo de titulación presenta la siguiente estructura: En el capítulo 2 se muestra algunas definiciones previas. La formulación de los modelos de programación lineal entera para el problema de k-particionamiento con restricciones de tamaño y pesos se presentan en el capítulo 3. En el capítulo 4 se procederá a la identificación de desigualdades válidas (triangulares, 2-partition y cover) que serán incluidas en la formulación del problema de k-particionamiento con restricciones de tamaño y peso, como planos cortantes. En el capítulo 5 se procederá a la presentación de los resultados computacionales, y finalmente en el capítulo 6 se darán las conclusiones obtenidas del trabajo realizado.

Capítulo 2

Definiciones

Antes de presentar el problema de k-particionamiento con restricción de tamaño y peso, es necesario introducir algunas definiciones, notaciones y terminologías. En su mayoría fueron tomadas de Bichot y Siarry [2011], Grötschel y Wakabayashi [1989], Bertsimas y Tsitsiklis [1998], Oktay *et al.* [2011] y Ji y Mitchell [2005], y serán constantemente utilizadas en este trabajo.

2.1 Planos cortantes

En la presente sección se definirán conceptos elementales para entender y resolver un problema de programación entera.

DEFINICIÓN 2.1. *Un programa lineal (LP) es un problema que optimiza una función lineal de varias variables, conocidas como variables de decisión, sobre una región definida por ecuaciones y desigualdades lineales.*

Formalmente, un LP puede ser presentado de la siguiente forma:

$$\begin{array}{ll} \text{mín } c^T \boldsymbol{x} & \\ \text{Sujeto a : } \quad A\boldsymbol{x} \geq b & \\ \qquad\qquad \boldsymbol{x} \geq 0 & \end{array} \quad o \quad \begin{array}{ll} \text{máx } c^T \boldsymbol{x} & \\ \text{Sujeto a : } \quad A\boldsymbol{x} \leq b & \\ \qquad\qquad \boldsymbol{x} \geq 0 & \end{array} \tag{2.1}$$

donde $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$ y $\boldsymbol{x}$ es un vector no nulo de dimensión n.

Existen extensiones de un *LP* con cambios en las variables de decisión, donde sus formulaciones se reescriben en la forma estándar (2.1) con la inclusión de ciertas restricciones. Entre ellos se encuentran los problemas de programación lineal ente-

ra, donde las variables son enteras; entera-mixta, si una parte de las variables son enteras y las restantes continuas; y binario $0-1$, cuando las variables enteras están restringidas exclusivamente a dos posibles valores $\{0,1\}$, denominadas variables binarias.

El resto del capítulo se centrará en el estudio del problema de programación lineal entera, debido a que los problemas de programación entera-mixta y binario $0-1$ son casos particulares.

DEFINICIÓN 2.2. *Un programa lineal entero (IP) es un problema de programación lineal, sujeto a la restricción adicional de que las variables de decisión* $\boldsymbol{x}$ *tomen valores enteros:*

$$\begin{array}{lll} \text{mín} & c^T\boldsymbol{x} & \\ \textit{sujeto a}: & A\boldsymbol{x} & \geq b \\ & \boldsymbol{x} & \geq 0 \\ & \boldsymbol{x} & \in \mathbb{Z}^n \end{array}$$

Notar que el vector que satisface todas las restricciones se lo conoce como solución factible, y al conjunto de todas las soluciones factibles se lo conoce como región factible denotado por $S := \{\mathbf{x} \in \mathbb{Z}^n : A\mathbf{x} \geq b\}$. Si S esta vacío entonces el problema no tendrá solución, es decir, IP es no factible; caso contrario, dependiendo si está acotado presentará una o múltiples soluciones y si no está acotado habrá infinitas soluciones.

Encontrar el punto óptimo o la respuesta óptima de un problema de programación discreta es usualmente difícil, por lo cual se opta por relajar las restricciones o la función objetivo. La mayoría de los métodos de solución para un IP parten de esta relajación lineal del problema y considerando que la relajación lineal de S es un conjunto de la forma $S' := \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \geq b\}$, tal que $S \subseteq S'$, se presenta el siguiente concepto.

DEFINICIÓN 2.3. *Dado un problema de programación lineal entera (IP), el problema de programación lineal*

$$\begin{array}{lll} \text{mín} & c^T\boldsymbol{x} \\ \textit{sujeto a}: & \boldsymbol{x} \in S' \end{array}$$

es definido como la relajación lineal (PL) para IP.

En el presente proyecto se usará la relajación continua definida de la siguiente manera.

DEFINICIÓN 2.4. *Dado un problema de programación lineal entera (IP), las relajaciones continuas se forman al tratar cualquier variable discreta como continua, manteniendo todas las otras restricciones.*

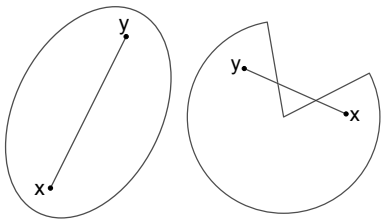
Hay que tener en cuenta que al realizar la relajación continua a un IP, no se pierden soluciones factibles y el proceso de solución del modelo relajado suele ser significativamente más tratable.

A lo largo de los años se han desarrollado diversos métodos exactos para resolver los problemas IP, los cuales serán tratados a continuación. Pero antes, es necesario presentar algunos conceptos básicos sobre geometría poliedral.

DEFINICIÓN 2.5. *Un conjunto $T \subset \mathbb{R}^n$ es convexo si y solo para todo $\boldsymbol{x}, \boldsymbol{y} \in T$ y para todo $\lambda \in [0,\ 1]$ se tiene que:*

$$\lambda \boldsymbol{x} + (1 - \lambda)\boldsymbol{y} \in T$$

El siguiente gráfico ilustra dicha definición:



(a) Convexo **(b)** No convexo

Las restricciones de un IP son conjuntos convexos y formalmente se encuentran definidos de la siguiente manera.

DEFINICIÓN 2.6. *Sea $a \in \mathbb{R}^n$ con $a \neq 0$ y b un escalar:*

1. *El conjunto $\{\boldsymbol{x} \in \mathbb{R}^n : a^T\boldsymbol{x} = b\}$ se llama hiperplano.*
2. *El conjunto $\{\boldsymbol{x} \in \mathbb{R}^n : a^T\boldsymbol{x} \geq b\}$ se llama semi-espacio.*

De las definiciones precedentes se concluye que:

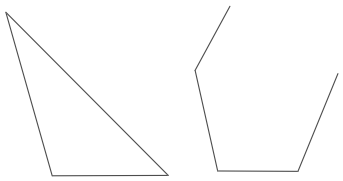
DEFINICIÓN 2.7. *Un poliedro es un conjunto que puede ser descrito de la forma:*

$$\mathcal{P} = \{\boldsymbol{x} \in \mathbb{R}^n : A\boldsymbol{x} \geq b\}$$

donde A es una matriz de $m \times n$ y b es un vector de $\mathbb{R}^m$.

En una perspectiva geométrica se considera a un poliedro como la intersección finita de semi−espacios.

Además, dado que la región factible de un IP define un poliedro $\mathcal{P}$ y éste puede ser acotado o extenderse hasta el infinito, como se puede observar en la figura 2.1, es necesario introducir la siguiente definición que nos ayudará a distinguir cuando una región factible es acotada.



(a) Acotado **(b)** No acotado

Figura 2.1: Poliedros

DEFINICIÓN 2.8. *Un conjunto $S \subset \mathbb{R}^n$ es acotado si existe una constante k, tal que el valor absoluto de cada elemento de S es menor igual que k.*

En el presente trabajo nos concentramos solo en poliedros que son acotados. Previo a su definición, se presentan los siguientes conceptos:

DEFINICIÓN 2.9. *Sea $x^1, \ldots, x^k$ vectores en $\mathbb{R}^n$ y sea $\lambda_1, \ldots, \lambda_k$ escalares no negativos tal que $\sum_{i=1}^{k} \lambda_i = 1$, entonces:*

1. *El vector $\sum_{i=1}^{k} \lambda_i x^i$ se llama **combinación convexa** de los vectores $x^1, \ldots, x^k$.*
2. *La **envolvente convexa** de los vectores $x^1, \ldots, x^k$ es el conjunto de todas las combinaciones convexas de dichos vectores.*

Junto con las conceptos presentados, se formaliza la siguiente definición.

DEFINICIÓN 2.10. *Un politopo $\mathcal{P}$ es definido como la envolvente convexa de puntos finitos, es decir:*

$$\mathcal{P} = conv\{x^1, \ldots, x^k\} = \left\{ \begin{array}{ll} & x = \sum_{j=1}^{k} \lambda_j x^j \\ x \in \mathbb{R}^n : & \sum_{j=1}^{k} \lambda_j = 1 \\ & 0 \leq \lambda_j \leq 1, \qquad j = 1, \ldots, k \end{array} \right\}$$

Un politopo $\mathcal{P}$ también puede ser definido como un poliedro acotado.

Desde el punto de vista geométrico es importante tomar en cuenta las caras y dimensión de un politopo. Para ello se considera las siguientes definiciones.

DEFINICIÓN 2.11. *Dado un politopo $\mathcal{P}$, una desigualdad de la forma $\alpha x \leq \beta$ es una desigualdad válida para $\mathcal{P}$ si se satisface para todo $x \in \mathcal{P}$.*

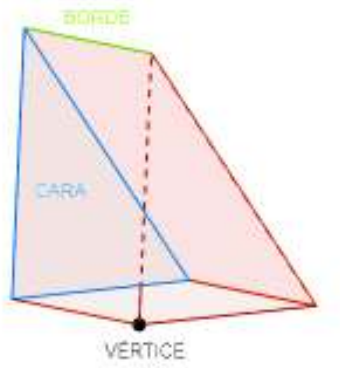
DEFINICIÓN 2.12. *Se llama cara de un politopo $\mathcal{P}$ correspondiente a la desigualdad $\alpha x \leq \beta$ al conjunto $F = \{x \in \mathcal{P} : \alpha x = \beta\}$. Si $F \neq \emptyset$ y $F \neq \mathcal{P}$, la cara se llama propia. En este caso se dice que $\alpha x \leq \beta$ define una cara.*

DEFINICIÓN 2.13. *Un conjunto de puntos $x^1, \ldots, x^k \in \mathbb{R}^n$ es afinmente independiente, si la única solución para $\sum_{i=1}^{k} \lambda_i x^i = 0$ y $\sum_{i=1}^{k} \lambda_i = 0$ es $\lambda_i = 0$ para todo $i \in \{1, \ldots, k\}$.*

DEFINICIÓN 2.14. *Un politopo $\mathcal{P}$ es de dimensión d, si tiene a lo más $d + 1$ puntos afinmente independientes.*

Dado a que es necesario restringir el espacio de soluciones para la obtención de soluciones enteras, es necesario considerar los siguientes conceptos.

DEFINICIÓN 2.15. *Sea $\mathcal{P}$ un poliedro de dimensión d.*



- *Una cara o faceta es de dimensión d - 1.*
- *Una cara de dimensión 1, es un borde.*
- *Una cara de dimensión 0, es un vértice.*

Por otro lado, debido a la importancia de conocer la metodología para llegar a la solución de un IP, a continuación se presentan los principales métodos exactos.

Planos cortantes: El método de los planos cortantes fue desarrollado por Gomory [1958] para resolver programas lineales enteros. Su propósito es mejorar o ajustar la relajación lineal de un IP mediante la incorporación de desigualdades que resulten válidas para todas las soluciones enteras del problema a tratarse.

DEFINICIÓN 2.16. *Los planos cortantes son desigualdades lineales que satisfacen todas las soluciones factibles de un IP, pero pueden ser violadas por soluciones para su relajación continua.*

El esquema general del método consiste en resolver la relajación lineal del IP, produciendo así una solución x^*. Si $x^* \in \mathbb{Z}_+^n$, entonces x^* es óptimo; caso contrario, se identifica un plano cortante $a_1x_1 + .. + a_nx_n \leq b$ y es introducido en la relajación, cortando el punto x^*. Este proceso se repite hasta que se identifica una solución entera factible.

Algoritmo 1: Planos cortantes

Entrada: Un programa lineal entero IP. Sea x^* la solución del LP.

```
1 Resolver la relajación lineal PL
2 si x* ∈ Z+^n entonces
3     x* es la solución óptima del IP y el algoritmo termina.
4 en otro caso
5     agregar una restricción a PL, tal que sea satisfecha por toda solución
      entera del IP, pero no por x*.
```

El potencial de los planos cortantes para resolver problemas generales fue demostrado por Gomory [1958] por primera vez en el ajuste de la programación entera, en donde se describió un procedimiento que identifica una secuencia correcta de planos cortantes que garantiza una solución entera en finitas iteraciones.

Hoy en día existen algoritmos de planos cortantes diseñados para ser aplicados sobre cualquier IP, donde la fuerza impulsora detrás de estos algoritmos surgió en la década de los 60 conocida como el algoritmo fraccionario de Gomory (Gomory [1960a]). El algoritmo de Gomory consiste en resolver el problema sin considerar las restricciones de carácter entero de las variables, y si la solución no es entera añade restricciones que reduce el conjunto de soluciones del problema lineal continuo

asociado, sin excluir ninguna solución entera.

Algoritmo 2: Corte de Gomory

Entrada: Un programa lineal entero IP. Sea x^* la solución del LP, $\lfloor a \rfloor$ el entero más grande menor que a, y se define la parte fraccional de a como $a - \lfloor a \rfloor$.

1 Resolver la relajación lineal PL.

2 **si** $x^* \in \mathbb{Z}_+^n$ **entonces**

3 x^* es la solución óptima del IP y el algoritmo termina

4 **en otro caso**

5 De la tabla resultante aplicando el método Simplex, elegir cualquier restricción no entera b_i^*:

$$\sum_j a_{ij}^* x_j = b_i^*$$

6 Usando partes fraccionarias $f_{ij} = a_{ij} - \lfloor a_{ij} \rfloor$, $f_i = b_i - \lfloor b_i \rfloor$, la restricción se reescribe:

$$\sum_j f_{ij}^* x_j - f_j^* = \lfloor b_j^* \rfloor - \sum_j \lfloor a_{ij}^* \rfloor x_j$$

7 Agregar la nueva restricción $\sum_j f_{ij} x_j - f_j \geq 0$ con exceso entero.

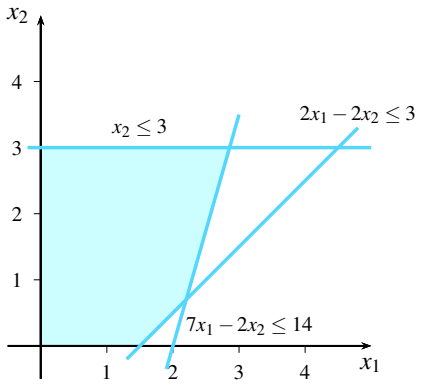
8 Repetir los pasos 5-7 hasta que todos los b_i^* sean enteros.

A continuación se considera un problema de programación lineal entera de dos variables como una ilustración al enfoque del método de los planos cortantes y al corte fraccional de Gomory.

$$\begin{aligned} \text{máx}\, z &= 4x_1 - 2x_2 \\ \text{Sujeto a :} \quad 7x_1 - 2x_2 &\leq 14 \\ x_2 &\leq 3 \\ 2x_1 - 2x_2 &\leq 3 \\ x_1,\ x_2 &\in \mathbb{Z}_+ \end{aligned}$$

Su relajación PL y su región factible se encuentran definidas de la siguiente forma:

$$\begin{aligned} \text{máx}\, z &= 4x_1 - 2x_2 \\ \text{Sujeto a :} \quad 7x_1 - 2x_2 &\leq 14 \\ x_2 &\leq 3 \\ 2x_1 - 2x_2 &\leq 3 \\ x_1,\ x_2 &\geq 0 \end{aligned}$$



Se considera una variable z que representa la función objetivo y tres variables de holgura (x_3, x_4, x_5) para convertir las restricciones de desigualdades en igualdades. Resolviendo la relajación del IP mediante el método del simplex, se obtiene que la solución óptima (ver tabla 2.1) correspondiente es $(x_1, x_2) = (1.857, 3)$ y cuyo valor objetivo es $z = 8.43$.

Tabla 2.1
Solución simplex del PL

	z	x_1	x_2	x_3	x_4	x_5	
z	1	0	0	0.57	0.14	0	8.43
x_2	0	0	1	0	1	0	3
x_5	0	0	0	-2/7	10/7	1	3.28
x_1	0	1	0	1/7	2/7	0	13/7

Debido a que el valor de x_1 no es entero, (x_1, x_2) no es una solución para el IP, por lo que se procede a incluir una restricción lineal a la relajación del problema. Para ello, consideramos la restricción $x_1 + \frac{1}{7}x_3 + \frac{2}{7}x_4 = \frac{13}{7}$ de la tabla 2.1, y utilizando el algoritmo de Gomory se procede a generar el primer corte como se indica a continuación:

$$(1 - \lfloor 1 \rfloor)\, x_1 + \left(\frac{1}{7} - \left\lfloor \frac{1}{7} \right\rfloor\right) x_3 + \left(\frac{2}{7} - \left\lfloor \frac{2}{7} \right\rfloor\right) x_4 \geq \frac{13}{7} - \left\lfloor \frac{13}{7} \right\rfloor$$

$$\frac{1}{7}x_3 + \frac{2}{7}x_4 \geq \frac{6}{7}$$

Teniendo así el famoso corte fraccional de Gomory.

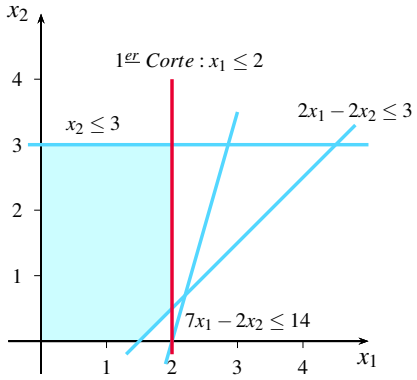
Dado que $x_3 = 14 - 7x_1 + 2x_2$ y $x_4 = 3 - x_2$, se tiene que el corte fraccional de

Gomory puede expresarse en el espacio (x_1, x_2) de la siguiente forma:

$$\begin{aligned} \frac{1}{7}(14 - 7x_1 + 2x_2) + \frac{2}{7}(3 - x_2) &\geq \frac{6}{7} \\ 14 - 7x_1 + 2x_2 + 6 - 2x_2 &\geq 6 \\ -7x_1 &\geq -14 \\ x_1 &\leq 2 \end{aligned}$$

Una vez encontrada esta desigualdad, se procede a añadirla en el PL:

$$\begin{aligned} \text{máx}\, z &= 4x_1 - 2x_2 \\ \text{Sujeto a :} \quad 7x_1 - 2x_2 &\leq 14 \\ x_2 &\leq 3 \\ 2x_1 - 2x_2 &\leq 3 \\ x_1 &\leq 2 \\ x_1,\ x_2 &\geq 0 \end{aligned}$$



De igual manera se considera una nueva variable de holgura (x_6) para la desigualdad añadida, obteniendo que la solución óptima es $(x_1, x_2) = (2, 0.5)$ y cuyo valor objetivo es $z = 7.5$ (ver tabla 2.2).

Tabla 2.2
Solución simplex del PL añadido el primer corte

	z	x_1	x_2	x_3	x_4	x_5	x_6	
z	1	0	0	0	0	1/2	3	15/2
x_3	0	0	0	1	0	-1	-5	1
x_4	0	0	0	0	1	1/2	-1	5/2
x_1	0	1	0	0	0	0	1	2
x_2	0	0	1	0	0	-1/2	1	1/2

Dado que x_2 no es entero, es necesario generar un nuevo corte. Para ello se considera la restricción $x_2 - \frac{1}{2}x_5 + x_6 = \frac{1}{2}$ de la tabla 2.2, y siguiendo el mismo razonamiento del primer corte se tiene que:

$$\begin{aligned} (1 - \lfloor 1 \rfloor)\, x_2 + \left(-\frac{1}{2} - \left\lfloor -\frac{1}{2} \right\rfloor \right) x_5 + (1 - \lfloor 1 \rfloor)\, x_6 &\geq \frac{1}{2} - \left\lfloor \frac{1}{2} \right\rfloor \\ \frac{1}{2}x_5 &\geq \frac{1}{2} \end{aligned}$$

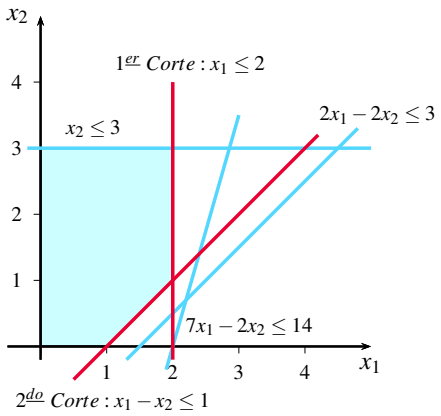
$$x_5 \geq 1$$

Como $x_5 = 3 - 2x_1 + 2x_2$, el corte fraccional de Gomory puede expresarse en el espacio (x_1, x_2), de la siguiente manera:

$$3 - 2x_1 + 2x_2 \geq 1$$
$$x_1 - x_2 \leq 1$$

Añadiendo la desigualdad en el PL y resolviéndolo, se llega a la solución óptima $(x_1, x_2) = (2, 1)$ y con valor $z = 7$.

$$\begin{aligned} \text{máx } z &= 4x_1 - 2x_2 \\ \text{Sujeto a :} \quad 7x_1 - 2x_2 &\leq 14 \\ x_2 &\leq 3 \\ 2x_1 - 2x_2 &\leq 3 \\ x_1 &\leq 2 \\ x_1 - x_2 &\leq 1 \\ x_1,\ x_2 &\geq 0 \end{aligned}$$



A pesar que el método de planos cortantes 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; éste puede ser utilizado para generar buenas cotas inferiores para el valor óptimo del problema.

Branch & Bound: Es un algoritmo de optimización genérico basado en la enumeración inteligente para llegar a una solución óptima de cualquier problema de optimización combinatoria. Fue diseñado por Dakin [1965], cuya idea se basa en dividir y conquistar: divide el problema en sub-problemas que son fáciles de resolver, mientras el óptimo no sea una solución factible entera. Su desarrollo ha sido crucial para la obtención de la solución de la mayoría de los IP.

Este método consiste en resolver la relajación lineal del IP con el fin de encontrar un óptimo x^*. Si $x^* \in \mathbb{Z}_+^n$, entonces x^* es óptimo para el IP; de lo contrario, existe algún $x_j^* \notin \mathbb{Z}_+$ y el problema se puede dividir en dos sub-problemas disjuntos: $x_j \leq \lfloor x_j^* \rfloor$ y $x_j \geq \lceil x_j^* \rceil$. Este proceso se lo aplica de una forma recursiva hasta que el sub-problema resultante tenga un óptimo entero como se describe a continuación.

Algoritmo 3: Branch & Bound

Entrada: Un programa lineal entero IP.

Salida: x_l^* una solución entera óptima del IP y z_l^* su valor objetivo.

Inicializar: $x_l^* = \emptyset$, $z_l^* = -\infty$, y $S = \{IP\}$.
si $S \neq \emptyset$: **entonces**
  Seleccionar un sub−problema IP_i de S y definir $S := S \backslash \{IP_i\}$.
  si PL_i *es no factible* **entonces**
    Regresar al paso 2.
  en otro caso
    Sea x^* una solución óptima de PL_i y z^* su valor.
    si $z^* \leq z_l^*$ **entonces**
      Regresar al paso 2.
    en otro caso
      si x^* *es entero* **entonces**
        Definimos $x_l^* := x^*$ y $z_l^* := z^*$ y se procede al paso 2.
      en otro caso
        Seleccionar una entrada fraccional x_j^* de x^* y añadir los sub−problemas a S.

$$(IP_i \cup x_i \leq \lfloor x_i^* \rfloor) y (IP_i \cup x_i \geq \lceil x_i^* \rceil)$$

        Regresar al paso 2.

El procedimiento del método puede representarse mediante un árbol de Branch & Bound como se muestra en la figura 2.2, donde cada nodo es otro problema LP a resolverse. La efectividad del algoritmo de Branch & Bound se da en la capacidad para podar nodos, es decir, omitir los sub-problemas innecesarios.

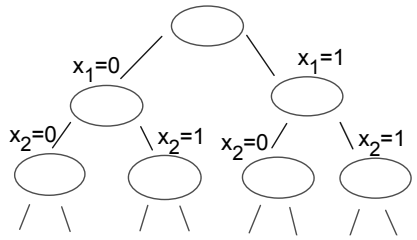


Figura 2.2: Árbol Branch & Bound para un programa lineal binario

Branch & Cut: El método de Branch & Cut es introducido por Padberg y G.Rinaldi [1989] para resolver programas enteros grandes (en donde sus variables eran binarias). Este método fue tratado en el contexto del problema del agente viajero (TSP), en donde hace uso de la teoría poliedral para encontrar fuertes planos cortantes.

Es una combinación de los dos métodos antes descritos. El esquema general de solución es generar cortes globalmente válidos (tanto para los nodos en donde se introducen, como para el problema original) en cada nodo del árbol de ramificación para ajustarse cada vez al conjunto de soluciones factibles. El método de Branch & Cut, descrito en el algoritmo 4, tiene como objetivo obtener una solución factible entera a partir del PL y obtener cotas inferiores, para una poda más efectiva.

Algoritmo 4: Branch & Cut

Entrada: Un problema de programación lineal entera IP.

Salida: x_l^* una solución entera óptima del IP y z_l^* su valor objetivo.

1 **Elección del próximo nodo:**

2 Inicializar $S = \{IP\}$, $z_l^* = +\infty$ y $x_l^* = \emptyset$

3 **si** $S = \emptyset$ **entonces**

4 x_l^* es óptimo y el algoritmo termina.

5 **en otro caso**

6 Seleccionar y borrar un nodo de S.

7 **Evaluación:**

8 Resolver el PL_i. Sea x_i^* una solución óptima de PL_i y z_i^* su valor objetivo.

Caso a) PL_i **es no factible:** ir al **paso 3**

Caso b) $z^* \leq {z_l}^*$**:** ir al **paso 3**.

Caso c) La solución óptima es entera: asignar $z_l^* = z^*$ y $x_l^* = x^*$ e ir al **paso 3**.

9 **Adición de planos cortantes:**

10 **si** *se decide buscar planos cortantes* **entonces**

11 **Separación:** Resolver el problema de separación para la solución fraccionaria de LP_i.

Caso a) Se encontraron cortes: Agregar a la formulación e ir al **paso 7**.

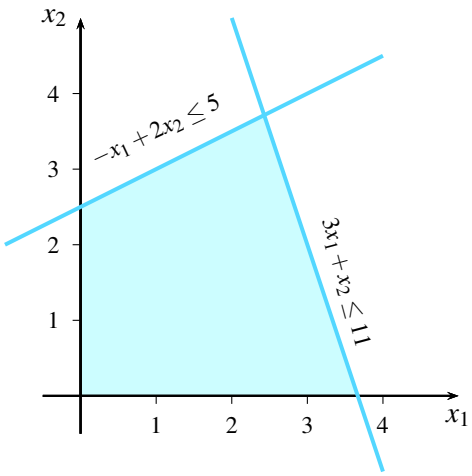
Caso b) No se encontraron planos cortantes: ir al **paso 13**.

12 **en otro caso**

13 **División:** Particionar la región factible del LP en dos o más regiones, agregando un nuevo nodo a S por cada nueva región. Ir al **paso 3**.

Para comprender el funcionamiento del algoritmo se presenta el siguiente problema de programación lineal entera junto con la región factible de su relajación:

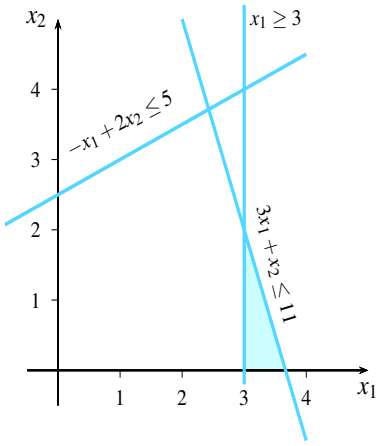
$$
\begin{aligned}
P_0 : \text{mín } z &= -6x_1 - 5x_2 \\
\text{Sujeto a :} \quad 3x_1 + \ x_2 &\leq 11 \\
-x_1 + 2x_2 &\leq 5 \\
x_1,\ x_2 &\in \mathbb{Z}_+
\end{aligned}
$$



Al momento de resolver el PL, la solución óptima es $(x_1, x_2) = (2.43, 3.71)$, con $z = -33.14$.

Tomando en cuenta la solución encontrada en P_0 y considerando que x_1 no es entero, el problema se divide en dos sub-problemas en los que se agregan nuevas restricciones según el criterio del algoritmo de Branch & Bound.

P_1**:** $min\ z = -6x_1 - 5x_2$

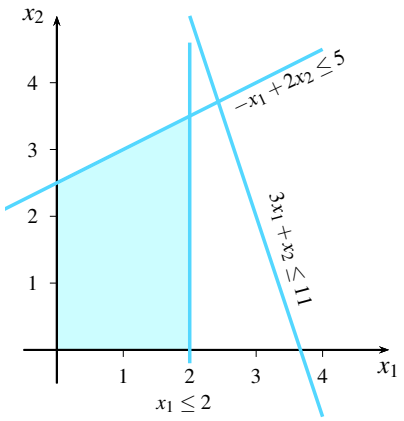


$$
\begin{aligned}
\text{Sujeto a :} \quad 3x_1 + \ x_2 &\leq 11 \\
-x_1 + 2x_2 &\leq 5 \\
x_1 \qquad\quad &\geq 3 \\
x_1,\ x_2 &\in \mathbb{Z}_+
\end{aligned}
$$

Resolviendo el problema se tiene que la solución óptima es $(x_1, x_2) = (3, 2)$ y

$z = -28$. Obteniendo así la mejor solución entera hasta el momento y una cota superior.

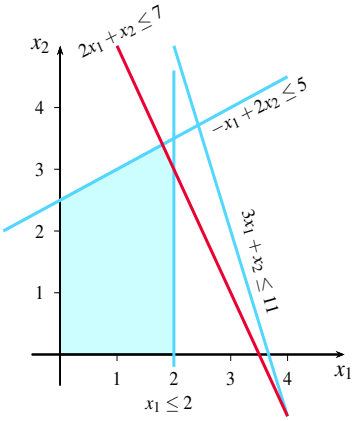
P_2: $min\ z = -6x_1 - 5x_2$



$$\text{Sujeto a :} \quad \begin{aligned} 3x_1 + \ x_2 &\leq 11 \\ -x_1 + 2x_2 &\leq 5 \\ x_1 \qquad\ \ &\geq 2 \\ x_1,\ x_2 &\in \mathbb{Z}_+ \end{aligned}$$

Donde su solución óptima es $(x_1, x_2) = (2, 3.5)$ y $z = -29.5$. Se agrega entonces el plano cortante $2x_1 + x_2 \leq 7$ a P_2 teniendo:

- P_3: $min\ z = -6x_1 - 5x_2$



$$\textit{sujeto a :} \quad \begin{aligned} 3x_1 + x_2 \quad &\leq 11 \\ -x_1 + 2x_2 \quad &\leq 5 \\ x_1 \qquad\ \ &\leq 2 \\ 2x_1 + x_2 \quad &\leq 7 \qquad \textit{Plano cortante} \\ x_1, x_2 &\in \mathbb{Z}_+ \end{aligned}$$

Resolviendo P_3 se tiene que la solución óptima es $(x_1, x_2) = (1.8, 3.4)$ y $z = -27.8$. Efectivamente $2x_1 + x_2 \leq 7$ es una desigualdad válida, ya que se satisface con cada punto que sea factible en P_3. Además, esta desigualdad es violada por $(2, 3.5)$, por lo que es un plano cortante. Pero como ya teníamos una cota $z = -28$ podemos podar esta rama y la solución óptima es $(x_1, x_2) = (3, 2)$.

En conclusión y comparando con el algoritmo clásico de Branch & Bound se tiene que:

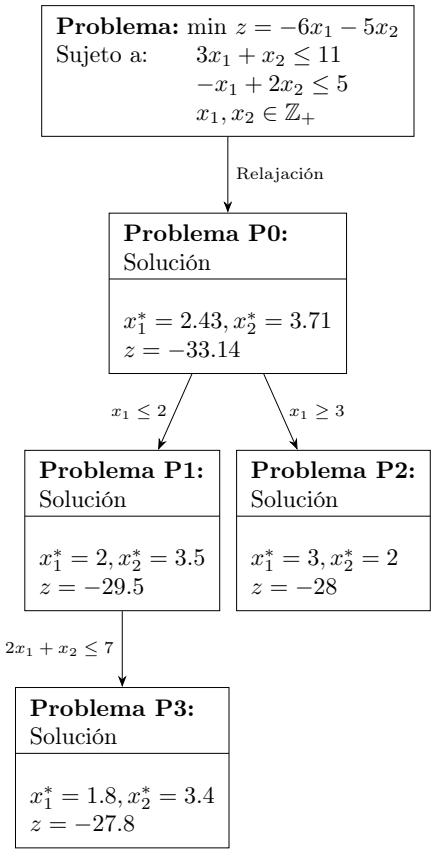


Figura 2.3: Branch & Cut

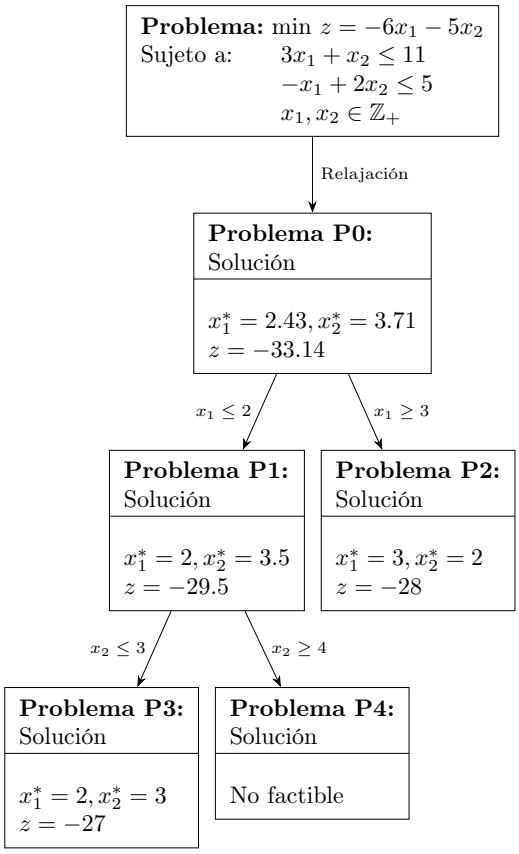


Figura 2.4: Branch & Bound

El problema resuelto con el algoritmo de Branch & Cut lo resuelve en solo 3 pasos mientras que el algoritmo Branch & Bound lo resuelve en 4 pasos. Por tanto se considera que existe una mejora al incluir planos cortantes al modelo.

2.2 Problema de particionamiento de grafos

Con las definiciones previas, formalizamos el problema de particionamiento de grafos junto con sus variaciones. Debido a que la palabra *"particionamiento"* representa la acción de crear una partición al segmentar un conjunto en varias partes, definimos lo siguiente.

DEFINICIÓN 2.17. *Sea V un conjunto cualquiera, P un conjunto de subconjuntos de V. P se llama partición de V si:*

- *Ningún elemento de P está vacío.*
- *Los elementos de P son disjuntos dos a dos.*

- *La unión de los elementos de P es igual a V.*

Los elementos de P se llaman partes de la partición P, y la cardinalidad de la partición P es por lo tanto, el número de partes de P.

Con la ayuda de la definición de partición de un conjunto, y dado que un grafo G se puede definir completamente como el par ordenado de conjunto de nodos V y el conjunto de aristas E, se extiende el concepto de la definición 2.17 a grafos.

DEFINICIÓN 2.18. *Sea $G = (V, E)$ un grafo y $P = \{V_1, \ldots, V_k\}$ un conjunto de k subconjuntos de V. P se denomina una partición de G si:*

- $V_i \neq \emptyset$, $\forall i \in \{1, \ldots, k\}$,
- $V_i \cap V_j = \emptyset$, $\forall i, j \in \{1, \ldots, k\}, i \neq j$ *y*
- $\bigcup_{i=1}^{k} V_i = V$.

En la partición de grafos, el número de partes de P a menudo se denota por k.

Además, cuando G es un grafo completo, cada partición de G es un subgrafo completo, es decir, es una ***clique****.*

DEFINICIÓN 2.19. *Un conjunto de aristas A en un grafo $G = (V, E)$ es llamado un particionamiento en cliques de G, si existe una partición $P = \{V_1, \ldots, V_k\}$ de V tal que $A = E(V_1, \ldots, V_k) := \bigcup_{i=1}^{k} E(V_i)$ y el subgrafo $G[V_i]$ inducido por V_i es una clique para $i = \{1, \ldots, k\}$.*

Una vez entendidos estos conceptos, se presenta el problema de particionamiento de grafos de la siguiente manera:

DEFINICIÓN 2.20. *Sea $k \in \mathbb{N}$ un número fijo con $k > 1$ y $G = (V, E)$ un grafo no dirigido con ponderaciones no negativas en las aristas, el problema busca una partición de G, con el fin de dividir el conjunto de nodos V en k subconjuntos que satisfagan la definición 2.18.*

Aunque algunos problemas buscan dividir las aristas de un grafo (Holyer [1981]), la partición de grafos se entiende principalmente como la partición de los nodos del grafo. A continuación se presentan algunas variaciones del problema.

Problema de partición en cliques: Este problema consiste en particionar los nodos V de un grafo completo en conjuntos de cualquier tamaño, tal que el costo total

de las aristas que tienen ambos extremos en la misma clique sea minimizado. El número de cliques k es una variable más del problema.

Esta versión ha sido estudiada en Bhasker y Samad [1990] donde se prueba una nueva generación de cota superior para el número de cliques. Además presentan dos nuevos métodos eficientes de solución, uno de los cuales es una extensión del otro. Grötschel y Wakabayashi [1989] estudiaron el problema desde el punto de vista poliedral, a partir de una formulación basada en las conocidas desigualdades triangulares. Además, un algoritmo tipo Branch & Cut es propuesto por los autores. Finalmente, Ji y Mitchell [2005] consideran el problema de partición en cliques con restricción de tamaño mínimo, como una extensión del problema de partición de clique clásico.

Por otro lado, si el número de partes k en que debe ser particionado el grafo es conocido, muchas variantes se han reportado durante los últimos años.

Problema de equipartición: Se desea particionar a V en dos conjuntos de igual tamaño, a fin de minimizar el costo total de las aristas que tienen ambos puntos finales en la misma clique. Cuando hay un número impar de nodos, deben dividirse en 2 grupos con una diferencia de tamaño 1. En Conforti *et al.* [1990] se discutió este problema dentro de un enfoque poliedral.

Maxcut Problema: El problema consiste en particionar el conjunto V en 2 conjuntos, tal que el costo total de las aristas que tienen puntos finales en diferentes subconjuntos sea maximizado. Un estudio profundo se puede encontrar en Poljak y Tuza [1995], donde se examina los resultados más importantes, técnicas típicas de prueba y algunas aplicaciones.

Problema de equipartición k-way: Particionar el conjunto V en $k > 2$ conjuntos de igual tamaño, a fin de minimizar el costo total de las aristas que tienen ambos puntos finales en la misma clique. Se considera que el número de nodos tiene que ser un múltiplo del número de particiones o cliques, es decir, cada clique tiene que ser de tamaño $\frac{n}{k}$. En Mitchell [2001] se investigó la estructura poliedral asociado a este problema y un algoritmo tipo Branch & Cut fue presentado. Usando este método se resolvió el problema de programación de la NFL en Mitchell [2003] con éxito.

Problema de partición de k-way: Se desea particionar a V en no más de k conjuntos, a fin de minimizar el costo total de las aristas que tienen ambos puntos finales en la misma partición. En Chopra y Rao [1993] se describen varias formas

del problema considerando un grafo general, proporcionando las respectivas formulaciones usando programación lineal entera, junto con la identificación de algunas facetas básicas.

Problema de partición con capacidad: Particiona a V en conjuntos que satisfacen cierta restricción de peso sobre los nodos, a fin de minimizar el costo total de las aristas que tienen puntos finales en la misma clique. En Recalde *et al.* [2016], los autores proponen una aplicación en la segunda división del fútbol ecuatoriano, donde un conjunto de equipos deben ser particionados en 4 grupos, respetando requerimientos de equidad y homogeneidad en cada grupo.

Todos los problemas de particionamiento citados pertenecen a la clase $NP-$hard, es decir, cualquier algoritmo exacto que pueda encontrar una solución factible se ejecutará en tiempo exponencial.

Capítulo 3

Problema de k-particionamiento con restricciones de tamaño y peso

En el presente capítulo nos enfocaremos en el problema de k-particionamiento con restricciones de tamaño y peso. Formalmente el problema puede ser presentado de la siguiente forma:

Dado $G = (V, E)$ un grafo completo no dirigido con $V = \{1, \ldots, n\}$ un conjunto de nodos, $E = \{\{i, j\} : i, j \in V, i \neq j\}$ conjunto de aristas, $d : E \to \mathbb{R}^+$ una función de costo sobre las aristas con $d_{ij} = d_{ji}$, $w : V \to \mathbb{R}^+$ una función de pesos sobre los nodos y un número fijo k con $2 \leq k \leq n$. Además, cotas inferiores y superiores $W_L, W_U \in \mathbb{R}^+$ con $W_L \leq W_U$, para el peso de cada clique considerada.

El problema consiste en encontrar una partición $P = \{V_1, .., V_k\}$ tal que :

$$W_L \leq \sum_{j \in V_c} w_j \leq W_U, \qquad \forall c \in \{1, \ldots, k\}, \tag{3.1}$$

$$\big||V_i| - |V_j|\big| \leq 1, \qquad \forall i, j \in \{1, \ldots, k\}, i < j \tag{3.2}$$

y el costo total de las aristas sobre todas las cliques sea minimizado. En adelante, consideramos que (3.1) representa las condiciones de peso y (3.2) a las condiciones de balance de cada clique. El problema pertenece a la clase $NP-$completo, su demostración se encuentra en Recalde *et al.* [2016].

El problema de particionamiento con restricciones de tamaño y peso ha sido estudiado por varios autores con sus respectivas modificaciones. Así, Ferreira *et al.* [1998] consideran el problema de k-particionamiento de grafos con restricciones de capacidad. El problema consiste en dividir el conjunto de nodos de un grafo tal que la suma de los pesos de los nodos dentro de cada partición sea limitada y la suma

de los costos de las aristas que tienen sus puntos finales en diferentes conjuntos de la partición sea minimizada. Se reportan familias de desigualdades válidas las que son incorporadas en un algoritmo tipo Branch & Cut, usado para resolver tres clases de problemas: problema de equi-particionamiento que surgen en los métodos de elementos finitos, problema de particionamiento asociado con el diseño del circuito electrónico y problemas derivados del diseño del compilador.

Por otra parte, Ji y Mitchell [2006] consideran el problema de particionamiento con restricción de tamaño mínimo en las cliques, donde para su solución hacen uso del método Branch & Price. El problema se lo formula mediante un modelo de programación lineal entero y se sugieren formas efectivas de agregar planos cortantes dentro del Branch & Cut. La resolución de este problema ha sido estudiado por primera vez en Mitchell [2001], siendo aplicable en el problema de realineación de equipos de la NFL en Mitchell [2003].

Finalmente Labbé y Ozsoy [2010] consideran un problema de particionamiento de grafos con restricción de tamaño, donde cotas superiores e inferiores asociadas a la cardinalidad de las particiones son incluidas. Para resolver el problema, los autores proponen un modelo de programación lineal entera junto con la implementación de un algoritmo de tipo Branch & Cut. Además, se hace un análisis de la estructura del politopo correspondiente a los problemas y se desprenden resultados importantes para el problema de equipartición y el problema de equipartición k-way. Este trabajo generaliza varias versiones del problema de particionamiento de grafos conocidos dentro de la literatura, como es el caso del problema de partición en cliques, el problema de equipartición y el problema de equipartición k-way.

Existen varios métodos que resuelven el problema de particionamiento de grafos, sin embargo, la utilización del enfoque en la programación lineal entera se ha usado ampliamente para su solución exacta en muchos trabajos previos como Brunetta *et al.* [1997], Ferreira *et al.* [1998], Lisser y Rendl [2003], Armbruster *et al.* [2008] y Catanzaro *et al.* [2011].

3.1 Formulación

A continuación se describe dos modelos de programación lineal entera para el problema de k-particionamiento con restricciones de tamaño y peso. Ambos modelos consideran las siguientes variables binarias:

$$y_i^c = \begin{cases} 1 & \text{si el nodo } i \in V \text{ pertenece a la clique } c \\ 0 & \text{caso contrario} \end{cases}, \qquad \forall c \in \{1, \ldots, k\}$$

$$x_{ij}^c = \begin{cases} 1 & \text{si la arista } \{i,j\} \in E \text{ pertenece a la clique } c \\ 0 & \text{caso contrario} \end{cases}, \qquad \forall c \in \{1, \ldots, k\}$$

3.1.1 Primer modelo

Este modelo se encuentra propuesto en Recalde *et al.* [2016] como una aplicación a la segunda categoría del fútbol ecuatoriano. Se encuentra formulado de la siguiente manera ($\mathcal{F}_1$):

$$\text{mín} \sum_{c=1}^{k} \sum_{\{i,j\} \in E} d_{ij} x_{ij}^c \tag{3.3}$$

Sujeto a:

$$\sum_{i \in V} y_i^c = \frac{n}{k}, \qquad \forall c = 1, \ldots, k \tag{3.4}$$

$$\sum_{c=1}^{k} y_i^c = 1, \qquad \forall i \in V \tag{3.5}$$

$$\sum_{i \in A} y_i^c \leq 1, \qquad \forall c = 1, \ldots, k \tag{3.6}$$

$$W_L \leq \sum_{i \in V} w_i y_i^c \leq W_U, \qquad \forall c = 1, \ldots, k \tag{3.7}$$

$$\sum_{j \in \delta(i)} x_{ij}^c = \left(\frac{n}{k} - 1\right) y_i^c, \qquad \forall i \in V, \forall c = 1, \ldots, k \tag{3.8}$$

$$y_i^c, x_{ij}^c \in \{0, 1\}, \qquad \forall i \in V, \forall c = 1, \ldots, k \text{ y } \{i,j\} \in E \tag{3.9}$$

donde $\delta(i)$ es el conjunto de las aristas incidentes al nodo $i \in V$ y A es un conjunto de nodos ficticios. Los autores proponen la inclusión de nodos artificiales o ficticios en los casos en que n no sea múltiplo de k. Así, si $n \, mod \, k \neq 0$, entonces se incluye $k - (n \, mod \, k)$ nodos ficticios en A con $w_i = 0$ y el nuevo conjunto de nodos es definido por $V_A = V \cup A$. La inclusión del conjunto A en el conjunto V implica que $n \, mod \, k = 0$. El conjunto de aristas es ahora definido por $E_A = E \cup \{\{i,j\} : i \in A, j \in V \cup A, i \neq j\}$, donde la función de costos sobre las aristas ficticias es de valor cero.

La función objetivo (3.3) busca minimizar el costo total de las aristas dentro de

las cliques; las restricciones (3.4) nos garantiza la construcción de cliques de igual tamaño; (3.5) aseguran que cada nodo sea asignado exactamente a una sola clique; (3.6) garantizan la existencia de un nodo ficticio como máximo en cada clique; (3.7) imponen el requisito de peso en cada clique permitiendo la construcción de cliques homogéneas; (3.8) establece que si un nodo pertenece a una clique, entonces dicho nodo debe estar conectado con los $\frac{n}{k} - 1$ nodos en la clique. Finalmente (3.9) define que todas las variables son binarias. Además, se puede observar que en la formulación proporcionada se presentan exactamente $k\left[\frac{n'(n'-1)}{2}\right] + kn'$ variables y $k(n'+3) + n'$ restricciones, donde $n' = |V_A|$.

3.1.2 Segundo Modelo

Por otro lado, para evitar el uso del conjunto de nodos ficticios, el siguiente modelo considera construir cliques con un número mínimo de nodos igual a $\left\lfloor \frac{n}{k} \right\rfloor$ y máximo $\left\lceil \frac{n}{k} \right\rceil$ nodos. Puede ser formulado de la siguiente manera ($\mathcal{F}_2$):

$$\text{mín} \sum_{c=1}^{k} \sum_{\{i,j\} \in E} d_{ij} x_{ij}^c \tag{3.10}$$

Sujeto a:

$$x_{ij}^c + x_{jk}^c - x_{ik}^c \leq 1, \qquad \forall\, 1 \leq i < j < k \leq n, \forall c = 1, \ldots, k \tag{3.11}$$

$$x_{ij}^c - x_{jk}^c + x_{ik}^c \leq 1, \qquad \forall\, 1 \leq i < j < k \leq n, \forall c = 1, \ldots, k \tag{3.12}$$

$$-x_{ij}^c + x_{jk}^c + x_{ik}^c \leq 1, \qquad \forall\, 1 \leq i < j < k \leq n, \forall c = 1, \ldots, k \tag{3.13}$$

$$\sum_{c=1}^{k} y_i^c = 1, \qquad \forall i \in V \tag{3.14}$$

$$\sum_{j \in \delta(i)} x_{ij}^c \geq \left(\left\lfloor \frac{n}{k} \right\rfloor - 1 \right) y_i^c, \qquad \forall i \in V, \forall c = 1, \ldots, k \tag{3.15}$$

$$\sum_{j \in \delta(i)} x_{ij}^c \leq \left(\left\lceil \frac{n}{k} \right\rceil - 1 \right) y_i^c, \qquad \forall i \in V, \forall c = 1, \ldots, k \tag{3.16}$$

$$W_L \leq \sum_{i \in V} w_i y_i^c \leq W_U, \qquad \forall c = 1, \ldots, k \tag{3.17}$$

$$y_i^c, x_{ij}^c \in \{0, 1\}, \qquad \forall i \in V, \forall c = 1, \ldots, k \text{ y } \{i, j\} \in E \tag{3.18}$$

La función objetivo (3.10) busca minimizar el costo total de las aristas en las k cliques; las restricciones (3.11), (3.12) y (3.13) corresponden a las desigualdades triangulares, que garantizan que si tres nodos de V están unidos por dos aristas en la cli-

que c, entonces la tercer arista también pertenece a la misma clique c, $\forall\ c = 1, \ldots, k$; la restricción (3.14) asegura que cada nodo sea asignado exactamente a una clique; las restricciones (3.15) y (3.16) son restricciones de acoplamiento y aseguran que la cardinalidad de las cliques difieran a lo máximo en uno; (3.17) imponen el requisito de peso en cada clique. Finalmente, (3.18) determinan que todas las variables son binarias. Una condición necesaria sobre esta formulación es que el número mínimo de nodos en cada clique sea al menos igual a 3. De la misma manera, se puede notar que en la formulación existen $k\left[\frac{n(n-1)}{2}\right] + kn$ variables y $k\left[3\binom{n}{3} + 2n + 1\right] + n$ restricciones.

Capítulo 4

Planos cortantes para el modelo de particionamiento

En el presente capítulo se describirán tres clases de planos cortantes que serán incluidos en los modelos $\mathcal{F}_1$ y $\mathcal{F}_2$. El presente trabajo usa y adapta algunos resultados reportados en el paper de Grötschel y Wakabayashi [1990], el cual describe el problema de particionamiento en cliques de la siguiente forma: Dado un grafo completo $K_n = (V_n, E_n)$ con pesos $w_e \in \mathbb{R}$ para todo $e \in E_n$, la tarea consiste en encontrar un particionamiento de E_n en cliques de peso mínimo.

Cabe mencionar que el problema de particionamiento en cliques puede ser considerado para grafos generales $G = (V, E)$, puesto que estos pueden convertirse en grafos completos al agregar aristas faltantes con costos de valor cero.

El problema de particionamiento en cliques puede ser enfocado desde un punto de vista poliedral, y para ello lo asociamos con un poliedro denotado por

$$\mathcal{P}_n = conv\{\mathcal{X}^A \in \mathbb{R}_n^E : \text{A es un particionamiento de } K_n\}$$

donde $\mathcal{X}^A$ denota el vector de incidencia de $A \subseteq E$, es decir,

$$\mathcal{X}_e^A = \begin{cases} 1 & \text{si } e \in A \\ 0 & \text{caso contrario} \end{cases},$$

$\mathcal{P}_n$ representa la envolvente convexa de todos los vectores de incidencia de particiones de clique y es comúnmente denominado el politopo de particionamiento en cliques. Grötschel y Wakabayashi [1990] determinan que las desigualdades triangulares y 2-partición son facetas para $\mathcal{P}_n$. Previo a sus demostraciones, se presentan la descripción respectiva de cada desigualdad, así como también las modificaciones

realizadas para los modelos $\mathcal{F}_1$ y $\mathcal{F}_2$ tratados en este proyecto.

4.1 Desigualdades triangulares

Las desigualdades triangulares son desigualdades lineales que se encuentran definidas para tres nodos diferentes $i, j, k \in V$ de la siguiente manera:

$\forall\ 1 \leq i < j < k \leq n,$

$$x_{ij} + x_{jk} - x_{ik} \leq 1$$

$$x_{ij} - x_{jk} + x_{ik} \leq 1$$

$$-x_{ij} + x_{jk} + x_{ik} \leq 1$$

Éstas garantizan que si tres nodos $i, j, k \in V$ están unidos por 2 aristas $\{i, j\}$ y $\{j, k\}$, entonces la tercer arista $\{i, k\}$ también debe estar en la solución. Sin perdida de generalidad, se considera la siguiente desigualdad como representación de las tres combinaciones a formarse de las desigualdades triangulares:

$$x_a + x_b - x_c \leq 1 \qquad \forall a, b \text{ y } c \in E$$

Grötschel y Wakabayashi [1990] demuestran que éstas desigualdades son facetas para $\mathcal{P}_n$, como se observa en el siguiente teorema.

TEOREMA 4.1. *(Grötschel y Wakabayashi [1990],Teorema 3.1.) Para todo politopo de particionamiento en cliques, $\mathcal{P}_n$, $n \geq 3$, se tiene los siguientes resultados:*

a) Toda restricción no negativa $x_e \geq 0$ define una faceta de $\mathcal{P}_n$.

b) Toda desigualdad triangular de la forma $x_a + x_b - x_c \leq 1$ define una faceta de $\mathcal{P}_n$.

c) Ninguna desigualdad de cota superior $x_e \leq 1$ define una faceta de $\mathcal{P}_n$.

Demostración.

a) Sea $e \in E_n$. Entonces $x_e = 0$ es satisfecho por el vector cero y todo vector unitario $\mathcal{X}^{\{f\}}$, $f \in E_n$, $f \neq e$. Se tiene que los vectores $|E_n|$ son vectores de incidencia del particionamiento en cliques y son afines independientes.

b) Sea $\{a, b, c\}$ un triángulo de K_n. Notemos $a := \{i, j\}$, $b := \{j, k\}$, $c := \{i, k\}$. Entonces, los vectores de incidencia $|E_n|$ del particionamiento en cliques son

los siguientes:

$$\begin{array}{ccc} \{a\}, & \{b\}, & \{a,b,c\} \\ \{a,e\} & \forall e \in E_n, & e \notin \delta(i) \cup \delta(j) \\ \{b,e\} & \forall e \in E_n, & e \in \delta(i) \backslash \{a,c\}, \\ E(\{i,j,k,z\}) & \forall z \in V \backslash \{i,j,k\} & . \end{array}$$

Que satisface $x_a + x_b - x_c \leq 1$ y son linealmente independientes

c) Sean $e, f, g \in E_n$ un triángulo. Entonces $2x_e \leq 2$ es la suma de dos desigualdades triangulares las cuales definen facetas: $x_e + x_f - x_g \leq 1$ y $x_e - x_f + x_g \leq 1$, por lo tanto no define una faceta de $\mathcal{P}_n$.

□

Considerando el resultado del teorema 4.1 y fijando un nuevo índice c, se puede observar fácilmente que las siguientes desigualdades son válidas para las formulaciones $\mathcal{F}_1$ y $\mathcal{F}_2$ y, por lo tanto, se pueden usar en el algoritmo de Branch & Cut. Así, para todo $c \in \{1, \ldots k\}$ y para todo $1 \leq i < j < k \leq n$, se tiene

$$\begin{aligned} x^c_{ij} + x^c_{jk} - x^c_{ik} &\leq 1 \\ x^c_{ij} - x^c_{jk} + x^c_{ik} &\leq 1 \\ -x^c_{ij} + x^c_{jk} + x^c_{ik} &\leq 1 \end{aligned} \tag{4.1}$$

Recordando que x^c_{ij} fue definido en el capitulo 3 de la siguiente manera:

$$x^c_{ij} = \begin{cases} 1 & \text{si } \{i,j\} \in E \text{ pertenece a la clique c} \\ 0 & \text{caso contrario} \end{cases}, \qquad \forall c = \{1, .., k\}$$

En este caso, las desigualdades establecen que para todo $i, j, k \in V$, si las aristas $\{i,j\}$ y $\{j,k\}$ están en la clique c, la arista $\{i,k\}$ también está en la misma clique c.

En la formulación $\mathcal{F}_1$ estas desigualdades serán ingresadas como planos cortantes. Así, para cada clique c, se generarán $3\binom{n}{3}$ desigualdades y se ingresarán al modelo aquellas que mayormente sean violadas. Esta rutina puede ser descrita bajo

el siguiente pseudocódigo.

Algoritmo 5: Rutina de inclusión de desigualdades triangulares

Entrada: Sea x^* la solución óptima del PL de $\mathcal{F}_1$ y $\alpha > 0$.

```
1 Generar todas las desigualdades triangulares.
2 para cada c = 1 hasta k hacer
3     para cada desigualdad triangular a_c hacer
4         Calcular el grado de violación δ_ac = a_c^T x* − 1.
5         si δ_ac ≥ 1 − α entonces
6             Agregar a F_1 la desigualdad a_c.
```

Para la formulación $\mathcal{F}_2$, las desigualdades triangulares son parte del modelo (restricciones (3.11) - (3.13)). Debido al gran número de restricciones, dichas desigualdades triangulares serán ingresadas como restricciones tipo lazy y como planos cortantes. Así, iniciaremos con el modelo $\mathcal{F}_2-$Reducido ($\mathcal{F}_2 - \mathcal{R}$).

$$\text{mín} \sum_{c=1}^{k} \sum_{\{i,j\} \in E} d_{ij} x_{ij}^c$$

sujeto a

$$\sum_{c=1}^{k} y_i^c = 1, \qquad \forall i \in V \tag{4.2}$$

$$\sum_{j \in \delta(i)} x_{ij}^c \geq \left(\left\lfloor \frac{n}{k} \right\rfloor - 1 \right) y_i^c, \qquad \forall i \in V, \forall c = 1, \ldots, k \tag{4.3}$$

$$\sum_{j \in \delta(i)} x_{ij}^c \leq \left(\left\lceil \frac{n}{k} \right\rceil - 1 \right) y_i^c, \qquad \forall i \in V, \forall c = 1, \ldots, k \tag{4.4}$$

$$W_L \leq \sum_{i \in V} w_i y_i^c \leq W_U, \qquad \forall c = 1, \ldots, k \tag{4.5}$$

$$y_i^c, x_{ij}^c \in \{0, 1\}, \qquad \forall i \in V, \forall c = 1, \ldots, k \text{ y } \{i, j\} \in E \tag{4.6}$$

En cada nodo del árbol de ramificación se obtiene una solución fraccionaria x^* y se aplica el algoritmo 5 de inclusión de las desigualdades triangulares. Además si una solución entera es encontrada, entonces se verifica si las desigualdades triangulares han sido respetadas. Si no es el caso, entonces son incluidas en el modelo $\mathcal{F}_2 - \mathcal{R}$

usando el siguiente algoritmo de inclusión de restricciones tipo *Lazy*:

Algoritmo 6: Rutina 2 de inclusión de desigualdades triangulares

Entrada: Sea $\bar{x}$ solución entera de la formulación $\mathcal{F}_2 - \mathcal{R}$.

```
1 Generar todas las desigualdades triangulares.
2 para cada c = 1 hasta k hacer
3     para cada desigualdad triangular a_c hacer
4         si a_c^T x̄ ≥ 1. entonces
5             Ingresar la restricción a_c^T x̄ ≤ 1 al modelo F_2 − R.
```

4.2 Desigualdades 2-Partición

Sea $K_n = (V_n, E_n)$ un grafo completo, S y T dos subconjuntos disjuntos no vacíos de V_n, la desigualdad

$$x([S:T]) - x(E_n(S)) - x(E_n(T)) \leq \text{mín}\{|S|, |T|\}$$

se denomina 2-partición, o desigualdad de $2-$partición inducida por S y T (desigualdad-$[S:T]$), donde $[S:T] = \{\{u,v\} : u \in S, v \in T\}$. Esta clase de desigualdad es una generalización de las desigualdades triangulares, puesto que si $|S| = 1$ y $|T| = 2$, entonces la desigualdad correspondiente no es más que una desigualdad triangular. Por ejemplo, sea $S = \{i\}$ y $T = \{j,k\}$, entonces se tiene la desigualdad

$$x([\{i\} : \{j,k\}]) - x(E(\{i\})) - x(E(\{j,k\})) \leq min\{1,2\},$$

que es equivalente a la desigualdad

$$x_{ij} + x_{ik} - x_{jk} \leq 1$$

En la figura 4.1 se puede observar un ejemplo de esta desigualdad tomando $|S| = 2$ y $|T| = 3$, donde su desigualdad 2-partición se encuentra definida de la siguiente forma:

$$x([\{u,v\} : \{t,y,z\}]) - x(E[\{u,v\}]) - x(E[\{t,y,z\}]) \leq 2$$

y cuya expansión se encuentra descrita de la siguiente manera:

$$x_{ut} + x_{uy} + x_{uz} + x_{vt} + x_{vy} + x_{vz} - x_{uv} - x_{ty} - x_{tz} - x_{yz} \leq 2$$

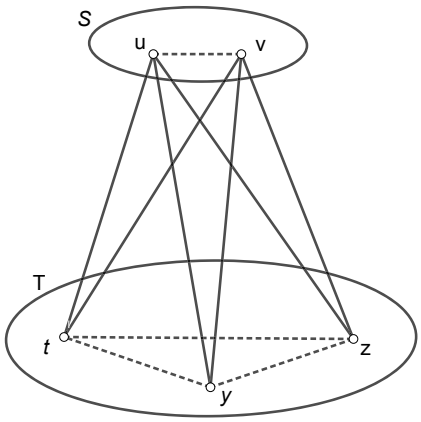


Figura 4.1: Grafo de una desigualdad 2-partición con $S = \{u, v\}$ y $T = \{t, y, z\}$

Como se mencionó anteriormente, Grötschel y Wakabayashi [1990] también determinan que las desigualdades 2-partición son facetas para $\mathcal{P}_n$. Previo a su demostración se considera el siguiente concepto y teorema.

DEFINICIÓN 4.1. *Un emparejamiento M en un grafo $G = (V, E)$ es un conjunto de aristas tal que no existen dos aristas de M que tengan un nodo final común.*

OBSERVACIÓN 1. Si $|M| = p$ entonces decimos que M es un $p - emparejamiento$.

OBSERVACIÓN 2. Si un nodo v es el nodo final de una arista de un emparejamiento M, entonces decimos que v está cubierto por M o M cubre v.

TEOREMA 4.2. *(Grötschel y Wakabayashi [1990],Teorema 3.3 (Teorema de elevación)) Supongamos que $\sum_{e \in E_n} a_e x_e \leq \alpha$ define una faceta no trivial de $\mathcal{P}_k$. Entonces, esta desigualdad también define una faceta de $\mathcal{P}_n$, para todo $n > k$, siempre que se cumpla la siguiente condición:*

(L) Existe un particionamiento en cliques $A = E_k(W_1, .., W_s)$ de K_k y un nodo $v \in V_k$ tal que $\sum_{e \in E_n} a_e \mathcal{X}_e^A = \alpha$ y $\{v\} = W_i$, para algún $i \in \{1, .., s\}$.

La demostración detallada del teorema 4.2 se lo puede encontrar en Grötschel y Wakabayashi [1990]. Con este resultado, se da paso a la demostración del teorema principal.

TEOREMA 4.3. *(Grötschel y Wakabayashi [1990],Teorema 4.1.) Para cada $n \geq 3$ y dos subconjuntos disjuntos no vacíos S y T de V_n, la desigualdad correspondiente de 2-partición*

$$x([S : T]) - x(E_n(S)) - x(E_n(T)) \leq min\{|S|, |T|\} \tag{4.7}$$

es válida para $\mathcal{P}_n$. Esta desigualdad define una faceta si y solo si $|S| \neq |T|$.

Demostración. Sin pérdida de generalidad supongamos que $|S| \leq |T|$. Probaremos la validez de 4.7 por inducción en $|S| + |T|$. Sea $|S| = 1$ y $|T| \geq 1$. Para $|T| \leq 2$ el resultado es evidente. Así que supongamos que $|T| = t \geq 3$. Por hipótesis de inducción, para cada $v \in T$ la desigualdad-$[S : T\backslash\{v\}]$

$$x([S : T\backslash\{v\}]) - x(E_n(T\backslash\{v\})) \leq 1$$

es válido para $\mathcal{P}_n$. Añadiendo estas desigualdades para todo $v \in T$ obtenemos

$$(t-1)(x([S : T])) - (t-2)\,(x(E_n(T))) \leq t$$

Como $-x(E_n(T)) \leq 0$ también es válido para $\mathcal{P}_n$, agregando esta desigualdad a la anterior, obtenemos

$$(t-1)(x([S : T]) - x(E_n(T))) \leq t,$$

por lo tanto

$$x([S : T]) - x(E_n(T)) \leq \frac{t}{t-1},$$

lo que implica que

$$x([S : T]) - x(E_n(T)) \leq \left\lfloor \frac{t}{t-1} \right\rfloor = 1,$$

es válido para $\mathcal{P}_n$.

Ahora, sea $|S| = s \geq 2$, $|T| = t \geq 2$, $|S| + |T| = k$, y supongamos que 4.7 es válido para $|S| + |T| \leq k - 1$.

Para cada $v \in S$ se considera la desigualdad-$[S\backslash\{v\} : T]$

$$x([S\backslash\{v\} : T]) - x\,(E_n(S\backslash\{v\})) - x\,(E_n(T)) \leq s - 1 \tag{4.8}$$

y para cada $v \in T$, considere la desigualdad-$[S : T\backslash\{v\}]$,

$$x([S : T\backslash\{v\}]) - x(E_n(S)) - x(E_n(T\backslash\{v\})) \leq \text{mín}\{s, t-1\} \tag{4.9}$$

Por hipótesis de inducción, todas estas desigualdades son válidas para $\mathcal{P}_n$. Sumando las desigualdades (4.8) para cada $v \in S$ y (4.9) para cada $v \in T$ obtenemos

$$(s+t-2)(x([S : T]) - x(E_n(S)) - x(E_n(T))) \leq s(s-1) + t(min\{s, t-1\}) \tag{4.10}$$

Si $|S| < |T|$, entonces (4.10) implica

$$x([S:T]) - x(E_n(S)) - x(E_n(T)) \leq \left\lfloor \frac{s(s+t-1)}{s+t-2} \right\rfloor = |S|.$$

Si $|S| = |T|$, es decir, $s = t$, entonces (4.10) puede escribirse como

$$(2s-2)(x([S:T]) - x(E_n(S)) - x(E_n(T))) \leq s(2s-2),$$

lo que implica que

$$x([S:T]) - x(E_n(S)) - x(E_n(T)) \leq |S|$$

Esto completa la prueba de que la desigualdad (4.7) es válida para $\mathcal{P}_n$. Cuando $|S| = |T|$ la demostración dada anteriormente muestra que la desigualdad (4.7) puede obtenerse mediante una combinación lineal no negativa de otras desigualdades válidas, y por lo tanto no define una faceta de $\mathcal{P}_n$.

Ahora suponga que $|S| < |T|$. Primero se demostrará que (4.7) define una faceta de $\mathcal{P}_k$, donde $k = |S| + |T|$.

Por comodidad de la notación, podemos suponer que $S := \{1, 2, .., s\}$. Denotamos $a^T x \leq a_0$ como la desigualdad (4.7), es decir,

$$a^T x := x([S:T]) - x(E_n(S)) - x(E_n(T)) \leq s =: a_0$$

y sea $b^T x \leq b_0$ una desigualdad que define una faceta para $\mathcal{P}_k$ tal que

$$F_a := \{x \in \mathcal{P}_k | a^T x = a_0\} \subseteq F_b := \{x \in \mathcal{P}_k | b^T x = b_0\}.$$

Claramente, $F_a \neq \mathcal{P}_k$, así si podemos demostrar que $b = \alpha a$ para algunos $\alpha \in \mathbb{R}$ entonces ya que $F_a \neq \emptyset$ podemos concluir que (4.7) define una faceta de $\mathcal{P}_k$. Comenzamos estableciendo lo siguiente:

OBSERVACIÓN 3. Existe un $\alpha \in \mathbb{R}$ tal que $b_e = \alpha$ para todo $e \in [S:T]$

Demostración. Para demostrar la observación 3, consideramos un subconjunto $T' \subset T$ con $|T'| = s$, y sea i un nodo en S. Para cada nodo $w \in T'$, definimos $M_i(w)$ un s-emparejamiento que contenga $\{i, w\}$ con $M_i(w) \subseteq [S : T']$; y para cada par $\{w, v\}$ con $w \in T'$ y $v \in T \setminus T'$, definimos

$$M_i(w, v) := (M_i(W) \setminus \{i, w\}) \cup \{i, v\}.$$

Está claro que $\mathcal{X}^{M_i(w)}, \mathcal{X}^{M_i(w,v)} \in F_a \subseteq F_b$, y por lo tanto

$$0 = b_0 - b_0 = b^T\mathcal{X}^{M_i(w)} - b^T\mathcal{X}^{M_i(w,v)} = b_{iw} - b_{iv}.$$

Entonces se tiene que para un nodo (fijo) $i \in S$ y para cada $w \in T^{'}$, se tiene que $b_{iw} = b_{iv}$ para cada $v \in T \setminus T^{'}$. Esto implica que por todo $i \in S$ existe un $\alpha_i \in \mathbb{R}$ tal que $b_{it} = \alpha_i$ para todo $t \in T$.

Para completar la prueba de la observación 3, demostraremos que para $s \geq 2$, $\alpha_1 = \cdots = \alpha_s$. Para $i, j \in S$, $i \neq j$, sea M un $(s-2)$-emparejamiento contenida en $[S \setminus \{i,j\} : T]$. Si $s = 2$, tomamos $M = \emptyset$. Sean u, v y w nodos distintos en T, no cubiertos por M-emparejamiento. Sea

$$A := M \cup \{\{i,u\},\{i,v\},\{u,v\}\} \cup \{j,w\} \qquad y \qquad B := M \cup \{\{j,u\},\{j,v\},\{u,v\}\} \cup \{i,w\}$$

Como $\mathcal{X}^A, \mathcal{X}^B \in F_a \subseteq F_b$, se sigue que $b_{iu} + b_{iv} + b_{jw} = b_{ju} + b_{jv} + b_{iw}$, lo que implica que $\alpha_i = \alpha_j$. Comprobando la observación 3. □

Nuestro siguiente paso es probar que:

OBSERVACIÓN 4. $b_e = -\alpha$ para todo $e \in E_n(S) \cup E_n(T)$.

Demostración. Sea $e := \{u,v\} \in E_n(T)$ y sea M un s-emparejamiento cubierto por v pero no u, $M \subset [S : T]$. Sea $i \in S$ tal que $\{i,v\} \in M$ y sea

$$A_e := M \cup \{\{i,u\},\{u,v\}\}$$

Claramente $\mathcal{X}^M, \mathcal{X}^{A_e} \in F_a \subseteq F_b$, y por lo tanto $b_{iu} + b_{uv} = 0$, es decir,

$$b_e = -b_{uv} = -b_{iu} = -\alpha$$

Ahora, sea $e := \{i,j\} \in E_n(S)$ y sea M un s-emparejamiento contenida en $[S : T]$. Sean u, v nodos de T tales que $\{i,u\}, \{j,v\} \in M$ y sea

$$B_e := M \cup \{\{i,v\},\{i,j\},\{j,u\},\{u,v\}\}$$

Ya que $\mathcal{X}^M, \mathcal{X}^B \in F_a \subseteq F_b$, se deduce que $b_{iv} + b_{ij} + b_{ju} + b_{uv} = 0$. Por los resultados anteriores se tiene que $b_{iv} = b_{ju} = \alpha = -b_{uv}$. Por lo tanto, $b_e = b_{ij} = -\alpha$, y esto comprueba la demostración de la observación 4. □

Las dos observaciones implican que $b_e = \alpha a_e$ para todo $e \in E_k$, y esto demuestra que (4.7) define una faceta para $\mathcal{P}_k$.

Para cualquier nodo $v \in T$ existe $M \subseteq [S : T]$ un s-emparejamiento que no cubre v. M es una partición de clique y v un nodo como se requiere en la condición (L). Así, mediante el teorema 4.3, la desigualdad $[S : T]$ define una faceta de $\mathcal{P}_n$ para todo $n \geq k$. □

Al igual que con las desigualdades triangulares, el resultado anterior puede ser usado en las formulaciones $\mathcal{F}_1$ y $\mathcal{F}_2$. Así, para algún índice c y dos conjuntos distintos no vacíos $S, T \in V$, la desigualdad 2-partición

$$x^c([S : T]) - x^c(E_n(S)) - x^c(E_n(T)) \leq min\{|S|, |T|\}$$

es una desigualdad válida para las formulaciones $\mathcal{F}_1$ y $\mathcal{F}_2$.

Usando las ideas descritas en Grötschel y Wakabayashi [1989] podemos introducir 2 heurísticas de separación para implementar dichas desigualdades:

a) Para cada $c = 1, \ldots, k$ y para cada nodo $v \in V_n$, la rutina identifica todos los nodos w con x^c_{vw} fraccionarios y los incluye en W. El conjunto S se forma de un solo nodo $S = \{v\}$ y el conjunto T esta conformado por todos los nodos i que satisfacen ${x^c_{ij}}^* = 0$, $i \in W \backslash \{w\}$ y $j \in T$. La rutina escrita en pseudocódigo se presenta a continuación:

Algoritmo 7: Primera heurística de separación 2-partición

```
1 para c = 1 hasta k hacer
2     para cada v ∈ V hacer
3         Establecer W := {w ∈ V_n\{v} : 0 < x^c_vw* < 1}
4         Definir T := {w}, donde w es el primer nodo de W.
5         para cada i ∈ W\{w} hacer
6             T := T ∪ {i} si x^c_ij* = 0 para todo j ∈ T.
7         si x^c*([{v} : T]) − x^c*(E(T)) > 1 entonces
8             Se agrega la desigualdad x^c*([{v} : T]) − x^c*(E(T)) ≤ 1.
9         Realizar el paso (5) con el orden opuesto.
```

b) La segunda heurística de separación difiere de la primera en la definición del conjunto T, como se puede observar:

Algoritmo 8: Segunda heurística de separación 2-partición

para $c = 1$ *hasta* k **hacer**
  para *cada* $v \in V_n$ **hacer**
    Establecer $W := \{w \in V_n \backslash \{v\} : 0 < {x^c_{vw}}^* < 1\}$
    Definir $T := \{w\}$, dónde w es el primer nodo de W
    para *cada* $i \in W \backslash \{w\}$ **hacer**
      $T := T \cup \{i\}$ si ${x^c_{iv}}^* - \sum {x^c_{ij}}^* > 0$ para todo $j \in T$.
    si $x^{c*}([\{v\} : T]) - x^{c*}(E(T)) > 1$ **entonces**
      Se agrega la desigualdad $x^{c*}([\{v\} : T]) - x^{c*}(E(T)) \leq 1$.
    Realizar el paso (5) con el orden opuesto.

En el presente trabajo la implementación de las rutinas de separación de las desigualdades 2-partición usan el primer algoritmo propuesto.

4.3 Desigualdades Cover

Las desigualdades cover o de cubrimiento fueron definidas originalmente en el contexto del problema de la mochila $0-1$, definido de la siguiente forma:

$$\begin{aligned} \text{máx} \quad & p^T x \\ sujeto\ a: \quad & w^T x \leq c \\ & x \in \{0,1\}^n, \end{aligned}$$

donde $p \in \mathbb{Z}^n_+$ es un vector de beneficio obtenido por ingresar un producto en la mochila, $w \in \mathbb{Z}^n_+$ es el vector de pesos de los productos y $c \in \mathbb{Z}_+$ es la capacidad de la mochila.

Para construir desigualdades válidas de tipo cover, se toma un subconjunto de objetos, tal que su peso supere la capacidad de la mochila. Esto implica que el subconjunto seleccionado no puede ser una solución factible y consecuentemente al menos 1 objeto debe ser desechado de la selección anterior para poder construir una solución factible . Así, sea $N = \{1, \ldots, n\}$ el conjunto de objetos y $C \subseteq N$ un subconjunto tal que $\sum_{i \in C} w_i > c$. Luego, se tiene que la desigualdad cover se encuentra definida de la siguiente manera:

$$\sum_{j \in C} x_j \leq |C| - 1$$

Esta desigualdad es válida para el politopo correspondiente al problema, definido

de la siguiente manera:

$$conv\{x \in \{0,1\}^n : w^T x \leq c\},$$

que representa a la envolvente convexa de todos los vectores $0/1$ que satisfacen la desigualdad de la mochila $\sum_{J \in N} w_j x_i \leq c$. Para tener una mejor idea de esta desigualdad se presenta el siguiente ejemplo. Supongamos que se dispone de 8 objetos y que la restricción de mochila toma la forma:

$$4x_1 + 6x_2 + 6x_3 + 6x_4 + 6x_5 + 7x_6 + 8x_7 + 15x_8 \leq 22.$$

El conjunto $\{1,2,6,7\}$ forma un cover, ya que $4+6+7+8 = 25 > 22$. La desigualdad cover correspondiente toma la forma $x_1 + x_2 + x_6 + x_7 \leq 3$.

Usando estas ideas, nuevas desigualdades válidas se derivan para los pesos de los cliques. Dado que las variables y_i^c de las formulaciones $\mathcal{F}_1$ y $\mathcal{F}_2$ están relacionados a la formación de cliques homogéneas, es decir, se debe crear conjuntos cuyo peso se encuentra acotado superior e inferiormente, podemos definir desigualdades válidas tipo cover.

Así, sea $S \subseteq V$ un subconjunto de nodos. Diremos que S es un cubrimiento si $\sum_{i \in S} w_i > W_u$. Además, se dice que S es un cubrimiento minimal si $\sum_{i \in S} w_i > W_u$ y $\sum_{i \in S \setminus \{j\}} w_i < W_u$, $\forall j \in S$. Iniciamos con el caso en que S dispone únicamente 2 elementos. Si existen 2 nodos $i, j \in V$ con $w_i + w_j > W_u$, entonces el par de nodos no pueden ser incluidos en la misma clique. Así, se tiene:

PROPOSICIÓN 4.4. *Sea $\{i,j\} \in E$ tal que $w_i + w_j > w_u$. Entonces se tiene que para todo $c \in \{1,2,\ldots,k\}$, la ecuación $x_{ij}^c = 0$ es válida.*

Generalizando el resultado anterior se tiene:

PROPOSICIÓN 4.5. *Sea $S \subseteq V$ un cubrimiento minimal. Las desigualdades:*

$$\sum_{i \in S} y_i^c \leq |S| - 1, \qquad c = 1,2,\ldots,k$$

son válidas

Demostración. Dado que S es un cubrimiento minimal, todos los nodos en S no pueden ser asignados a la misma clique. Esto es, el mayor número de nodos en alguna clique debe ser $|S| - 1$. □

Para el presente trabajo se introduce el siguiente algoritmo, donde para su implementación consideramos conjuntos de cardinalidad 3 y 4.

Algoritmo 9: Desigualdades cover

```
1 para c = 1 hasta k hacer
2     para i = d hasta |V| − 2 hacer
3         para j = i + 1 hasta |V| − 1 hacer
4             para l = j + 1 hasta |V| hacer
5                 si w_i y_i^c + w_j y_j^c + w_l y_l^c > W_u entonces
6                     Introducir la desigualdad y_i^c + y_j^c + y_l^c ≤ 3
```

Capítulo 5

Resultados computacionales

En este capítulo se presentan los resultados computacionales obtenidos al implementar los modelos $\mathcal{F}_1$ y $\mathcal{F}_2$ para el problema de $k-$particionamiento con restricciones de tamaño y peso en el solver comercial Gurobi, así como también los resultados al incluir planos cortantes, definidos en el capítulo 4. Además, se incluye algunos detalles sobre la implementación de modelos enteros en Gurobi y los procesos usados para la inclusión de planos cortantes.

Las instancias usadas para el desarrollo de este proyecto fueron tomadas de Recalde *et al.* [2016], donde consideran instancias simuladas y reales con diferentes número de nodos y número de grupos o cliques a formarse. Dentro de los modelos $\mathcal{F}_1$ y $\mathcal{F}_2$ se definen las cotas superiores e inferiores para el peso de cada clique de la siguiente manera $W_L = \mu\frac{n}{k} - \sigma$ y $W_U = \mu\frac{n}{k} + \sigma$, donde μ es la media y σ la desviación estándar correspondientes al peso de todos los nodos.

Todas las pruebas computacionales fueron resueltas usando el solver de programación lineal entera Gurobi 7.01 en su configuración por defecto, en una Intel Core $I7$ 3.60 GHZ con 8 GB de memoria RAM y Ubuntu 14.01. El tiempo de ejecución para cualquier instancia se limitó a 7200s.

5.1 Gurobi

Para el presente proyecto se ha utilizado el solver Gurobi como motor de optimización computacional. Gurobi resuelve problemas de programación lineal (LP), programación cuadrática (QP) y programación entera mixta (MIP). Presenta una implementación de alto rendimiento del método del simplex e inclusión de planos cortantes, así como también de métodos no usados en este estudio, como el méto-

do dual simplex y el método de Barrier. Además, dispone de librerías que permiten trabajar en varios lenguajes de programación como C, C++, Java, Python, Matlab y *R*. Para mayor información ver en http://www.gurobi.com/documentation/.

Para resolver un problema de programación lineal *IP* dentro del solver elegido, se debe seguir los siguientes pasos:

1. **Creación del ambiente:** Los modelos de optimización funcionan dentro de un entorno utilizando el constructor de objeto *GRBEnv*, que inicialmente debe ser creado de la siguiente manera:

```
GRBEnv* env_modelo;
env_modelo = new GRBEnv();
```

2. **Creación del modelo:** El siguiente paso es crear un modelo que consta de:

   - Conjunto de variables
   - Conjunto de restricciones
   - Función objetivo

   El proceso más simple es crear el modelo vacío dentro de un ambiente fijo haciendo uso del constructor *GRBModel*. El código es el siguiente:

```
GRBModel*  modelo_ip;
modelo_ip = new GRBModel(*env_modelo);
```

   A continuación, se procede a la creación de variables, restricciones y función objetivo asociadas a un modelo:

   Variables: Para agregar variables de decisión a un modelo se utiliza la función *addVar*, cuyos argumentos de izquierda a derecha representa el valor de la cota inferior, cota superior, coeficiente en la función objetivo, tipo de variable (continua (*GRB_CONTINUOUS*), binaria (*GRB_BINARY*), entera (*GRB_INTEGER*)) y nombre para la nueva variable.

```
for (int l = 0; l < grupos; l++){
      for (int i = 0; i< card_V; i++){
         for (int j = i + 1; j < card_V; j++){
   ostringstream nombre_var;
   nombre_var << "x^" << l << "_" << i << "_" << j;
   x[l][i][j] = modelo_ip->addVar(0.0, 1.0, Datos[i][j],
   GRB_BINARY, nombre_var.str().c_str());}
```

```
        }
    }
```

Observar que el código descrito representa la creación de la variable x_{ij}^{c} para todo $c = \{1, \ldots, k\}$ y para todo $i, j \in V$ correspondientes a nuestros modelos. Donde el parámetro *grupos* nos indica el número de cliques a formarse, *card_V* representa la cardinalidad del conjunto de nodos V o V_A dependiendo del modelo a considerar, y la matriz $Datos[i][j]$ corresponde a los valores que toma la función de costo d con las instancias consideradas. Además, se puede observar que para eliminar soluciones redundantes solo se considera la variable donde $i < j$, puesto que se trabaja con un grafo no dirigido.

Restricciones: Para agregar restricciones lineales a un modelo se considera la función *addConstr*, donde los argumentos tomados de izquierda a derecha representan la expresión del lado izquierdo, sentido de la restricción (menor igual (*GRB_LESS_EQUAL*), igual (*GRB_EQUAL*), mayor igual (*GRB_GREATER_EQUAL*)), expresión del lado derecho y nombre para la nueva restricción.

En el presente código se muestra la creación de la restricción (3.4) del modelo $\mathcal{F}_1$ (ver capítulo 3).

```
for (int c = 0; c < grupos; c++){
      ostringstream restriccion1;
      restriccion1 << "Rest_1" << c;
      GRBLinExpr res1 = 0;
      for (int i = 0; i < card_V; i++){
         res1 += y[c][i];}
      modelo_ip->addConstr(res1, GRB_EQUAL, card_V / grupos,
   restriccion1.str());}
```

Función objetivo: Para establecer el sentido de optimización de un modelo se considera el atributo *ModelSense*. Debido a que éste tiene como valor predeterminado +1.0 que indica que la función será minimizada (*GRB_MINIMIZE*), como es el caso de nuestros modelos, no hace falta generar ninguna linea de código. Por otro lado si el sentido de optimización es maximizar, es necesario considerar la función *set* que nos permite modificar el valor del atributo:

```
modelo_ip->set(GRB_IntAttr_ModelSense, GRB_MAXIMIZE);
```

Una vez definido el modelo, se procede a optimizarlo mediante el método de Branch & Bound haciendo uso de la función *optimize*.

```
1   modelo_ip->optimize();
```

Debido a que la mayoría de información asociada a un modelo de Gurobi se almacena en un conjunto de atributos asociados con las variables, algunas restricciones o con el modelo mismo, es necesario en listar aquellos que han sido constantemente usados en este trabajo. Los atributos pueden tomar valores con caracteres (*GRB_CharAttr*), decimal doble (*GRB_DoubleAttr*), enteros (*GRB_IntAttr*) y de cadena de caracteres (*GRB_StringAttr*), donde pueden ser modificados o consultados mediante las funciones *set* y *get*, respectivamente.

Tabla 5.1
Atributos del modelo

Nombre del atributo	**Descripción**
NumVars	Número de variables
NumConstrs	Número de restricciones lineales
NumNZs	Número de coeficientes distintos de cero
ModelName	Nombre del modelo
ObjVal	Valor objetivo para la solución actual
MIPGap	La brecha de optimalidad MIP relativa actual
Runtime	Tiempo de ejecución para la optimización más reciente
Status	Estado de optimización actual
NodeCount	Número de nodos de Branch & Cut explorados en la optimización más reciente

Tabla 5.2
Atributos de variable

Nombre del atributo	**Descripción**
VarName	Nombre de variable
X	Valor en la solución actual

Además, es importante proporcionar información de aquellos parámetros que controlan el funcionamiento de los solucionadores de Gurobi dentro de los modelos propuestos. A diferencia de los atributos, los parámetros no pueden tomar valores con caracteres, es decir, solo pueden tomar valores con decimal doble (*GRB_DoubleParam*), enteros (*GRB_IntParam*) y de cadena de caracteres (*GRB_StringParam*). Éstos

deben ser modificados por la función *set* antes de que comience la optimización. Dentro de los diferentes tipos de parámetros disponibles de Gurobi, se presenta a continuación los más usados :

1. **Terminación:** Afectan la terminación de los algoritmos. Si el algoritmo supera cualquiera de estos límites, terminará e informará un estado de terminación no óptimo.
2. **Tolerancia:** Controlan las infracciones de factibilidad u optimalidad permisibles.
3. **MIP:** Controlan el funcionamiento de los algoritmos MIP.
4. **MIP Cuts:** Afectan la generación de planos cortantes en un MIP.
5. **Otros**

Tabla 5.3
Parámetros disponibles

Tipo	**Nombre**	**Propósito**
Terminación	TimeLimit	Límite de tiempo
Tolerancia	MIPGap	Brecha de optimalidad relativa de MIP
MIP	NodefileStart	Umbral de memoria para escribir nodos del árbol MIP en el disco
MIP Cuts	CliqueCuts	Generación de corte de clique
MIP Cuts	FlowCoverCut	Generación de corte de la cubierta de flujo
MIP Cuts	FlowPathCuts	Generación de corte de ruta de flujo
MIP Cuts	GUBCoverCuts	Generación de corte de cubierta GUB
MIP Cuts	ImpliedCuts	Generación de corte encuadernado implícita
MIP Cuts	MIPSepCuts	Generación de corte de separación MIP
MIP Cuts	ModKCuts	Generación de corte Mod-k
MIP Cuts	NetworkCuts	Generación de corte de red
MIP Cuts	SubMIPCuts	Generación de corte Sub-MIP
Otros	PreCrush	Se debe activar este parámetro cuando se usa Callback para agregar cortes propios
Otros	LazyConstraints	Se debe establecer este parámetro para agregar restricciones Lazys

Por otro lado, para la inclusión de las desigualdades válidas como planos cortantes en un modelo, Gurobi considera la clase **callback**. Para conectar esta clase con un modelo, se debe crear una subclase de la misma, como se muestra a continuación:

```
triineq_lazys cb = triineq_lazys(x, y, pesos, t_ineq, lazys_added,
    lazys_called, part2_added, cut_called, tricut_added, nodecnt,
    cover_added);
modelo_ip->setCallback(&cb);
```

En el código descrito, triineq_lazys es la clase requerida para la implementación de la clase callback, donde sus argumentos de izquierda a derecha representan la variable **x**, la variable **y**, valores que toma la función de pesos w sobre los nodos, vector de desigualdades triangulares, número de desigualdades lazys añadidas, número de desigualdades lazys llamadas, número de desigualdades 2-partición añadidas, número de desigualdades llamadas, número de desigualdades triangulares llamadas, número de desigualdades 2-partición añadidas, número de nodos de Branch & Bound explorados, número de desigualdades cover añadidas.

Las rutinas **callback** tiene dos argumentos:

where: Indica desde dónde se llama el optimizador de Gurobi.

what: Pasa por un método para obtener información adicional

Los valores que puede tomar el argumento *where* son:

Tabla 5.4
Parámetro where

where	**Valor numérico**	**Estado del optimizador**
POLLING	0	Retrollamada de sondeo periódica
PRESOLVE	1	Actualmente realizando presolve
SIMPLEX	2	Actualmente en simplex
MIP	3	Actualmente en *MIP*
MIPSOL	4	Encuentra una nueva solución *MIP*
MIPNODE	5	Actualmente explorando un nodo *MIP*
MESSAGE	6	Imprimir un mensaje de registro
BARRIER	7	Actualmente en barrera

Se ha considerado tomar en el trabajo los siguiente valores:

MIP: Se encuentra recorriendo el árbol.

MIPSOL: Encuentra una nueva solución que mejora la solución antes conocida.

MIPNODE: Acceso a los nodos del árbol de ramificación del Branch & Bound.

Por otro lado, para el argumento *what* que depende de *where*, se ha detallado solo aquellos valores usados en el presente trabajo .

Tabla 5.5
Parámetro what

what	where	Tipo	Descripción
MIP_NODCNT	*MIP*	double	Conteo de nodos explorado actual.
MIPNODE_STATUS	*MIPNODE*	int	Estado de optimización del nodo MIP actual

Para agregar desigualdades como planos cortantes dentro de los modelos, es necesario considerar los siguientes tipos de instrucciones que define Gurobi como métodos (*AddLAzy(),AddCut()*):

Lazys: Estas restricciones solamente son activadas cuando se encuentra una solución factible (entera) del nodo actual en el método Branch & Cut, en donde haciendo uso de la información de la solución, se procede ha verificar si viola cualquier restricción lazy. Si es así, una o más de las restricciones lazys violadas se incorporan al *PL*. Este tipo de restricciones pueden ser usadas cuando el valor del parámetro *where* es MIPSOL. Además, esta instrucción llamada a través de los callback necesita del parámetro *LazyConstraints* para ser usadas, como se observa:

```
modelo_ip->getEnv().set(GRB_IntParam_LazyConstraints, 1);
```

Planos cortantes (Cut): Son usados para agregar nuevos planos cortantes en cualquier nodo del árbol de Branch & Cut cortando así la solución actual (fraccional) del *PL*. El número de planos cortantes deben agregarse con moderación, puesto que el tamaño del modelo *PL* aumenta cada que se resuelve en cada nodo, generando así una degradación significativa en cuanto a la velocidad de procesamiento del nodo. Estas restricciones se lo ejecuta cuando el valor del parámetro *where* es MIPNODE. Así mismo, los planos cortantes para ser activados necesitan del parámetro *PreCrush*:

```
modelo_ip->getEnv().set(GRB_IntParam_PreCrush, 1);
```

Como se menciono en la sección anterior, la inclusión de las desigualdades triangulares en los modelos se lo hace mediante restricciones tipo lazy (formulación $\mathcal{F}_2$),

y como planos cortantes al momento de verificar el estado de las mismas. Las desigualdades 2-partición y cover son añadidas como planos cortantes.

Es importante mencionar que para la optimización de los modelos $\mathcal{F}_1$, $\mathcal{F}_2$ y $\mathcal{F}_2 - \mathcal{R}$, han sido desactivados los cortes de Gurobi (Clique, FlowCover, FlowPath, Implied, MIPSep, Network, ModK, GUBCover y SubMIP), debido a que estos parámetros afectan la generación de nuestros planos cortantes en el PL.

Finalmente, podemos indicar que un proceso de prefijado de variables es incluido en el método de solución. La idea sigue del hecho que podemos asignar un nodo específico a alguna clique sin que el conjunto de soluciones sea modificado, es decir, el nodo 1 puede ser asignado únicamente a la clique 1, el nodo 2 a la clique 1 o a la clique 2, y el nodo k puede pertenecer a las primeras k cliques. El resto de variables asociadas a estos nodos pueden ser fijadas a cero.

Entonces, el método de pre-procesamiento es definido de la siguiente manera:

- $y_1^1 = 1$,
- $y_i^c \in \{0,1\}$, $\forall i \in \{2,\ldots,k\}$, $c = 1,\ldots,i$,
- $y_i^c = 0$, $\forall i = \{1,\ldots,k\}$, $c = \{i+1,\ldots,k\}$.

Debido a que las variables asociadas a las aristas depende de las variables de asignación de los nodos a cliques, el proceso de ramificación está ligado únicamente a las variables y_i^c.

5.2 Análisis de resultados

Los resultados se encuentran resumidos en las tablas 5.7-5.16. En cada tabla se reporta la siguiente información: "M" toman los valores de: $\mathcal{F}_1$, cuando se considera el modelo $\mathcal{F}_1$; $\mathcal{F}_{1_C}$, $\mathcal{F}_1$ con la inclusión de planos cortantes; $\mathcal{F}_2$, considerando el modelo $\mathcal{F}_2$ y finalmente $\mathcal{F}_{2_C}$, cuando se refiere el modelo $\mathcal{F}_2 - \mathcal{R}$ con la inclusión de planos cortantes y la inclusión de restricciones tipo lazy. "F.Obj" muestra el valor de la función objetivo para la solución entera óptima encontrada en los diferentes modelos. Para la representación de los planos cortantes se ha usado la notación de "D.2-P", "D.Tr" y "D.C", donde nos devuelven el número de desigualdades 2-partición, triangulares y cover añadidas en los modelos, respectivamente; "B&B" indica el número de nodos explorados en el algoritmo Branch & Bound. Además en la columna "Tiempo" se proporciona el tiempo total de ejecución de los modelos; "Inst " indica

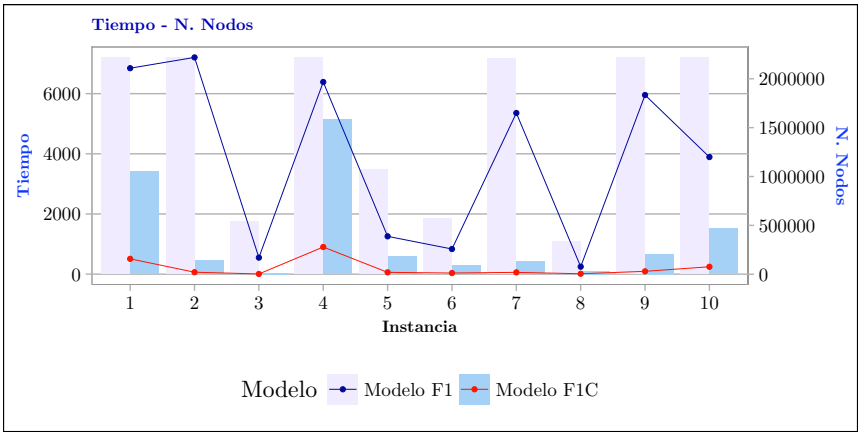
las instancias consideradas para la resolución del problema de k-particionamiento con restricciones de tamaño y peso. Además, para aquellas instancias cuyos resultados presenta un "$-$" en las respectivas columnas de cada tabla, es debido a que no se encontró solución. Además, se incluirá un análisis descriptivo que nos ayudará a visualizar el mejoramiento producido en los modelos al utilizar planos cortantes.

Dentro de las instancias aleatorias tomadas a manera de análisis, se considera la instancia con 34 nodos y 6 grupos a formarse (ver tabla 5.7 y 5.8), por ser la instancia que presenta mejor comportamiento en comparación de las otras.

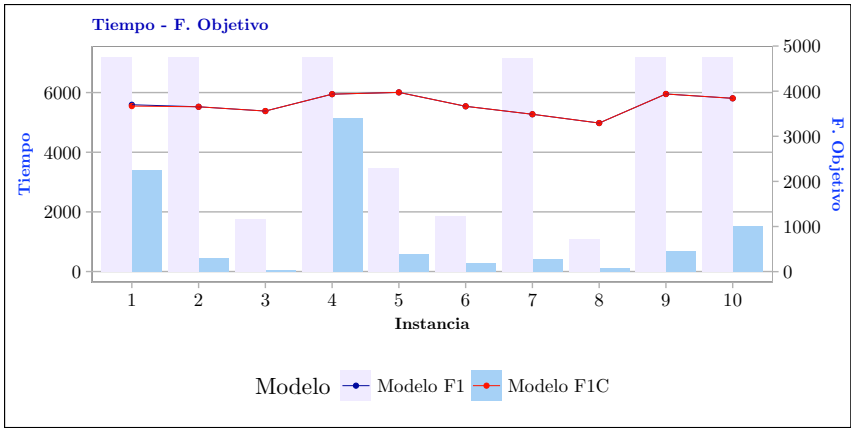
Se puede observar que la inclusión de los planos cortantes en el modelo $\mathcal{F}_1$, tomando las 10 primeras instancias, se llega al óptimo con una reducción considerable de tiempo al momento de la optimización, caso contrario, al no tener presentes estas desigualdades válidas en el modelo (en algunos casos) no hace posible llegar a la solución óptima por excederse en el tiempo limite establecido, llegando así a obtener un *Gap* mayor al 10 %. Gráficamente en la figura 5.1a se observa que el número de nodos explorados en el modelo $\mathcal{F}_1$ es mayor en comparación al modelo $\mathcal{F}_{1_C}$, haciendo que el tiempo de ejecución llegue al limite establecido con anterioridad y sin la necesidad de llegar a la solución óptima en ciertas instancias. Además, en la figura 5.1b se nota que la función a optimizar en ambos modelos son semejantes, sin embargo el modelo $\mathcal{F}_{1_C}$ mejora el tiempo de optimización del modelo $\mathcal{F}_1$.

Por otro lado, considerando las mismas instancias pero tomando el modelo $\mathcal{F}_2$, se puede observar (ver tabla 5.8) que no se presentan resultados tan comprometedores como en $\mathcal{F}_1$. Puesto que, a pesar de que el modelo $\mathcal{F}_{2_C}$ presenta una reducción considerable de *Gap* en la mayoría de las instancias, éste no llega a la obtención del óptimo. Así mismo, observando la figura 5.2a se puede notar que a pesar de que el tiempo de optimización en ambos modelos llega el límite, la cantidad de nodos explorados en $\mathcal{F}_{2_C}$ hace que el valor de la solución se encuentre más próxima al óptimo, ver figura 5.2b.

En general, se puede observar en el resto de instancias que el modelo $\mathcal{F}_1$ presenta mejoras en la optimización del problema al momento de la inclusión de planos cortantes, puesto que en en la mayoría de instancias se llega a la obtención de la solución óptima o en su peor caso, se obtiene una buena aproximación de la misma. Por el contrario el modelo $\mathcal{F}_{2_C}$, a pesar de presentar reducciones en el *Gap*, éste no llega a obtener una solución óptima en ninguna instancia aleatoria. Además, se puede observar que en el modelo $\mathcal{F}_2$ se presentan tiempos de optimización mucho más altos en comparación al modelo $\mathcal{F}_1$, esto se debe al número de restricciones



(a) Tiempo - Nodos



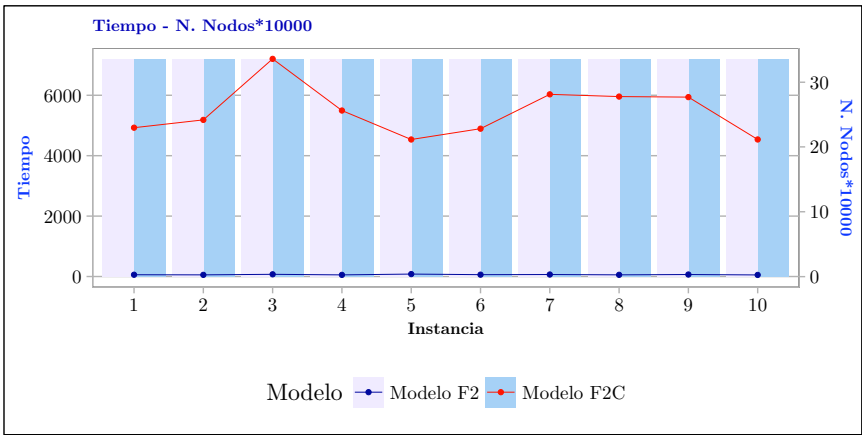
(b) Tiempo - F.Objetivo

Figura 5.1: Comparación entre los modelos: $\mathcal{F}_1$-$\mathcal{F}_{1_C}$

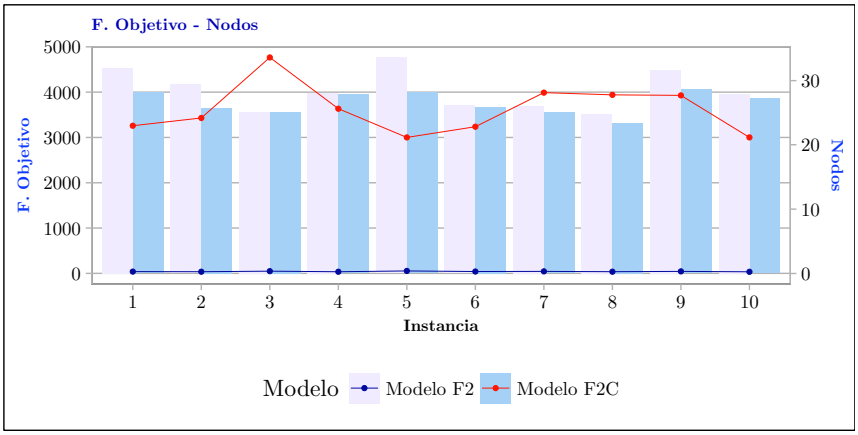
existentes en el modelo (ver capitulo 3).

Es importante notar que la desigualdades triangulares como restricciones tipo lazy en el modelo $\mathcal{F}_{1_C}$ no son incluidas ya que no forman parte de la formulación, caso contrario pasa en el modelo $\mathcal{F}_2$ que estás como se mencionó previamente pertenecen al modelo, pero que al considerarlas en el modelo $\mathcal{F}_2 - \mathcal{R}$ estás son incluidas como restricciones lazy y no como restricciones propias.

Por otro lado, las instancias reales relacionadas a la segunda división del Campeonato Ecuatoriano de Fútbol tratadas en Recalde *et al.* [2016], se encuentran reportadas en las tablas 5.15 y 5.16. Se puede notar que en dicho artículo las instancias 2014 y 2015 correspondientes a las ediciones de los años respectivos, sin la necesidad de incluir planos cortantes al modelo $\mathcal{F}_1$ se llega a la optimalidad, a diferencia de la instancia 2015−proposal, que como se puede observar en la tabla 5.15, gracias a la inclusión de las desigualdades válidas se alcanza su solución óptima en un



(a) Tiempo - Nodos



(b) F.Objetivo - Nodos

Figura 5.2: Comparación entre los modelos: $\mathcal{F}_2$-$\mathcal{F}_{2_C}$

tiempo razonable. Por el contrario, al considerar el modelo $\mathcal{F}_2$ (ver tabla 5.16), se observa que solo la instancia 2014 llega al óptimo sin considerar los planos cortantes; la instancia 2015 requiere que se añada las desigualdades válidas al modelo para encontrar la solución óptima, mejorando así el tiempo de optimización, y finalmente la instancia 2015-proposal, solo llega a la obtención de una solución factible con la ayuda de la inclusión de los planos cortantes, pero excediendo el tiempo establecido.

Además, se puede observar en la tabla (5.6) que la mayoría de las instancias presentan en promedio un menor tiempo cuando se incluyen planos cortantes, y en varios casos se tiene una reducción significativa en el *Gap*. De igual manera, se observa un cambio notorio de la disminución en promedio del número de nodos que se usa en el algoritmo de Branch & Cut. Debido a que en la instancia 36 con número de nodos igual a 54 no se encontró una solución factible considerando el modelo $\mathcal{F}_{2_C}$, se procedió a calcular el promedio únicamente con 9 valores. Un caso similar

se presenta con las instancias con 54 nodos considerando el modelo $\mathcal{F}_2$, las cuales no están incluidas en la tabla (5.6) debido a que ninguna presenta una solución factible.

Tabla 5.6
Promedios

n	Modelo	F.Objetivo	Tiempo	Gap	Nodos
34	$\mathcal{F}_1$	3704.32	5131.94	3.94	1186696.10
34	$\mathcal{F}_{1_C}$	3701.42	1263.89	0.00	61182.80
34	$\mathcal{F}_2$	4036.39	7201.07	33.27	2909.70
34	$\mathcal{F}_{2_C}$	3760.36	7200.48	21.89	255112.10
38	$\mathcal{F}_1$	3418.34	3014.46	0.97	136004.50
38	$\mathcal{F}_{1_C}$	3418.34	3942.46	1.38	80305.70
38	$\mathcal{F}_2$	3578.10	7202.80	22.57	1763.40
38	$\mathcal{F}_{2_C}$	3459.13	7200.29	13.48	240643.70
44	$\mathcal{F}_1$	3893.67	7200.05	9.68	131748.10
44	$\mathcal{F}_{1_C}$	3861.14	7157.97	9.37	61764.30
44	$\mathcal{F}_2$	4606.89	7213.00	37.38	1224.10
44	$\mathcal{F}_{2_C}$	4063.84	7200.73	25.49	127075.90
54	$\mathcal{F}_1$	4912.53	7200.09	30.21	34349.50
54	$\mathcal{F}_{1_C}$	4738.72	7204.01	26.27	27244.90
54	$\mathcal{F}_{2_C}$	5334.81	7200.33	38.23	71494.56

Tabla 5.7
Modelos $\mathcal{F}_1$ y $\mathcal{F}_{1_C}$- Instancias con $n = 34$ y $k = 6$

Inst	M	F.Obj	Tiempo	Gap	D-2P	D.Tr	D.C	Lazy	B&B
1	$\mathcal{F}_1$	3698.9	7200.01	10.74					2108680
	$\mathcal{F}_{1_C}$	3669.6	3212.16	0	10001	586	4	0	143205
2	$\mathcal{F}_1$	3653.6	7200.02	64.26					2218970
	$\mathcal{F}_{1_C}$	3653.6	460.37	0	1758	579	10	0	19500
3	$\mathcal{F}_1$	3559.3	1749.4	0					169289
	$\mathcal{F}_{1_C}$	3559.3	32.97	0	1490	596	40	0	1924
4	$\mathcal{F}_1$	3933.7	7200.02	12.36					1967230
	$\mathcal{F}_{1_C}$	3933.7	5148.96	0	87001	680	123	0	277504
5	$\mathcal{F}_1$	3973.5	3471.2	0					386992
	$\mathcal{F}_{1_C}$	3973.5	578.83	0	9503	634	6	0	18487

Inst	M	F.Obj	Tiempo	Gap	D-2P	D.Tr	D.C	Lazy	B&B
6	$\mathcal{F}_1$	3664.5	1852.81	0					257161
	$\mathcal{F}_{1_C}$	3664.5	294.79	0	3566	642	83	0	11401
7	$\mathcal{F}_1$	3487.8	7160.92	0					1649650
	$\mathcal{F}_{1_C}$	3487.8	431.59	0	1690	621	57	0	18248
8	$\mathcal{F}_1$	3293.1	1084.97	0					76189
	$\mathcal{F}_{1_C}$	3293.1	102.73	0	1173	576	11	0	3918
9	$\mathcal{F}_1$	3937.4	7200.01	68.32					1834070
	$\mathcal{F}_{1_C}$	3937.4	671.29	0	6359	616	1	0	28762
10	$\mathcal{F}_1$	3841.7	7200.01	29.96					1198730
	$\mathcal{F}_{1_C}$	3841.7	1513.36	0	18617	469	105	0	75597

Tabla 5.8
Modelos $\mathcal{F}_2$ y $\mathcal{F}_{2_C}$ - Instancias con n=34 y k=6

Inst	M	F.Obj	Tiempo	Gap	D-2P	D.Tr	D.C	Lazy	B&B
1	$\mathcal{F}_2$	4530.8	7200.01	47.45					2753
	$\mathcal{F}_{2_C}$	3990.8	7200.72	34.1	18264	266	12	6322	229748
2	$\mathcal{F}_2$	4167.3	7218.99	37.61					2579
	$\mathcal{F}_{2_C}$	3653.6	7200.65	19.64	9063	268	63	4887	241798
3	$\mathcal{F}_2$	3559.3	7200.1	22.34					3405
	$\mathcal{F}_{2_C}$	3559.3	7200.2	13.76	40479	341	2930	3234	335985
4	$\mathcal{F}_2$	3988.6	7200.02	30.56					2544
	$\mathcal{F}_{2_C}$	3950.8	7200.36	24.16	12441	351	177	4464	256257
5	$\mathcal{F}_2$	4765.7	7200.02	38.48					3784
	$\mathcal{F}_{2_C}$	3994.4	7200.29	21.89	16273	270	59	5361	211526
6	$\mathcal{F}_2$	3705.1	7200.03	27.72					2843
	$\mathcal{F}_{2_C}$	3664.5	7200.08	21.08	12925	346	874	4770	228199
7	$\mathcal{F}_2$	3697.8	7200.01	27.31					3061
	$\mathcal{F}_{2_C}$	3551.5	7200.7	19.73	19433	284	2845	6169	281254
8	$\mathcal{F}_2$	3512.4	7200.02	30.15					2599
	$\mathcal{F}_{2_C}$	3322.8	7200.69	16.87	9991	348	271	2619	277813
9	$\mathcal{F}_2$	4485.6	7200.01	38.44					3076
	$\mathcal{F}_{2_C}$	4055.7	7200.14	23.61	11569	274	69	6538	276957
10	$\mathcal{F}_2$	3951.3	7200.02	32.63					2453
	$\mathcal{F}_{2_C}$	3860.2	7200.92	24.1	20615	263	481	4804	211584

Tabla 5.9
Modelos $\mathcal{F}_1$ y $\mathcal{F}_{1_C}$- Instancias con n=38 y k=7

Inst	M	F.Obj	Tiempo	Gap	D-2P	D.Tr	D.C	Lazy	B&B
11	$\mathcal{F}_1$	3344.4	1489.67	0					79489
	$\mathcal{F}_{1_C}$	3344.4	358.98	0	3665	875	99	0	4183
12	$\mathcal{F}_1$	3451.3	2306.56	0					87627
	$\mathcal{F}_{1_C}$	3451.3	7200.9	46.97	10001	781	235	0	118852
13	$\mathcal{F}_1$	3306	674.95	0					38211
	$\mathcal{F}_{1_C}$	3306	552.94	0	4742	1050	199	0	9704
14	$\mathcal{F}_1$	3646.4	939.55	0					46805
	$\mathcal{F}_{1_C}$	3646.4	7200.01	37.32	84966	941	2766	0	150784
15	$\mathcal{F}_1$	3262.3	7200.03	33.90					367941
	$\mathcal{F}_{1_C}$	3262.3	7200.93	55.73	94055	966	7166	0	146945
16	$\mathcal{F}_1$	3441.3	3148.2	0					138233
	$\mathcal{F}_{1_C}$	3441.3	4000.39	0	10001	936	892	0	65993
17	$\mathcal{F}_1$	3366.4	7200.04	62.68					260862
	$\mathcal{F}_{1_C}$	3366.4	4869.04	0	69839	1006	1327	0	170111
18	$\mathcal{F}_1$	3532	2128.13	0					138074
	$\mathcal{F}_{1_C}$	3532	352.21	0	2752	1183	16	0	4682
19	$\mathcal{F}_1$	2986.7	288.52	0					17764
	$\mathcal{F}_{1_C}$	2986.7	1395.45	0	17953	984	201	0	25518
20	$\mathcal{F}_1$	3846.6	4768.98	0					185039
	$\mathcal{F}_{1_C}$	3846.6	6165.14	0	108885	1133	2414	0	99499

Tabla 5.10
Modelos $\mathcal{F}_2$ y $\mathcal{F}_{2_C}$- Instancias con n=38 y k=7

Inst	M	F.Obj	Tiempo	Gap	D-2P	D.Tr	D.C	Lazy	B&B
11	$\mathcal{F}_2$	3591.4	7237.77	23.36					1149
	$\mathcal{F}_{2_C}$	3344.4	7200.16	9.62	7204	350	977	2469	199466
12	$\mathcal{F}_2$	3609.7	7200.02	21.54					2038
	$\mathcal{F}_{2_C}$	3456.8	7200.44	13.1	5976	223	587	2214	226816
13	$\mathcal{F}_2$	3361.7	7200.04	18.08					1788
	$\mathcal{F}_{2_C}$	3361.7	7200.19	12.54	8951	299	535	2841	249393
14	$\mathcal{F}_2$	3677.5	7200.12	21.63					1441
	$\mathcal{F}_{2_C}$	3646.4	7200.35	15.20	6335	307	367	2039	288668

Inst	M	F.Obj	Tiempo	Gap	D-2P	D.Tr	D.C	Lazy	B&B
15	$\mathcal{F}_2$	3815.2	7200.03	29.5					1797
	$\mathcal{F}_{2_C}$	3389.1	7200.29	18.11	5201	382	1186	3655	165722
16	$\mathcal{F}_2$	3550.6	7200.03	23.48					2012
	$\mathcal{F}_{2_C}$	3538.2	7200.18	16.59	10376	344	982	4297	180914
17	$\mathcal{F}_2$	3638.4	7200.09	28.78					2736
	$\mathcal{F}_{2_C}$	3408.1	7200.49	13.89	14178	361	1420	3386	306643
18	$\mathcal{F}_2$	3565.2	7200.1	17.74					1846
	$\mathcal{F}_{2_C}$	3565.2	7200.29	11.04	6846	452	75	2441	265357
19	$\mathcal{F}_2$	3047.9	7200.02	17.36					1388
	$\mathcal{F}_{2_C}$	3029	7200.04	11.11	5891	460	495	2043	289742
20	$\mathcal{F}_2$	3923.4	7200.03	24.22					1439
	$\mathcal{F}_{2_C}$	3852.4	7200.43	13.62	12175	272	1618	2550	233716

Tabla 5.11
Modelos $\mathcal{F}_1$ y $\mathcal{F}_{1_C}$ - Instancias con n=44 y k=8

Inst	M	F.Obj	Tiempo	Gap	D-2P	D.Tr	D.C	Lazy	B&B
21	$\mathcal{F}_1$	3706.1	7200.03	63.95					107339
	$\mathcal{F}_{1_C}$	3729	7203.4	82.6	60833	1031	74	0	48920
22	$\mathcal{F}_1$	3684.1	7200.02	90.14					91441
	$\mathcal{F}_{1_C}$	3664.6	7203.27	14.29	10003	932	601	0	63380
23	$\mathcal{F}_1$	3734.1	7200.01	74.80					279538
	$\mathcal{F}_{1_C}$	3591.2	7201.02	59.95	90972	1058	98	0	87802
24	$\mathcal{F}_1$	3812.3	7200.09	10.15					279538
	$\mathcal{F}_{1_C}$	3840.9	7201.19	14.87	10001	1114	171	0	69795
25	$\mathcal{F}_1$	4450.3	7200.14	16.86					77715
	$\mathcal{F}_{1_C}$	4308.7	7201.43	14.814	10002	1143	2238	0	50541
26	$\mathcal{F}_1$	3954.8	7200.03	15.39					231734
	$\mathcal{F}_{1_C}$	3887.1	7204.23	81.47	75097	1076	370	0	70183
27	$\mathcal{F}_1$	3555	7200.02	49.34					110858
	$\mathcal{F}_{1_C}$	3555	7201.55	25.46	61070	1159	62	0	74982
28	$\mathcal{F}_1$	3603.5	7200.03	41.93					134736
	$\mathcal{F}_{1_C}$	3567.7	6733.45	0	20534	943	92	0	51766
29	$\mathcal{F}_1$	4574.2	7200.06	20.02					103860
	$\mathcal{F}_{1_C}$	4376.7	7206.99	14.26	96950	1190	21	0	52420

Inst	M	F.Obj	Tiempo	Gap	D-2P	D.Tr	D.C	Lazy	B&B
30	$\mathcal{F}_1$	3862.3	7200.02	23.85					91441
	$\mathcal{F}_{1_C}$	4027.5	7206.9	87.75	106690	1099	66	0	47762

Tabla 5.12
Modelo $\mathcal{F}_2$ y $\mathcal{F}_{2_C}$- Instancias con n=44 y k=8

Inst	M	F.Obj	Tiempo	Gap	D-2P	D.Tr	D.C	Lazy	B&B
21	$\mathcal{F}_2$	3914.1	7200.42	31.02					1727
	$\mathcal{F}_{2_C}$	3772	7200.14	22.59	8078	379	332	5540	116796
22	$\mathcal{F}_2$	4522.4	7201.42	40.52					1194
	$\mathcal{F}_{2_C}$	4159	7200.84	32.91	11892	458	1713	6912	144494
23	$\mathcal{F}_2$	4218	7200.19	32.77					1135
	$\mathcal{F}_{2_C}$	3692.7	7203.07	21.37	34978	398	558	10783	109346
24	$\mathcal{F}_2$	4200	7200.91	37.08					1158
	$\mathcal{F}_{2_C}$	4068.8	7200.43	32.69	29529	372	491	10834	170902
25	$\mathcal{F}_2$	5361.4	7226.25	41.55					1105
	$\mathcal{F}_{2_C}$	4794.4	7200.53	32.38	14215	346	1835	8247	123273
26	$\mathcal{F}_2$	5189.8	7200.62	45.15					1328
	$\mathcal{F}_{2_C}$	4233.7	7201.11	27.78	38404	388	1156	11654	126879
27	$\mathcal{F}_2$	4047.5	7200.46	31.18					1143
	$\mathcal{F}_{2_C}$	3732.8	7200.62	21.15	10289	475	129	3038	146933
28	$\mathcal{F}_2$	4694.7	7200.67	38.75					1164
	$\mathcal{F}_{2_C}$	3567.7	7200.4	15.16	7914	458	188	4589	112913
29	$\mathcal{F}_2$	4947.7	7225.01	37.82					1166
	$\mathcal{F}_{2_C}$	4509.6	7200.12	27.8	15433	458	9	5864	113915
30	$\mathcal{F}_2$	4973.3	7200.04	35.58					1121
	$\mathcal{F}_{2_C}$	4107.7	7200.09	21.06	12956	362	148	9902	105308

Tabla 5.13
Modelos $\mathcal{F}_1$ y $\mathcal{F}_{1_C}$- Instancias con n=54 y k=8

Inst	M	F.Obj	Tiempo	Gap	D-2P	D.Tr	D.C	Lazy	B&B
31	$\mathcal{F}_1$	5572.5	7200.02	34.16					38912
	$\mathcal{F}_{1_C}$	5329	7200.56	29.98	43306	1796	763	0	30593

Inst	M	F.Obj	Tiempo	Gap	D-2P	D.Tr	D.C	Lazy	B&B
32	$\mathcal{F}_1$	4595.9	7200.02	27.85					33312
	$\mathcal{F}_{1_C}$	4213.9	7200.95	20.85	10002	1659	2678	0	22727
33	$\mathcal{F}_1$	4508.3	7200.12	24.93					40098
	$\mathcal{F}_{1_C}$	4486.8	7204.14	22.50	10004	1694	1129	0	27822
34	$\mathcal{F}_1$	5063.1	7200.27	32.59					36093
	$\mathcal{F}_{1_C}$	5113	7203.72	34.01	61268	1646	36	0	32060
35	$\mathcal{F}_1$	5248.7	7200.19	33.05					35504
	$\mathcal{F}_{1_C}$	4897.9	7208.25	25.94	75651	1870	831	0	28895
36	$\mathcal{F}_1$	5629.9	7200.03	39.99					29050
	$\mathcal{F}_{1_C}$	4763.9	7200.9	23.45	10002	1470	338	0	27630
37	$\mathcal{F}_1$	4986.7	7200.09	32.09					31131
	$\mathcal{F}_{1_C}$	4770.8	7204.88	27.25	10003	1598	756	0	24409
38	$\mathcal{F}_1$	4510.4	7200.07	26.14					34998
	$\mathcal{F}_{1_C}$	4176.8	7200.03	21.18	10004	1692	882	0	26643
39	$\mathcal{F}_1$	4553.1	7200.02	23.92					32968
	$\mathcal{F}_{1_C}$	4860.5	7200.59	28.85	85578	1742	11	0	28186
40	$\mathcal{F}_1$	4456.7	7200.05	27.4					31429
	$\mathcal{F}_{1_C}$	3907.2	7200.88	14.27	10001	1567	883	0	29381

Tabla 5.14
Modelos $\mathcal{F}_2$ y $\mathcal{F}_{2_C}$ - Instancias con n=54 y k=8

Inst	M	F.Obj	Tiempo	Gap	D-2P	D.Tr	D.C	Lazy	B&B
31	$\mathcal{F}_2$	-	-	-					-
	$\mathcal{F}_{2_C}$	5146.7	7200.66	32.48	6475	502	880	6981	77073
32	$\mathcal{F}_2$	-	-	-					-
	$\mathcal{F}_{2_C}$	5149.5	7201.54	37.31	27623	535	2111	6566	50566
33	$\mathcal{F}_2$	-	-	-					-
	$\mathcal{F}_{2_C}$	5305.7	7202.29	38.74	9947	467	2019	5799	59144
34	$\mathcal{F}_2$	-	-	-					-
	$\mathcal{F}_{2_C}$	6173	7203.03	49.43	29457	457	1009	11395	99015
35	$\mathcal{F}_2$	-	-	-					-
	$\mathcal{F}_{2_C}$	5840.2	7202.24	41.96	22921	506	3977	9277	78070
36	$\mathcal{F}_2$	-	-	-					-
	$\mathcal{F}_{2_C}$	-	-	-	-	-	-	-	-

Inst	M	F.Obj	Tiempo	Gap	D-2P	D.Tr	D.C	Lazy	B&B
37	$\mathcal{F}_2$	-	-	-					-
	$\mathcal{F}_{2_C}$	6201.1	7201.7	46.88	12645	466	2424	12144	75857
38	$\mathcal{F}_2$	-	-	-					-
	$\mathcal{F}_{2_C}$	4493.5	7202.2	29.22	16034	532	2443	7714	63711
39	$\mathcal{F}_2$	-	-	-					-
	$\mathcal{F}_{2_C}$	5001.2	7200.33	34.53	21229	441	49	9162	76464
40	$\mathcal{F}_2$	-	-	-					-
	$\mathcal{F}_{2_C}$	4702.4	7200.96	33.47	15617	465	1334	10602	63551

Tabla 5.15
Modelos $\mathcal{F}_1$ y $\mathcal{F}_{1_C}$ - Instancias reales

Inst	n	k	M	F.Obj	Tiempo	Gap	D-2P	D.Tr	D.C	Lazy	B&B
2014	21	4	$\mathcal{F}_1$	9648.7	36.43	0					15744
			$\mathcal{F}_{1_C}$	9648.7	2.2	0	230	699	8	0	266
2015	22	4	$\mathcal{F}_1$	11412.7	20.65	0					7508
			$\mathcal{F}_{1_C}$	11412.7	1.47	0	94	558	6	0	220
2015-proposal	44	8	$\mathcal{F}_1$	14000	7200.02	14.74					47479
			$\mathcal{F}_{1_C}$	14000	519.18	0	4039	1068	174	0	3939

Tabla 5.16
Modelos $\mathcal{F}_2$ y $\mathcal{F}_{2_C}$ - Instancias reales

Inst	n	k	M	F.Obj	Tiempo	Gap	D-2P	D.Tr	D.C	Lazy	B&B
2014	21	4	$\mathcal{F}_2$	9648.7	399.56	0					1886
			$\mathcal{F}_{2_C}$	9648.7	2.13	0	61	205	24	464	960
2015	22	4	$\mathcal{F}_2$	11412.7	7200.01	100.75					110065
			$\mathcal{F}_{2_C}$	11412.7	181.39	0	1105	256	117	1381	83090
2015-proposal	44	8	$\mathcal{F}_2$	-	-	-					-
			$\mathcal{F}_{2_C}$	14000	7200.8	15.7	37425	450	9634	4374	169345

Capítulo 6

Conclusiones y comentarios

En este trabajo diseñamos dos métodos de solución para encontrar k particiones de un grafo no dirigido, sujetas a restricciones de tamaño y peso. Ambos trabajan sobre un modelo de programación entera, aprovechando su estructura y propiedades; estudiamos las formulaciones $\mathcal{F}_1$ y $\mathcal{F}_2$ las cuales consisten en la formulación del problema sin la inclusión de planos cortantes, y las formulaciones $\mathcal{F}_{1_C}$ y $\mathcal{F}_{2_C}$, que consisten en la formulación del problema con la inclusión de planos cortantes válidos.

El primer método, es un algoritmo tipo Branch & Bound, es muy utilizado para resolver problemas de programación lineal debido a su enumeración inteligente para poder llegar a una solución óptima. Por otro lado, el segundo método de solución es un algoritmo tipo Branch & Cut; el cual nos permitió encontrar planos cortantes válidos para los modelos $\mathcal{F}_1$ y $\mathcal{F}_2$. Este método es un algoritmo exacto que consiste en la unión del método de planos de corte de Gomory y el método de Branch & Bound, donde mejora la eficiencia de los métodos antes mencionados reduciendo el número de nodos. Estas diferencias tienen su impacto en los resultados computacionales, puesto que el segundo método nos indica que al comparar los modelos $\mathcal{F}_1$ con $\mathcal{F}_{1_C}$ y $\mathcal{F}_2$ con $\mathcal{F}_{2_C}$, se obtiene mejores resultados con el algoritmo de Branch & Cut, observándose una mejora significativa en el tiempo ocupado y Gap alcanzado. Es así, que el tiempo promedio de resolución entre $\mathcal{F}_1$ y $\mathcal{F}_{1_C}$ se redujo en un 46.16 %, y a su vez el Gap bajó en promedio un 24.74 %. Estas variaciones se encuentran en la tabla 6.1, en donde los valores que toman –, se debe a que en el modelo $\mathcal{F}_2$ las instancias con 54 nodos y 8 grupos a formarse, y la instancia 2015-proposal no presentan soluciones factibles, haciendo que no sea posible comparar con las soluciones presentadas en el modelo $\mathcal{F}_{2_C}$.

Tabla 6.1
Variación porcentual del Tiempo y Gap entre modelos

	Tiempo		*Gap*	
Instancias	$Var(\mathcal{F}_1 - \mathcal{F}_{1_C})$	$Var(\mathcal{F}_2 - \mathcal{F}_{2_C})$	$Var(\mathcal{F}_1 - \mathcal{F}_{1_C})$	$Var(\mathcal{F}_2 - \mathcal{F}_{2_C})$
34	-75.37 %	-0.01 %	-100.00 %	-34.19 %
38	30.78 %	-0.03 %	43.06 %	-40.27 %
44	-0.58 %	-0.17 %	-3.18 %	-31.80 %
54	0.05 %	–	-13.05 %	–
2014	-93.97 %	-94.67 %	0.00 %	0.00 %
2015	-91.72 %	-97.48 %	0.00 %	-100.00 %
2015-proposal	-92.33 %	–	-100.00 %	–
Promedio	-46.16 %	-27.48 %	-24.74 %	-29.47 %

Así mismo, en las tablas proporcionadas en la sección 5.2, se puede notar que el número de nodos utilizados en el algoritmo de Branch & Cut es inferior al número de nodos utilizados en el algoritmo de Branch & Bound. Como se mencionó anteriormente mejora la eficiencia del algoritmo Branch & Bound, reduciendo el número de problemas PL generados para resolver el problema de programación lineal entera (IP); lo que significa una disminución notable en el tiempo de solución.

Por otro lado, cabe recalcar que la inclusión de las desigualdades cover como planos cortantes en los modelos han sido de gran importancia para lograr tener una solución óptima o una buena aproximación de la misma en la mayoría de instancias. Estas desigualdades han sido consideradas únicamente en el presente trabajo como desigualdades válidas para este problema, por otro lado, las desigualdades triangulares y 2-partición han sido estudiadas por varios autores en trabajos previos, como es el caso de Grötschel y Wakabayashi [1989] en donde fueron demostradas desigualdades válidas.

Así, podemos concluir que la inclusión de los planos cortantes, que hemos propuesto en el presente trabajo, han sido útiles para resolver el problema de particionamiento de grafos en k subgrafos sujetos a restricciones de tamaño y peso. La eficiencia del método propuesto nos lleva a cuestionarnos si existen diferentes planos cortantes que sean válidos para este problema en particular, lo cual se propone para trabajo a futuro. También, sería de interés estudiar una modificación a la formulación del problema, o incluso estudiar la posibilidad de generar una heurísitca que ayude a la solución del problema.

Bibliografía

Armbruster, M., Fügenschuh, M., Helmberg, C., y Martin, A. (2008). A Comparative Study of Linear and Semidefinite Branch-and-Cut Methods for Solving the Minimum Graph Bisection Problem. *Springer*.

Bertsimas, D. y Tsitsiklis, J. (1998). *Introduction to Linear Optimization*.

Bhasker, J. y Samad, T. (1990). The clique-partitioning problem. *Computera Math. Applic.*

Bichot, C. (2007). Élaboration d'une nouvelle métaheuristique pour le partitionnement de graphe: la méthode de fusion-fission. Aplication au découpage de l'espace aérien. *Tesis PhD, Institut National Polytechnique de Toulouse (INPT)*.

Bichot, C. y Siarry, P., editores (2011). *Graph Partitioning*. ISTE.

Brunetta, L., Conforti, M., y Rinaldi, G. (1997). A Branch-and-Cut Algorithm for the Equicut Problem. *Mathematical Programming*.

Catanzaro, D., Gourdinb, E., Labbé, M., y Ozsoy, F. (2011). A branch-and-cut algorithm for the partitioning-hub location-routing problem. *Comput Oper Res*.

Chopra, S. y Rao, M. (1993). The partition problem. *Mathematical Programming*.

Conforti, M., Rao, M., y Sassano, A. (1990). The equipartition polytope I and II. *Mathematical Programming, 49:49–70*.

Dakin, R. J. (1965). A tree-search algorithm for mixed integer programming problems. *The Computer Journal*.

Fairbrother, J., Letchford, A., y Briggs, K. (2017). A two-level graph partitioning problem arising in mobile wireless communications. *Discr Optim (accepted paper)*.

Ferreira, C., Martin, A., de Souza, C., Weismantel, R., y Wolsey, L. (1998). The node capacitated graph partitioning problem: a computational study. *Mathematical Programming 81*, pp. 229–256.

Gomory, R. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, p. 275–278.

Gomory, R. (1960a). An algorithm for the mixed integer problem. *DTIC Document*.

Gomory, R. (1960b). Solving linear programming problems in integers. *Combinatorial Analysis*.

Grötschel, M. y Wakabayashi, Y. (1989). A cutting plane algorithm for a clustering problem. *Mathematical Programming 45*, pp. 59–96.

Grötschel, M. y Wakabayashi, Y. (1990). Facets of the clique partitioning polytope. *Mathematical Programming 47*, pp. 367–387.

Holyer, I. (1981). The NP-Completeness of some edge-partition problem. *SIAM Journal of Computing*.

Jaehn, F. y Pesch, E. (2013). New bounds and constraint propagation techniques for the clique partitioning problem. *Discrete Applied Mathematics 161*, pp. 2025–2037.

Ji, X. y Mitchell, J. (2005). The Clique Partition Problem with Minimum Clique Size Requirement. *Department of Mathematical, Sciences Rensselaer Polytechnic Institute*, pp. 5–8.

Ji, X. y Mitchell, J. (2006). Branch-and-price-and-cut on the clique partitioning problem with minimum clique size requirement. *Department of Mathematical Sciences,Rensselaer Polytechnic Institute*.

Kahng, A., Lienig, J., Markov, I., y Hu, J. (2011). VLSI Physical Design: From Graph Partitioning to Timing Closure. *Springer Publishing Company, Incorporated*.

Kaparis, K. y Letchford, A. (2010). Cover Inequalities. *Lancaster University*.

Labbé, M. y Ozsoy, F. A. (2010). Size-constrained graph partitioning polytopes. *Discrete Mathematics 310 (24)*, p. 3473 – 3493.

Lisser, A. y Rendl, F. (2003). Graph Partitioning using Linear and Semidefinite Programming. *Mathematical Programming*.

Menegola, B. (2012). A Study of the k-way Graph Partition Problema. *Department of Theoritical Computer Science*, pp. 25–30.

Mitchell, J. (1999). Branch-and-Cut Algorithms for Combinatorial Optimization Problem. *Mathematical Sciences Rensselaer Polytechnic Institute*.

Mitchell, J. (2001). Branch & cut for the k-way equipartition problem. *Technical report, Rensselaer Polytechnic Institute*.

Mitchell, J. (2003). Realignment in the national football league: Did they do it right? Naval. *Naval research Logistics (NRL) 50(7)*.

Oktay, Gunluk, Gerhard, J., y Woerginger (2011). *International Conference on Integer Programming and Combinatorial Optimization*.

Padberg, M. y G.Rinaldi (1989). A branch and cut approach to a traveling salesman problem with side constraints. *Management Science*.

Poljak, S. y Tuza, Z. (1995). Maximum cuts and largest bipartite subgraphs. *Series in Discrete Mathematics and Theoretical Computer Science.*

Recalde, D., Severín, D., Torres, R., y Vaca, P. (2016). Balance Partition of a Graph for Football Team Realingment in Ecuador. *Lecture Notes in Computer Science 9849. Springer Verlag. LNCS*, pp. 357–368.

Schloegel, K., Karypis, G., y Kumar, V. (2003). Graph Partitioning for High-Performance Scientific Simulations. *Morgan Kaufmann Publishers*.

Capítulo 7

Anexos

7.1 Código ($\mathcal{F}_1$, $\mathcal{F}_2 - \mathcal{R}$)

7.1.1 ProblemaMIP.h

```
#pragma once
#ifndef PROBLEMAMIP_H
#define PROBLEMAMIP_H

#include <iostream>
#include <sstream>
#include <gurobi_c++.h>
#include <vector>
#include <list>
#include <queue>

using namespace std;
 const double INF = 1e+20;
 double media(vector<double>x);
 double desviacion(vector<double>x);
 void escribir_matriz(vector<vector<double> > A);
 void instancia(string Nombre_Instancia, bool incluir_artificiales,
   vector<double>&pesos, vector<vector<double> >&Distancias, int &n,
   int &k);
 void crear_vars(int tipo_modelo, vector< vector<double> > Datos,
   vector<double> pesos, int card_V, int grupos, GRBEnv* env_modelo,
   GRBModel*  modelo_ip, vector<vector< vector<GRBVar> > > &x, vector
   <vector< GRBVar > > &y);
 void crear_restricciones(vector<double>  pesos, int grupos, int
   card_V, GRBEnv* env_modelo, GRBModel*  modelo_ip, int card_A,
```

```
    vector<vector< vector<GRBVar> > > x, vector<vector< GRBVar > > y);
 void crear_restricciones_sin_nodos_artificiales(vector<double>  pesos
    , int grupos, int card_V, GRBEnv* env_modelo, GRBModel*  modelo_ip
    , int card_A, vector<vector< vector<GRBVar> > > x, vector<vector<
    GRBVar > > y);
 void  presentar_sol(vector<vector< vector<GRBVar> > > x, vector<
    vector< GRBVar > > y, int card_V, int grupos, GRBEnv* env_modelo,
    GRBModel*  modelo_ip);
    // Variables globales
#define EPSILON 0.0000001
#define BIGVALUE 9999.9
    // Clase para la formulación de las desigualdades triangulares
 class   triangular_ineq
{
 public:
    int i, j, k, c;
    int sign_cij, sign_cik, sign_cjk;
    float dist;
    triangular_ineq()
    {
       int i = 0, j = 0, k = 0, c = 0;
       int sign_cij = 0, sign_cik = 0, sign_cjk = 0;
       float dist = 0;
    }
    friend ostream& operator<<(ostream& os, triangular_ineq ti)
    {
       os << "(" << ti.i << "," << ti.j << "," << ti.k << "," << ti.c
    << ")" << " signos (" << ti.sign_cij << "," << ti.sign_cik << ","
    << ti.sign_cjk << ")";
       return os;
    }
};
 bool wayToSort(triangular_ineq i, triangular_ineq j);
 void optimize1(vector<vector< vector<GRBVar> > > x, vector< vector<
    double> > Datos, int card_V, int grupo, int& t_ineq_count, vector<
    triangular_ineq>& t_ineq, GRBEnv* env_modelo, GRBModel*  modelo_ip
    );
    // Clase para la función callback
 class triineq_lazys : public GRBCallback
{
 public:
    vector<vector<vector<GRBVar> > >& vars; // Variables xcij
    vector<vector<GRBVar> >& yvars; // Variables yci
```

```
52      vector<triangular_ineq> t_ineq; // Vector de desigualdades
        triangulares
53      int& lazys_added, &lazys_called;
54     // lazys_added: Número de desigualdades Lazy agregadas, lazys_called
        :Número de desigualdades Lazy llamadas
55      int& part2_added, &cut_called;
56       // part2_added:Número de desigualdades 2 partición agregadas,
        cut_called:número de desigualdades llamadas
57      int& tricut_added; // Número de desigualdades triangulares
        agregadas
58      int& nodecnt; // Número de nodos de Branch and Cut explorados en la
         optimización más reciente.
59      int& cover_added; // Cortes sobre la variable y
60      int grupos; // Número de grupos a formarse
61      int card_V; // Nodos totales del grafo
62      int t_ineq_count; // Número de desigualdades triangulares creadas
63      vector<vector< vector<double> > > x; // Valores de la variable xcij
        (vars)
64      vector< vector<double> > y; // Valores de de la variable yci(yvars)
65      vector<double> wp; // Vector para los planos cortantes cover
66      triineq_lazys(vector<vector<vector<GRBVar> > >& xvars, vector<
        vector<GRBVar> >&  varsy, vector<double> pesos, vector<
        triangular_ineq> t_ineqx, int& lazys_addedx, int &lazys_calledx,
        int& part2_addedx, int &cut_calledx, int& tricut_addedx, int&
        nodecntx, int & cover_addedy) : vars(xvars), yvars(varsy), wp(
        pesos), t_ineq(t_ineqx), lazys_added(lazys_addedx), lazys_called(
        lazys_calledx), part2_added(part2_addedx), cut_called(cut_calledx)
        , tricut_added(tricut_addedx), nodecnt(nodecntx), cover_added(
        cover_addedy)
67      {
68         lazys_added = 0;
69         lazys_called = 0;
70         tricut_added = 0;
71         part2_added = 0;
72         cut_called = 0;
73         nodecnt = 0;
74         cover_added = 0;
75         t_ineq_count = t_ineqx.size();
76         t_ineq_count_during_cuts = 0;
77         grupos = yvars.size();
78         card_V = yvars[0].size();
79         x.resize(grupos);
80         for (unsigned int i = 0; i<x.size(); i++)
81         {
```

```
82            x[i].resize(card_V);
83            for (int j = 0; j < card_V; j++)
84            {
85               x[i][j].resize(card_V);
86            }
87         }
88         y.resize(grupos);
89         for (int i = 0; i < grupos; i++) {
90            y[i].resize(card_V);
91         }
92      }
93   protected:
94      // Planos cortantes
95      void callback();
96   };
97   #endif  PROBLEMAMIP_H
```

7.1.2 ProblemaMIP.cpp

```
1   #include "ProblemaMIP.h"
2   #include <iostream>
3   #include<fstream>
4   #include <cmath>
5   #include <algorithm>
6
7   using namespace std;
8
9      // Media
10  double media(vector<double>x) {
11     int n;
12     n = x.size();
13     double mu = 0;
14     for (int i = 0; i < n; i++) {
15        mu += x[i];
16     }
17     return (mu + 0.0) / n;
18  }
19     // Desviación estándar
20  double desviacion(vector<double>x) {
21     int n;
22     n = x.size();
23     double mu = media(x);
24     double des = 0;
```

```
25      for (int i = 0; i < n; i++) {
26          des += (x[i] - mu)*(x[i] - mu);
27      }
28      return sqrt((des + 0.0) / n);
29  }
30      // Matriz de costo
31   void escribir_matriz(vector<vector<double> > A)
32  {
33      int n;
34      n = A.size();
35      for (int i = 0; i<n; i++)
36      {
37          for (int j = 0; j<n; j++)
38          {
39              cout << A[i][j] << " ";
40          }
41          cout << endl;
42      }
43  }
44      // Leer instancia
45   void instancia(string Nombre_Instancia, bool incluir_artificiales,
        vector<double>&pesos, vector<vector<double> >&Distancias, int &n,
        int &k)
46  {
47      vector<double>aux_pesos;
48      vector<vector<double> >aux_distancia;
49      ifstream fa;
50      fa.open(Nombre_Instancia.c_str());
51      fa >> n;
52      cout << "Numero de nodos:" << "" << n << endl;
53      fa >> k;
54      cout << "Numero de cliques a formarse:" << "" << k << endl;
55      aux_distancia.resize(n);
56      aux_pesos.resize(n);
57      for (int i = 0; i<n; i++)
58      {
59          aux_distancia[i].resize(n);
60      }
61      for (int i = 0; i<n; i++)
62      {
63          fa >> aux_pesos[i];
64          for (int j = 1; j <= n; j++)
65          {
66              fa >> aux_distancia[i][j - 1];
```

```
67          }
68      }
69      fa.close();
70      cout << "Lectura de datos" << endl;
71      if ((n%k != 0) && (incluir_artificiales == true))
72      {
73          int N = k - (n%k); // Cardinalidad del conjunto de nodos
        ficticios: |A|
74          int nfic = n + N; // |V|=|V|+|A| nodos totales
75          Distancias.resize(nfic); // Matriz de costos con los nodos
        originales + los costos de los nodos ficticios
76          for (int i = 0; i<nfic; i++)
77          {
78              Distancias[i].resize(nfic, 0.0);
79          }
80                  pesos.resize(nfic, 0.0);
81          for (int i = 0; i<n; i++)
82          {
83              pesos[i] = aux_pesos[i];
84              for (int j = 0; j < n; j++)
85              {
86                  Distancias[i][j] = aux_distancia[i][j];
87              }
88          }
89      }
90      else
91      {
92          Distancias.resize(n);
93          for (int i = 0; i<n; i++)
94          {
95              Distancias[i].resize(n);
96          }
97          pesos.resize(n);
98          for (int i = 0; i<n; i++)
99          {
100             pesos[i] = aux_pesos[i];
101             for (int j = 0; j < n; j++)
102             {
103                 Distancias[i][j] = aux_distancia[i][j];
104             }
105         }
106     }
107 }
108     // Creación de variables
```

```
109  void crear_vars(int tipo_modelo, vector< vector<double> > Datos,
       vector<double> pesos, int card_V, int grupos, GRBEnv* env_modelo,
       GRBModel*  modelo_ip, vector<vector< vector<GRBVar> > > &x, vector
       <vector< GRBVar > > &y)
110  {
111     for (int l = 0; l < grupos; l++)
112     {
113        for (int i = 0; i< card_V; i++)
114        {
115           for (int j = i + 1; j < card_V; j++)
116           {
117              {
118                 ostringstream nombre_var;
119                 nombre_var << "x^" << l << "_" << i << "_" << j;
120                 x[l][i][j] = modelo_ip->addVar(0.0, 1.0, Datos[i][j],
       GRB_BINARY, nombre_var.str().c_str());
121              }
122           }
123        }
124     }
125     cout << "Creada variable x" << endl;
126     for (int l = 0; l<grupos; l++)
127     {
128        for (int i = 0; i<card_V; i++)
129        {
130           ostringstream nombre_var2;
131           nombre_var2 << "y^" << l << "_" << i;
132           y[l][i] = modelo_ip->addVar(0.0, 1.0, 0.0, GRB_BINARY,
       nombre_var2.str().c_str());
133        }
134     }
135     cout << "Creada variable y" << endl;
136     modelo_ip->update();
137     // Proceso de pre-procesamiento: Se genera una pre-asignación de
       los nodos con mayor peso
138     if (tipo_modelo != 0)
139     {
140        vector<double> pre(pesos.size());
141        vector<int> asignacion(grupos);
142        pre = pesos;
143        int mg;
144        double max_peso;
145        for (int g = 0; g<grupos; g++)
146        {
```

```
147             max_peso = -INF;
148             mg = -1;
149             for (unsigned int i = 0; i<pre.size(); i++)
150             {
151                 if (pre[i]>max_peso)
152                 {
153                     max_peso = pre[i];
154                     mg = i;
155                 }
156             }
157             asignacion[g] = mg;
158             pre[mg] = -INF;
159         }
160         for (int l = 0; l<grupos; l++)
161         {
162             for (int i = 0; i<card_V; i++)
163             {
164                 y[l][i].set(GRB_IntAttr_BranchPriority, 2);
165             }
166         }
167         y[0][asignacion[0]].set(GRB_DoubleAttr_LB, 1.0);
168         for (unsigned int i = 0; i<asignacion.size(); i++)
169         {
170             if (i>0)
171             {
172                 for (int l = 0; l <= i; l++)
173                 {
174                     y[l][asignacion[i]].set(GRB_IntAttr_BranchPriority, 4);
175                 }
176             }
177             for (int l = i + 1; l<grupos; l++)
178             {
179                 y[l][asignacion[i]].set(GRB_DoubleAttr_UB, 0.0);
180             }
181         }
182         for (int l = 0; l < grupos; l++)
183         {
184             for (int i = 0; i< card_V; i++)
185             {
186                 for (int j = i + 1; j < card_V; j++)
187                 {
188                     x[l][i][j].set(GRB_IntAttr_BranchPriority, 0);
189                 }
190             }
```

```
191         }
192         modelo_ip->update();
193     }
194 }
195     // Creación de restricciones para el modelo F1
196  void crear_restricciones(vector<double>  pesos, int grupos, int
    card_V, GRBEnv* env_modelo, GRBModel*  modelo_ip, int card_A,
    vector<vector< vector<GRBVar> > > x, vector<vector< GRBVar > > y)
197 {
198     // Restricción 1
199     for (int c = 0; c < grupos; c++)
200     {
201         ostringstream restriccion1;
202         restriccion1 << "Rest_1" << c;
203         GRBLinExpr res1 = 0;
204         for (int i = 0; i < card_V; i++) {
205             res1 += y[c][i];
206         }
207         modelo_ip->addConstr(res1, GRB_EQUAL, card_V / grupos,
    restriccion1.str());
208     }
209     // Restricción 2
210     for (int i = 0; i < card_V; i++)
211     {
212         ostringstream restriccion2;
213         restriccion2 << "Rest_2" << i;
214         GRBLinExpr res2 = 0;
215         for (int c = 0; c < grupos; c++) {
216             res2 += y[c][i];
217         }
218         modelo_ip->addConstr(res2, GRB_EQUAL, 1.0, restriccion2.str());
219     }
220     // Restricción 3
221     if (card_A > 0)
222     {
223         int n = card_V - card_A;
224         for (int c = 0; c < grupos; c++)
225         {
226             ostringstream restriccion3;
227             restriccion3 << "Rest_3" << c;
228             GRBLinExpr res3 = 0;
229             for (int i = n; i < card_V; i++) {
230                 res3 += y[c][i];
231             }
```

```
232             modelo_ip->addConstr(res3, GRB_LESS_EQUAL, 1.0, restriccion3.
    str());
233          }
234       }
235       // Restricción 4
236       double WL, WU;
237       WL = media(pesos)*((card_V + 0.0) / grupos) - desviacion(pesos);
238       WU = media(pesos)*((card_V + 0.0) / grupos) + desviacion(pesos);
239       for (int c = 0; c < grupos; c++) {
240          ostringstream restriccion4, restriccion4_1;
241          restriccion4 << "Rest_4" << c;
242          restriccion4_1 << "Rest_4_1" << c;
243          GRBLinExpr res4 = 0;
244          for (int i = 0; i < card_V; i++) {
245             res4 += pesos[i] * y[c][i];
246          }
247          modelo_ip->addConstr(res4, GRB_GREATER_EQUAL, WL, restriccion4.
    str());
248          modelo_ip->addConstr(res4, GRB_LESS_EQUAL, WU, restriccion4_1.
    str());
249       }
250       // Restricción 5
251       for (int c = 0; c < grupos; c++) {
252          for (int i = 0; i < card_V; i++) {
253             ostringstream restriccion5;
254             restriccion5 << "Rest_5" << c << "_" << i;
255             GRBLinExpr res5 = 0;
256             for (int j = 0; j <card_V; j++) {
257                if (i<j)
258                   res5 += x[c][i][j];
259                if (i>j)
260                   res5 += x[c][j][i];
261             }
262             modelo_ip->addConstr(res5, GRB_EQUAL, (((card_V + 0.0) /
    grupos) - 1) * y[c][i], restriccion5.str());
263          }
264       }
265       modelo_ip->update();
266    }
267       // Creación de restricciones para el modelo F2-R
268     void crear_restricciones_sin_nodos_artificiales(vector<double>  pesos
    , int grupos, int card_V, GRBEnv* env_modelo, GRBModel*  modelo_ip
    , int card_A, vector<vector< vector<GRBVar> > > x, vector<vector<
    GRBVar > > y)
```

```
{
    // Restricción 5
    for (int i = 0; i < card_V; i++)
    {
        ostringstream restriccion5;
        restriccion5 << "Rest_5_" << i;
        GRBLinExpr res5 = 0;
        for (int c = 0; c < grupos; c++)
        {
            res5 += y[c][i];
        }
        modelo_ip->addConstr(res5, GRB_EQUAL, 1.0, restriccion5.str());
    }
    // Restricción 6
    for (int c = 0; c < grupos; c++)
    {
        for (int i = 0; i < card_V; i++)
        {
            ostringstream restriccion6L, restriccion6U;
            restriccion6L << "Rest_6L_" << c << "_" << i;
            restriccion6U << "Rest_6U_" << c << "_" << i;
            GRBLinExpr res6 = 0;
            for (int j = 0; j <card_V; j++)
            {
                if (i<j)
                    res6 += x[c][i][j];
                if (i>j)
                    res6 += x[c][j][i];
            }
            modelo_ip->addConstr(res6, GRB_GREATER_EQUAL, (floor((card_V
    + 0.0) / grupos) - 1) * y[c][i], restriccion6L.str());
            modelo_ip->addConstr(res6, GRB_LESS_EQUAL,(ceil((card_V +
    0.0) / grupos) - 1) * y[c][i], restriccion6U.str());
        }
    }
    // Restricción 7
    double WL, WU;
    WL = media(pesos)*((card_V + 0.0) / grupos) - desviacion(pesos);
    WU = media(pesos)*((card_V + 0.0) / grupos) + desviacion(pesos);
    for (int c = 0; c < grupos; c++)
    {
        ostringstream restriccion7L, restriccion7U;
        restriccion7L << "Rest_7L_" << c;
        restriccion7U << "Rest_7U_" << c;
```

```
311         GRBLinExpr res7 = 0;
312         for (int i = 0; i < card_V; i++)
313         {
314            res7 += pesos[i] * y[c][i];
315         }
316         modelo_ip->addConstr(res7,GRB_GREATER_EQUAL, WL, restriccion7L.
    str());
317         modelo_ip->addConstr(res7,GRB_LESS_EQUAL, WU, restriccion7U.str
    ());
318      }
319      modelo_ip->update();
320   }
321      // Presentación de soluciones del modelo
322    void  presentar_sol(vector<vector< vector<GRBVar> > > x, vector<
       vector< GRBVar > > y, int card_V, int grupos, GRBEnv* env_modelo,
       GRBModel*  modelo_ip)
323   {
324      int st = modelo_ip->get(GRB_IntAttr_Status);
325      if ((st == 2) || (st >= 7) || (modelo_ip->get(GRB_DoubleAttr_MIPGap
       )<1e+20))
326      {
327         cout << "Solución Optima o factible" << endl;
328         cout << "status=" << st << endl
329               << "F objetivo=" << modelo_ip->get(GRB_DoubleAttr_ObjVal)
       << endl
330               << "Variables: " << modelo_ip->get(GRB_IntAttr_NumVars) <<
        endl
331               << "Restricciones: " << modelo_ip->get(
       GRB_IntAttr_NumConstrs) << endl
332               << "Tiempo=" << modelo_ip->get(GRB_DoubleAttr_Runtime) <<
       endl
333               << "Gap=" << 100 * modelo_ip->get(GRB_DoubleAttr_MIPGap)
       << endl;
334         cout << "\n\n";
335         for (int c = 0; c < grupos; c++)
336         {
337            for (int i = 0; i < card_V; i++)
338            {
339               for (int j = i + 1; j < card_V; j++)
340               {
341                  double val = 0.0;
342                  val = x[c][i][j].get(GRB_DoubleAttr_X);
343                  if (val > 0.1)
344                  {
```

```
345     string s = x[c][i][j].get(GRB_StringAttr_VarName);
346               cout << s << ":    " << val << endl;
347                  }
348               }
349            }
350            cout << endl << "....." << endl;
351         }
352         for (int c = 0; c < grupos; c++)
353         {
354            for (int i = 0; i < card_V; i++)
355            {
356               double val = 0.0;
357               val = y[c][i].get(GRB_DoubleAttr_X);
358               if (val > 0.1)
359               {
360                  string s = y[c][i].get(GRB_StringAttr_VarName);
361                  cout << s << ":    " << val << endl;
362               }
363            }
364         }
365         cout << endl << "....." << endl;
366      }
367      else
368         cout << "sin solución...." << endl;
369   }
370      bool wayToSort(triangular_ineq i, triangular_ineq j)
371      {
372         return i.dist > j.dist;
373      }
374      // Creación de desigualdades triangulares
375    void optimize1(vector<vector< vector<GRBVar> > > x, vector< vector<
       double> > Datos, int card_V, int grupo, int& t_ineq_count, vector<
       triangular_ineq>& t_ineq, GRBEnv* env_modelo, GRBModel*  modelo_ip
       ) {
376      t_ineq.resize(grupo*card_V *(card_V - 1)*(card_V - 2) / 2);
377      int t_ineq_count_during_cuts, ttcount = 0;
378      for (int c = 0; c < grupo; c++) {
379         for (int i = 0; i < card_V - 2; i++) {
380            for (int j = i + 1; j < card_V - 1; j++) {
381               for (int k = j + 1; k < card_V; k++) {
382      // Primera combinación
383      if (Datos[i][j] < BIGVALUE && Datos[i][k] < BIGVALUE) {
384         t_ineq[ttcount].i = i; t_ineq[ttcount].sign_cij = +1;
385         t_ineq[ttcount].j = j; t_ineq[ttcount].sign_cik = +1;
```

```
386         t_ineq[ttcount].k = k; t_ineq[ttcount].sign_cjk = -1;
387         t_ineq[ttcount].dist = Datos[i][j] + Datos[i][k];
388         t_ineq[ttcount].c = c;
389         ttcount++;
390                   }
391      // Segunda combinación
392      if (Datos[i][j] < BIGVALUE && Datos[j][k] < BIGVALUE) {
393         t_ineq[ttcount].i = i; t_ineq[ttcount].sign_cij = +1;
394         t_ineq[ttcount].j = j; t_ineq[ttcount].sign_cik = -1;
395         t_ineq[ttcount].k = k; t_ineq[ttcount].sign_cjk = +1;
396         t_ineq[ttcount].dist = Datos[i][j] + Datos[j][k];
397         t_ineq[ttcount].c = c;
398         ttcount++;
399                   }
400      // Tercera combinación
401      if (Datos[i][k] < BIGVALUE && Datos[j][k] < BIGVALUE){
402         t_ineq[ttcount].i = i; t_ineq[ttcount].sign_cij = -1;
403         t_ineq[ttcount].j = j; t_ineq[ttcount].sign_cik = +1;
404         t_ineq[ttcount].k = k; t_ineq[ttcount].sign_cjk = +1;
405         t_ineq[ttcount].dist = Datos[i][k] + Datos[j][k];
406          t_ineq[ttcount].c = c;
407         ttcount++;
408                   }
409               }
410            }
411         }
412      }
413      t_ineq_count = ttcount;
414      // Ordenamos las desigualdades
415      sort(t_ineq.begin(), t_ineq.end(), wayToSort);
416      cout << "Numero de desigualdades creadas: " << t_ineq_count << endl
      ;
417   }
418      //Función miembro de la clase callback
419    void triineq_lazys::callback() {
420      try {
421         if (where == GRB_CB_MIP)
422         {
423      nodecnt = (int)getDoubleInfo(GRB_CB_MIP_NODCNT);
424       if (nodecnt % 100 == 1){
425            {
426               cout << "CORTES: 2-part: " << part2_added << ", triang: "
      << tricut_added << ", cover=" << cover_added
427                  << " lazy=" << lazys_added << endl;
```

```
428         }
429      }
430      else if (where == GRB_CB_MIPSOL)
431      {
432   lazys_called++;
433         for (int l = 0; l < grupos; l++)
434         {
435            for (int i = 0; i < card_V; i++)
436            {
437               for (int j = 0; j < card_V; j++)
438               {
439                  if (i < j)
440                  {
441                     x[l][i][j] = getSolution(vars[l][i][j]);
442                     x[l][j][i] = x[l][i][j];
443                  }
444               }
445            }
446         }
447   for (int ttcount = 0; ttcount < t_ineq_count; ttcount++)
448         {
449            triangular_ineq ti;
450            ti = t_ineq[ttcount];
451            int i = ti.i;
452            int j = ti.j;
453            int k = ti.k;
454            int c = ti.c;
455            int sign_cij = ti.sign_cij;
456            int sign_cik = ti.sign_cik;
457            int sign_cjk = ti.sign_cjk;
458            float akku = 0.0;
459            if (sign_cij > 0)
460               akku += x[c][i][j];
461            else
462               akku -= x[c][i][j];
463            if (sign_cik > 0)
464               akku += x[c][i][k];
465            else
466               akku -= x[c][i][k];
467            if (sign_cjk > 0)
468               akku += x[c][j][k];
469            else
470               akku -= x[c][j][k];
471            if (akku > 1 + EPSILON)
```

```
472             {
473                 GRBLinExpr lazy = 0.0;
474                 if (sign_cij > 0)
475                 {
476                     if (i<j)
477                         lazy += vars[c][i][j];
478                     else
479                         lazy += vars[c][j][i];
480                 }
481                 else
482                 {
483                     if (i<j)
484                         lazy -= vars[c][i][j];
485                     else
486                         lazy -= vars[c][j][i];
487                 }
488                 if (sign_cik > 0)
489                 {
490                     if (i<k)
491                         lazy += vars[c][i][k];
492                     else
493                         lazy += vars[c][k][i];
494                 }
495                 else
496                 {
497                     if (i<k)
498                         lazy -= vars[c][i][k];
499                     else
500                         lazy -= vars[c][k][i];
501                 }
502                 if (sign_cjk > 0)
503                 {
504                     if (j<k)
505                         lazy += vars[c][j][k];
506                     else
507                         lazy += vars[c][k][j];
508                 }
509                     else
510                 {
511                     if (j<k)
512                         lazy -= vars[c][j][k];
513                     else
514                         lazy -= vars[c][k][j];
515                 }
```

```
516                addLazy(lazy <= 1);
517                lazys_added++;           }
518          }
519       }
520       else if (where == GRB_CB_MIPNODE)
521       {
522    if (getIntInfo(GRB_CB_MIPNODE_STATUS) == GRB_OPTIMAL)
523          {
524             cut_called++;
525    // Recuperación de los valores del vector de solución (vars[c][i][j
       ]-yvars[c][i]) del modelo
526             for (int l = 0; l < grupos; l++)
527             {
528                for (int i = 0; i < card_V; i++)
529                {
530       y[l][i] = getNodeRel(yvars[l][i]);
531                }
532             }
533       for (int l = 0; l < grupos; l++) {
534                for (int i = 0; i < card_V; i++) {
535                   for (int j = 0; j < card_V; j++)
536                      if (i < j) {
537       x[l][i][j] = getNodeRel(vars[l][i][j]);
538                x[l][j][i] = x[l][i][j];
539                      }
540                }
541             }
542    int tt_number = t_ineq_count_during_cuts;
543    if (nodecnt == 0) tt_number = t_ineq_count;
544    bool t_ineq_added = false, t_2part_added = false;
545    for (int ttcount = 0; ttcount < tt_number; ttcount++)
546             {
547                triangular_ineq ti;
548                ti = t_ineq[ttcount];
549                int i = ti.i;
550                int j = ti.j;
551                int k = ti.k;
552                int c = ti.c;
553                int sign_cij = ti.sign_cij;
554                int sign_cik = ti.sign_cik;
555                int sign_cjk = ti.sign_cjk;
556    float akku = 0.0;
557                if (sign_cij > 0) akku += x[c][i][j]; else akku -= x[c
       ][i][j];
```

```
558                 if (sign_cik > 0) akku += x[c][i][k]; else akku -= x[c
    ][i][k];
559                 if (sign_cjk > 0) akku += x[c][j][k]; else akku -= x[c
    ][j][k];
560                 if (akku > 1 + EPSILON)
561                 {
562                 GRBLinExpr cut = 0.0;
563                    if (sign_cij > 0) cut += vars[c][i][j]; else cut -=
    vars[c][i][j];
564                    if (sign_cik > 0) cut += vars[c][i][k]; else cut -=
    vars[c][i][k];
565                    if (sign_cjk > 0) cut += vars[c][j][k]; else cut -=
    vars[c][j][k];
566                    addCut(cut <= 1);
567                    tricut_added++;
568                    t_ineq_added = true;
569                 }
570              }
571              if (t_ineq_added == false)
572              {
573                 for (int l = 0; l < grupos; l++)
574                 {
575                    for (int v = 0; v < card_V; v++)
576                    {
577                       list<int> conj_W;
578                       for (int u = v + 1; u < card_V; u++)
579                       {
580                          double xluv = x[l][u][v];
581    if ((xluv > EPSILON) && (xluv < 1.0 - EPSILON))
582          conj_W.push_back(u);
583                       }
584    list<int> conj_T;
585                       if (conj_W.size() >= 5)
586                       {
587                                conj_T.clear();
588                          conj_T.push_back(conj_W.front());
589                          conj_W.pop_front();
590    for (list<int>::iterator itw = conj_W.begin(); itw != conj_W.end();
     itw++){
591              double sumax = 0;
592              int i, j;
593    for (list<int>::iterator itt = conj_T.begin(); itt != conj_T.end();
     itt++){
594              i = *itw;
```

```
595             j = *itt;
596             sumax += x[l][i][j];
597                             }
598             if (sumax <= EPSILON)
599                             {
600                conj_T.push_front(i);
601                             }
602                          }
603                       }
604          if (conj_T.size()>1)
605                       {
606                double x_v_T = 0;
607                list<int>::iterator itt, itt1;
608    for (itt = conj_T.begin(); itt != conj_T.end(); itt++)
609                          {
610                x_v_T = x_v_T + x[l][v][*itt];
611                          }
612             if (x_v_T <= 1)
613                          {
614             GRBLinExpr cut_2p = 0.0;
615                double Desig_2p = 0;
616    for (itt = conj_T.begin(); itt != conj_T.end(); itt++)
617                          {
618             Desig_2p = Desig_2p + x[l][v][*itt];
619          if (v<*itt)
620             cut_2p = cut_2p + vars[l][v][*itt];
621          else
622             cut_2p = cut_2p + vars[l][*itt][v];
623                       }
624    for (itt = conj_T.begin(); itt != conj_T.end(); itt++)
625                          {
626    for (itt1 = itt; itt1 != conj_T.end(); itt1++)
627                             {
628       Desig_2p = Desig_2p - x[l][*itt][*itt1];
629                if (*itt<*itt1)
630       cut_2p = cut_2p - vars[l][*itt][*itt1];
631                if (*itt>*itt1)
632          cut_2p = cut_2p - vars[l][*itt1][*itt];
633                             }
634                          }
635                   if (Desig_2p >= 1)
636                          {
637             addCut(cut_2p, GRB_LESS_EQUAL, 1);
638                   part2_added++;
```

```
                    t_2part_added = true;
                          }
                       }
                    }
                 }
              }
           }
if ((t_ineq_added == false) && (t_2part_added == false))
           {
              double WU, sumawp;
 WU = media(wp)*((card_V + 0.0) / grupos) + desviacion(wp);
for (unsigned int u1 = 0; u1<card_V - 3; u1++)
              {
 for (unsigned int u2 = u1 + 1; u2<card_V - 2; u2++)
                 {
  for (unsigned int u3 = u2 + 1; u3<card_V - 1; u3++)
                    {
      if (wp[u1] + wp[u2] + wp[u3]>WU)
                       {
                 sumawp = 0;
   for (unsigned int l = 0; l<grupos; l++)
                          {
   sumawp = y[l][u1] + y[l][u2] + y[l][u3];
              if (sumawp>2)
                             {
              GRBLinExpr cut_cover = 0.0;
   cut_cover = cut_cover + yvars[l][u1] + yvars[l][u2] + yvars[l][
u3];
      addCut(cut_cover <= 2);
         cover_added++;
                          }
                       }
                    }
for (unsigned int u4 = u3 + 1; u4<card_V; u4++)
                       {
   if (wp[u1] + wp[u2] + wp[u3] + wp[u4]>WU)
                          {
                 sumawp = 0;
      for (unsigned int l = 0; l<grupos; l++)
                             {
sumawp = y[l][u1] + y[l][u2] + y[l][u3] + y[l][u4];
                 if (sumawp>3)
                                {
          GRBLinExpr cut_cover = 0.0;
```

```
682         cut_cover = cut_cover + yvars[l][u1] + yvars[l][u2] + yvars[l][
    u3] + yvars[l][u4];
683           addCut(cut_cover <= 3);
684                 cover_added++;
685                                     }
686                                   }
687                                 }
688                               }
689                             }
690                           }
691                         }
692                       }
693                   }
694               }
695     }
696     catch (GRBException e) {
697         cout << "Error number: " << e.getErrorCode() << endl;
698         cout << e.getMessage() << endl;
699     }
700     catch (...) {
701         cout << "Error during callback" << endl;
702     }
703 }
```

7.1.3 Source.cpp

```
1  #include "ProblemaMIP.h"
2  #include <iostream>
3  #include <fstream>
4  using namespace std;
5
6  int main(int    argc, char *argv[])
7  {
8     cout << "   MODELO DE PARTICIONAMIENTO CON GUROBI" << endl << endl;
9     int TIEMPO_MAX = 7200;
10    vector<string> instancias;
11    instancias.resize(43);
12   instancias[0] = "random34_1.txt";
13    instancias[1] = "random34_2.txt";
14    instancias[2] = "random34_3.txt";
15    instancias[3] = "random34_4.txt";
16    instancias[4] = "random34_5.txt";
17    instancias[5] = "random34_6.txt";
```

```
    instancias[6] = "random34_7.txt";
    instancias[7] = "random34_8.txt";
    instancias[8] = "random34_9.txt";
    instancias[9] = "random34_10.txt";
    instancias[10] = "random38_1.txt";
    instancias[11] = "random38_2.txt";
    instancias[12] = "random38_3.txt";
    instancias[13] = "random38_4.txt";
    instancias[14] = "random38_5.txt";
    instancias[15] = "random38_6.txt";
    instancias[16] = "random38_7.txt";
    instancias[17] = "random38_8.txt";
    instancias[18] = "random38_9.txt";
    instancias[19] = "random38_10.txt";
    instancias[20] = "random44_1.txt";
    instancias[21] = "random44_2.txt";
    instancias[22] = "random44_3.txt";
    instancias[23] = "random44_4.txt";
    instancias[24] = "random44_5.txt";
    instancias[25] = "random44_6.txt";
    instancias[26] = "random44_7.txt";
    instancias[27] = "random44_8.txt";
    instancias[28] = "random44_9.txt";
    instancias[29] = "random44_10.txt";
    instancias[30] = "random54_1.txt";
    instancias[31] = "random54_2.txt";
    instancias[32] = "random54_3.txt";
    instancias[33] = "random54_4.txt";
    instancias[34] = "random54_5.txt";
    instancias[35] = "random54_6.txt";
    instancias[36] = "random54_7.txt";
    instancias[37] = "random54_8.txt";
    instancias[38] = "random54_9.txt";
    instancias[39] = "random54_10.txt";
    instancias[40] = "inst-2014.txt";
    instancias[41] = "inst-2015.txt";
    instancias[42] = "inst-2015-proposal.txt";
    vector<int> Modelo;
    Modelo.resize(3);
    Modelo[0] = 0;
    Modelo[1] = 1;
    Modelo[2] = 2;
       for (int i = 0; i < instancias.size(); i++)
       {
```

```
62          Nombre << instancias[i];
63          for (int j = 0; j < Modelo.size(); j++)
64          {
65             int TIPO_MODELO = Modelo[j];
66    // Valores de TIPO_MODELO:
67       // 0=modelo F1
68       // 1=modelo F1 + planos cortantes
69       // 2=modelo F2-R + planos cortantes
70             bool incluir_artificiales;
71             if ((TIPO_MODELO == 0) || (TIPO_MODELO == 1))
72                incluir_artificiales = true;
73             else
74                incluir_artificiales = false;
75        // Parámetros del modelo
76    cout << "Instancia:" << instancias[i] << endl;
77    cout << "Modelo: " << Modelo[j] << endl;
78             vector<vector<double> > Distancias; // Matriz de costos
      sobre las aristas
79    vector<double> pesos; // Matriz de pesos sobre los nodos
80    int k, n; // k:número de cliques a formarse,n: nodos originales
81    instancia(instancias[i], incluir_artificiales, pesos, Distancias, n
      , k); // Nos devuelve la matriz de costo y de peso
      correspondientes al grafo. Si es necesario construir el conjunto
      de nodos ficticios nos bota las matrices respectivas con esta
      nueva inclusión.
82    int nodo_tot = pesos.size(); // nodo_tot: nodos originales + nodos
      artificiales, si es el caso
83    cout << "Nodos totales: " << nodo_tot << endl;
84    int nodo_arti = nodo_tot - n;// Cardinalidad del conjunto de nodos
      ficticios
85    cout << "Nodos artificiales: " << nodo_arti << endl;
86    cout << "Nodos originales: " << n << endl;
87    cout << endl;
88    cout << "Datos Leidos" << endl;
89    try {
90       GRBEnv* env_modelo;
91             GRBModel*  modelo_ip;
92             env_modelo = new GRBEnv();
93             modelo_ip = new GRBModel(*env_modelo);
94    modelo_ip->getEnv().resetParams();
95    modelo_ip->getEnv().set(GRB_DoubleParam_TimeLimit, TIEMPO_MAX);
96    modelo_ip->getEnv().set(GRB_DoubleParam_NodefileStart, 0.5);
97    // Creación de la dimensión de las variable x,y
98             vector<vector< vector<GRBVar> > > x;
```

```
99                vector<vector< GRBVar > > y;
100               x.resize(k);
101         for (unsigned int i = 0; i < x.size(); i++)
102               {
103                  x[i].resize(nodo_tot);
104            for (int j = 0; j < nodo_tot; j++)
105                  {
106                     x[i][j].resize(nodo_tot);
107                  }
108               }
109               y.resize(k);
110               for (int i = 0; i < k; i++)
111               {
112                  y[i].resize(nodo_tot);
113               }
114   // Creación de variables
115 crear_vars(TIPO_MODELO, Distancias, pesos, nodo_tot, k, env_modelo,
    modelo_ip, x, y);
116   cout << " Variables creadas" << endl;
117   // Creación de restricciones
118   if (incluir_artificiales == true)
119 crear_restricciones(pesos, k, nodo_tot, env_modelo, modelo_ip,
    nodo_arti, x, y);
120               else
121 crear_restricciones_sin_nodos_artificiales(pesos, k, nodo_tot,
    env_modelo, modelo_ip, nodo_arti, x, y);
122   cout << " Restricciones creadas" << endl;
123   // Nombre del modelo
124 modelo_ip->set(GRB_StringAttr_ModelName, "Modelo de Particionamiento"
    );
125      modelo_ip->update();
126   // Devolución de valores de parámetros del modelo
127   cout << "Numero de variables: " << modelo_ip->get(
    GRB_IntAttr_NumVars) << endl;
128   cout << "Numero de restricciones: " << modelo_ip->get(
    GRB_IntAttr_NumConstrs) << endl;
129   cout << "Coeficientes distintos de cero: " << modelo_ip->get(
    GRB_IntAttr_NumNZs) << endl;
130   cout << "Nombre del modelo: " << modelo_ip->get(
    GRB_StringAttr_ModelName) << endl;
131   // Creación de desigualdades triangulares
132      int t_ineq_count;
133      vector<triangular_ineq> t_ineq;
```

```
134 optimize1(x, Distancias, nodo_tot, k, t_ineq_count, t_ineq, env_modelo
    , modelo_ip);
135    cout << "Creadas desigualdades" << endl;
136
137    int lazys_added = 0, lazys_called = 0, part2_added = 0, cut_called
    = 0, tricut_added = 0, nodecnt = 0, cover_added = 0;
138                ofstream f;
139                f.open("Reporte.csv", ios::app);
140                if (TIPO_MODELO != 0)
141                {
142    cout << "Llamando a planos cortantes" << endl;
143    // Activación de los planos cortantes
144    modelo_ip->getEnv().set(GRB_IntParam_PreCrush, 1);
145    // Activación de restricciones tipo Lazy
146    modelo_ip->getEnv().set(GRB_IntParam_LazyConstraints, 1);
147    // Desabilitación de cortes de Gurobi
148    modelo_ip->getEnv().set(GRB_IntParam_CliqueCuts, 0);
149    modelo_ip->getEnv().set(GRB_IntParam_FlowCoverCuts, 0);
150    modelo_ip->getEnv().set(GRB_IntParam_FlowPathCuts, 0);
151    modelo_ip->getEnv().set(GRB_IntParam_ImpliedCuts, 0);
152    modelo_ip->getEnv().set(GRB_IntParam_MIPSepCuts, 0);
153    modelo_ip->getEnv().set(GRB_IntParam_NetworkCuts, 0);
154    modelo_ip->getEnv().set(GRB_IntParam_ModKCuts, 0);
155    modelo_ip->getEnv().set(GRB_IntParam_GUBCoverCuts, 0);
156    modelo_ip->getEnv().set(GRB_IntParam_SubMIPCuts, 0);
157    cout << "Planos cortantes llamados" << endl;
158    //Creación de la subclase tipo callback y conección con el modelo
159    triineq_lazys cb = triineq_lazys(x, y, pesos, t_ineq, lazys_added,
    lazys_called, part2_added, cut_called, tricut_added, nodecnt,
    cover_added);
160    modelo_ip->setCallback(&cb);
161    cout << "Modelo asociado con callback" << endl;
162    // Resolver el modelo
163    modelo_ip->optimize();
164    cout << "Modelo Optmizado" << endl;
165    cout << "CORTES TOTATES:" << endl
166       << "2-part=" << part2_added << ", triang=" << tricut_added << ",
    cover=" << cover_added << endl
167       << "lazy=" << lazys_added << ", Nodos B&B=" << nodecnt << endl;
168    f << instancias[i]<< ";" << Modelo[j] << ";" << modelo_ip-> get(
    GRB_DoubleAttr_ObjVal) << ";" << modelo_ip-> get(
    GRB_DoubleAttr_Runtime) << ";" << 100 * modelo_ip-> get(
    GRB_DoubleAttr_MIPGap) << ";" << part2_added << ";" <<
    tricut_added << ";" << cover_added << ";" << lazys_added <<endl;
```

```
169                 }
170                 else
171                 {
172       modelo_ip->optimize();
173       cout << "Modelo Optmizado" << endl;
174     f << instancias[i]<< ";" << Modelo[j] << ";" << modelo_ip-> get(
      GRB_DoubleAttr_ObjVal) << ";" << modelo_ip-> get(
      GRB_DoubleAttr_Runtime)  << ";" << 100 * modelo_ip-> get(
      GRB_DoubleAttr_MIPGap) << endl;
175                 }
176                 f.close();
177     // Presentación de la solución
178   presentar_sol(x, y, nodo_tot, k, env_modelo, modelo_ip);
179     // Eliminación de estructuras
180     for (unsigned int i = 0; i < x.size(); i++)
181                 {
182      for (unsigned int j = 0; j < x[i].size(); j++)
183                    {
184                    x[i][j].clear();
185                    }
186                    x[i].clear();
187                 }
188                 x.clear();
189                 for (unsigned int i = 0; i < y.size(); i++)
190                 y[i].clear();
191                 y.clear();
192                 delete modelo_ip;
193                 delete env_modelo;
194              }
195              catch (...) {
196                 cout << "Exception during optimization" << endl;
197              }
198           }
199        }
200     return 0;
```

7.2 Código ($\mathcal{F}_2$)

7.2.1 ProblemaMIP2.h

```
1  #pragma once
2  #ifndef PROBLEMAMIP2_H
```

```
#define PROBLEMAMIP2_H

#include <iostream>
#include <cstdlib>
#include <sstream>
#include <gurobi_c++.h>
#include <vector>
#include <list>
#include <queue>
#include <stack>
#include <bitset>
#include <cmath>
#include <time.h>
#include<iomanip>

using namespace std;
 const double INF = 1e+20;
 void instancia(string Nombre_Instancia, vector<double>&pesos, vector<
   vector<double>>&Distancias, int &k);
 double media(vector<double>x);
 double desviacion(vector<double>x);
 void escribir_matriz(vector<vector<double>> A);
 void crear_vars(vector< vector<double> > Datos, vector<double> pesos,
    int card_V, int grupos, GRBEnv* env_modelo, GRBModel*  modelo_ip,
    vector<vector< vector<GRBVar> > > &x, vector<vector< GRBVar > > &
   y);
 void crear_restricciones(vector<double>  pesos, int grupos, int
   card_V, GRBEnv* env_modelo, GRBModel*  modelo_ip, vector<vector<
   vector<GRBVar> > > x, vector<vector< GRBVar > > y);
 void  presentar_sol(vector<vector< vector<GRBVar> > > x, vector<
   vector< GRBVar > > y, int card_V, int grupos, GRBEnv* env_modelo,
   GRBModel*  modelo_ip);
#endif
```

7.2.2 ProblemaMIP2.cpp

```
#include "ProblemaMIP2.h"
#include <iostream>

using namespace std;
   // Media
 double media(vector<double>x) {
   int n;
```

```
8     n = x.size();
9     double mu = 0;
10    for (int i = 0; i < n; i++) {
11       mu += x[i];
12    }
13    return (mu + 0.0) / n;
14 }
15    // Desviación estándar
16  double desviacion(vector<double>x) {
17    int n;
18    n = x.size();
19    double mu = media(x);
20    double des = 0;
21    for (int i = 0; i < n; i++) {
22       des += (x[i] - mu)*(x[i] - mu);
23    }
24
25    return sqrt((des + 0.0) / n);
26 }
27    // Leer instancia
28  void instancia(string Nombre_Instancia, vector<double>&pesos, vector<
      vector<double>>&Distancias, int &k)
29 {
30    int n;
31    ifstream fa;
32    fa.open(Nombre_Instancia.c_str());
33       fa >> n;
34       cout << "Numero de nodos:" << "" << n << endl;
35       fa >> k;
36       cout << "Numero de cliques a formarse:" << "" << k << endl;
37       Distancias.resize(n);
38       pesos.resize(n);
39       for (int i = 0; i < n; i++)
40       {
41          Distancias[i].resize(n);
42       }
43       for (int i = 0; i < n; i++)
44       {
45          fa >> pesos[i];
46          for (int j = 1; j <= n; j++)
47          {
48             fa >> Distancias[i][j - 1];
49          }
50       }
```

```
51          fa.close();
52  }
53      // Creación de variables
54   void crear_vars(vector< vector<double> > Datos, vector<double> pesos,
        int card_V, int grupos, GRBEnv* env_modelo, GRBModel*  modelo_ip,
        vector<vector< vector<GRBVar> > > &x, vector<vector< GRBVar > > &
       y)
55  {
56      unsigned int n;
57      n = Datos.size();
58      for (int l = 0; l < grupos; l++) {
59          for (int i = 0; i < card_V; i++) {
60              for (int j = 0; j < card_V; j++) {
61                  if (i < j)
62                  {
63                      ostringstream nombre_var;
64     nombre_var << "x^" << l << "_" << i << "_" << j;
65     x[l][i][j] = modelo_ip->addVar(0.0, 1.0, Datos[i][j], GRB_BINARY,
        nombre_var.str().c_str());
66                  }
67              }
68          }
69      }
70      for (int l = 0; l<grupos; l++)
71      {
72          for (int i = 0; i<n; i++)
73          {
74              ostringstream nombre_var2;
75              nombre_var2 << "y_" << l << "^" << i;
76     y[l][i] = modelo_ip->addVar(0.0, 1.0, 0.0, GRB_BINARY, nombre_var2.
        str().c_str());
77          }
78      }
79      modelo_ip->update();
80  }
81   void crear_restricciones(vector<double>  pesos, int grupos, int
       card_V, GRBEnv* env_modelo, GRBModel*  modelo_ip, vector<vector<
       vector<GRBVar> > > x, vector<vector< GRBVar > > y)
82  {
83      // Restricción 1
84      for (unsigned int c = 0; c < grupos; c++) {
85          for (unsigned int i = 0; i < card_V; i++)
86          {
87              for (unsigned int j = i + 1; j < card_V; j++)
```

```
88          {
89              for (unsigned int l = j + 1; l < card_V; l++)
90              {
91                  ostringstream restriccion1, restriccion2, restriccion3;
92                  restriccion1 << "Rest_11" << "_" << c << "_" << i << "_
    " << j << "_" << l;
93                  restriccion2 << "Rest_12" << "_" << c << "_" << i << "_
    " << j << "_" << l;
94                  restriccion3 << "Rest_13" << "_" << c << "_" << i << "_
    " << j << "_" << l;
95                  GRBLinExpr res1 = 0, res2 = 0, res3 = 0;
96                  res1 = x[c][i][j] + x[c][j][l] - x[c][i][l];
97                  modelo_ip->addConstr(res1, GRB_LESS_EQUAL, 1.0,
    restriccion1.str());
98                  res2 = x[c][i][j] - x[c][j][l] + x[c][i][l];
99                  modelo_ip->addConstr(res2, GRB_LESS_EQUAL, 1.0,
    restriccion2.str());
100                 res3 = -x[c][i][j] + x[c][j][l] + x[c][i][l];
101                 modelo_ip->addConstr(res3, GRB_LESS_EQUAL, 1.0,
    restriccion3.str());
102             }
103         }
104     }
105 }
106 // Restricción 4
107 for (unsigned int i = 0; i < card_V; i++) {
108     ostringstream restriccion4;
109     restriccion4 << "Rest_4" << "_" << i;
110     GRBLinExpr res4 = 0;
111     for (unsigned int c = 0; c < grupos; c++) {
112         res4 += y[c][i];
113     }
114     modelo_ip->addConstr(res4, GRB_EQUAL, 1.0, restriccion4.str());
115 }
116 // Restricción 5
117 for (unsigned int c = 0; c < grupos; c++) {
118     for (unsigned int i = 0; i < card_V; i++)
119     {
120         ostringstream restriccion51, restriccion52;
121         restriccion51 << "Rest_5U" << "_" << c << "_" << i;
122         restriccion52 << "Rest_5L" << "_" << c << "_" << i;
123         GRBLinExpr res51 = 0;
124         for (unsigned int j = 0; j < card_V; j++)
125         {
```

```
126             if (i<j)
127                res51 += x[c][i][j];
128             if (i>j)
129                res51 += x[c][j][i];
130          }
131          modelo_ip->addConstr(res51, GRB_GREATER_EQUAL, (floor((card_V
     + 0.0) / grupos) - 1)*y[c][i], restriccion51.str());
132          modelo_ip->addConstr(res51, GRB_LESS_EQUAL, (ceil((card_V +
    0.0) / grupos) - 1)*y[c][i], restriccion52.str());
133       }
134    }
135    // Restricción 7
136    double WU, WL;
137    WL = media(pesos)*((card_V + 0.0) / grupos) - desviacion(pesos);
138    WU = media(pesos)*((card_V + 0.0) / grupos) + desviacion(pesos);
139    for (unsigned int c = 0; c < grupos; c++) {
140       ostringstream restriccion7, restriccion7_1;
141       restriccion7 << "Rest_7" << c;
142       restriccion7_1 << "Rest_7_1" << c;
143       GRBLinExpr res7 = 0;
144       for (unsigned int i = 0; i < card_V; i++) {
145          res7 += pesos[i] * y[c][i];
146       }
147       modelo_ip->addConstr(res7, GRB_GREATER_EQUAL, WL, restriccion7.
    str());
148       modelo_ip->addConstr(res7, GRB_LESS_EQUAL, WU, restriccion7_1.
    str());
149    }
150    modelo_ip->update();
151 }
152  void  presentar_sol(vector<vector< vector<GRBVar> > > x, vector<
    vector< GRBVar > > y, int card_V, int grupos, GRBEnv* env_modelo,
    GRBModel*  modelo_ip)
153 {
154    int st = modelo_ip->get(GRB_IntAttr_Status);
155    if ((st == 2) || (st >= 7) || (modelo_ip->get(GRB_DoubleAttr_MIPGap
    )<1e+20))
156    {
157       cout << "Solución Optima o factible" << endl;
158       cout << "status=" << st << endl
159          << "F objetivo=" << modelo_ip->get(GRB_DoubleAttr_ObjVal) <<
    endl
160          << "Variables: " << modelo_ip->get(GRB_IntAttr_NumVars) <<
    endl
```

```
161         << "Restricciones: " << modelo_ip->get(GRB_IntAttr_NumConstrs
    ) << endl
162         << "Tiempo=" << modelo_ip->get(GRB_DoubleAttr_Runtime) <<
    endl
163         << "Gap=" << 100 * modelo_ip->get(GRB_DoubleAttr_MIPGap) <<
    endl
164             << "Nodos B&B=" << nodecnt << endl;
165       cout << "\n\n";
166          for (unsigned int c = 0; c < grupos; c++)
167       {
168          for (unsigned int i = 0; i < card_V; i++)
169          {
170             for (unsigned int j = i + 1; j < card_V; j++)
171             {
172                double val = 0.0;
173                val = x[c][i][j].get(GRB_DoubleAttr_X);
174                if (val > 0.1)
175                {
176                   string s = x[c][i][j].get(GRB_StringAttr_VarName);
177                   cout << s << ":   " << val << endl;
178                }
179             }
180          }
181          cout << endl << "....." << endl;
182       }
183       for (unsigned int c = 0; c < grupos; c++)
184       {
185          for (unsigned int i = 0; i < card_V; i++)
186          {
187             double val = 0.0;
188             val = y[c][i].get(GRB_DoubleAttr_X);
189             if (val > 0.1)
190             {
191                string s = y[c][i].get(GRB_StringAttr_VarName);
192                cout << s << ":   " << val << endl;
193             }
194          }
195       }
196       cout << endl << "....." << endl;
197    }
198    else
199    {
200       cout << "sin solución...." << endl;
201    }
```

```
}
```

7.2.3 Source.cpp

```
#include "ProblemaMIP2.h"
#include <iostream>

using namespace std;

int main()
{
    cout << "   MODELO DE PARTICIONAMIENTO CON GUROBI" << endl << endl;
    int TIEMPO_MAX = 7200;
    vector<string> instancias;
    instancias.resize(43);
    instancias[0] = "random34_1.txt";
    instancias[1] = "random34_2.txt";
    instancias[2] = "random34_3.txt";
    instancias[3] = "random34_4.txt";
    instancias[4] = "random34_5.txt";
    instancias[5] = "random34_6.txt";
    instancias[6] = "random34_7.txt";
    instancias[7] = "random34_8.txt";
    instancias[8] = "random34_9.txt";
    instancias[9] = "random34_10.txt";
    instancias[10] = "random38_1.txt";
    instancias[11] = "random38_2.txt";
    instancias[12] = "random38_3.txt";
    instancias[13] = "random38_4.txt";
    instancias[14] = "random38_5.txt";
    instancias[15] = "random38_6.txt";
    instancias[16] = "random38_7.txt";
    instancias[17] = "random38_8.txt";
    instancias[18] = "random38_9.txt";
    instancias[19] = "random38_10.txt";
    instancias[20] = "random44_1.txt";
    instancias[21] = "random44_2.txt";
    instancias[22] = "random44_3.txt";
    instancias[23] = "random44_4.txt";
    instancias[24] = "random44_5.txt";
    instancias[25] = "random44_6.txt";
    instancias[26] = "random44_7.txt";
    instancias[27] = "random44_8.txt";
```

```
    instancias[28] = "random44_9.txt";
    instancias[29] = "random44_10.txt";
    instancias[30] = "random54_1.txt";
    instancias[31] = "random54_2.txt";
    instancias[32] = "random54_3.txt";
    instancias[33] = "random54_4.txt";
    instancias[34] = "random54_5.txt";
    instancias[35] = "random54_6.txt";
    instancias[36] = "random54_7.txt";
    instancias[37] = "random54_8.txt";
    instancias[38] = "random54_9.txt";
    instancias[39] = "random54_10.txt";
    instancias[40] = "inst-2014.txt";
    instancias[41] = "inst-2015.txt";
    instancias[42] = "inst-2015-proposal.txt";
    ofstream f;
    f.open("Reporte.csv");
    f << "Instancias" << ";" << "Modelo" << ";" << "FuncionObjetivo" <<
     "; " << "Tiempo" << "; " << "Gap" << "Nodos B&B" << endl;
    f.close();
    cout << "   MODELO DE PARTICIONAMIENTO CON GUROBI" << endl << endl;
    // Parámetros del modelo
     for (int i = 0; i < instancias.size(); i++)
    {
       cout << "Instancia:" << instancias[i] << endl;
       cout << "Tiempo: " << TIEMPO_MAX << endl;
       vector<vector<double>> Distancias; // Matriz de costos sobre las
     aristas
       vector<double> pesos; // Matriz de pesos sobre los nodos
       int k;
       instancia(instancias[i],pesos, Distancias, k); // Devuelve la
    matriz de costos y pesos correspondientes al grafo.
       int nodos = pesos.size();
       cout << "Datos Leidos" << endl;
       try {
          GRBEnv* env_modelo;
          GRBModel*  modelo_ip;
          env_modelo = new GRBEnv();
          modelo_ip = new GRBModel(*env_modelo);
          modelo_ip->getEnv().resetParams();
          modelo_ip->getEnv().set(GRB_DoubleParam_TimeLimit, TIEMPO_MAX
    );
       modelo_ip->getEnv().set(GRB_DoubleParam_NodefileStart, 0.5);
    // Creando dimensión de variables x-y
```

```
80          vector<vector< vector<GRBVar> > > x;
81          vector<vector< GRBVar > > y;
82          x.resize(k);
83          for (unsigned int i = 0; i<x.size(); i++)
84          {
85             x[i].resize(nodos);
86             for (unsigned int j = 0; j < nodos; j++)
87             {
88                x[i][j].resize(nodos);
89             }
90          }
91          y.resize(k);
92          for (int i = 0; i< k; i++)
93             y[i].resize(nodos);
94    // Creación de Variables
95          crear_vars(Distancias, pesos, nodos, k, env_modelo, modelo_ip
    , x, y);
96          cout << " Variables creadas" << endl;
97    // Creación de restricciones
98          crear_restricciones(pesos, k, nodos, env_modelo, modelo_ip, x
    , y);
99          cout << " Restricciones creadas" << endl;
100         modelo_ip->set(GRB_StringAttr_ModelName, "Modelo de
    Particionamiento");
101         modelo_ip->update();
102         cout << "Numero de variables: " << modelo_ip->get(
    GRB_IntAttr_NumVars) << endl;
103         cout << "Numero de restricciones: " << modelo_ip->get(
    GRB_IntAttr_NumConstrs) << endl;
104         cout << "Coeficientes distintos de cero: " << modelo_ip->get(
    GRB_IntAttr_NumNZs) << endl;
105         cout << "Nombre del modelo: " << modelo_ip->get(
    GRB_StringAttr_ModelName) << endl;
106            modelo_ip->getEnv().set(GRB_DoubleParam_TimeLimit,
    TIEMPO_MAX);
107         modelo_ip->getEnv().set(GRB_DoubleParam_MIPGap, GAP);
108   // Desactivación de cortes de Gurobi
109      modelo_ip->getEnv().set(GRB_IntParam_CliqueCuts, 0);
110     modelo_ip->getEnv().set(GRB_IntParam_FlowCoverCuts, 0);
111     modelo_ip->getEnv().set(GRB_IntParam_FlowPathCuts, 0);
112     modelo_ip->getEnv().set(GRB_IntParam_ImpliedCuts, 0);
113     modelo_ip->getEnv().set(GRB_IntParam_MIPSepCuts, 0);
114     modelo_ip->getEnv().set(GRB_IntParam_NetworkCuts, 0);
115     modelo_ip->getEnv().set(GRB_IntParam_ModKCuts, 0);
```

```
116       modelo_ip->getEnv().set(GRB_IntParam_GUBCoverCuts, 0);
117       modelo_ip->getEnv().set(GRB_IntParam_SubMIPCuts, 0);
118       modelo_ip->update();
119     // Resolver el modelo
120           modelo_ip->optimize();
121     // Creando archivo resumen
122           cout << "Modelo Optmizado" << endl;
123           f.open("Reporte.csv", ios::app);
124                   f << instancias[i] << ";" << "2" << ";" << modelo_ip
      ->get(GRB_DoubleAttr_ObjVal) << "; " << modelo_ip->get(
      GRB_DoubleAttr_Runtime) << "; " <<
125           100 * modelo_ip->get(GRB_DoubleAttr_MIPGap) << modelo_ip->
      get(GRB_DoubleAttr_NodeCount) << endl;
126           f.close();
127     // Presentación de la solución
128        presentar_sol(x, y, nodos, k, env_modelo, modelo_ip);
129     /// Eliminación de estructuras
130           for (unsigned int i = 0; i<x.size(); i++)
131           {
132              for (unsigned int j = 0; j < x[i].size(); j++)
133              {
134                 x[i][j].clear();
135              }
136              x[i].clear();
137           }
138           x.clear();
139           for (unsigned int i = 0; i<y.size(); i++)
140           {
141              y[i].clear();
142           }
143           y.clear();
144           delete modelo_ip;
145           delete env_modelo;
146        }
147        catch (...) {
148           cout << "Exception during optimization" << endl;
149        }
150     }
151     system("pause");
152     return 0;
153 }
```