

ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE CIENCIAS

MÉTODOS EXACTOS PARA EL PROBLEMA DE EQUIPARTICIONAMIENTO DE GRAFOS EN COMPONENTES CONEXAS

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE
INGENIERO MATEMÁTICO

PROYECTO DE INVESTIGACIÓN

ESTÉFANO EDUARDO VITERI NEGRETE
estefano.viteri@epn.edu.ec

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

QUITO, JULIO 2021

DECLARACIÓN

Yo ESTÉFANO EDUARDO VITERI NEGRETE, declaro bajo juramento que el trabajo aquí escrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y que he consultado las referencias bibliográficas que se incluyen en este documento.

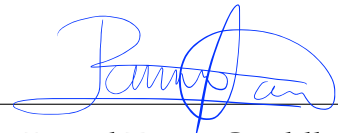
A través de la presente declaración cedo mis derechos de propiedad intelectual, correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su reglamento y por la normatividad institucional vigente.

A handwritten signature in black ink, enclosed within a hand-drawn oval. The signature is stylized and appears to read 'ESTEFANO V.'.

Estéfano Eduardo Viteri Negrete

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por ESTÉFANO EDUARDO VITERI NEGRETE, bajo mi supervisión.



Ramiro Daniel Torres Gordillo, PH.D.

Director del Proyecto

AGRADECIMIENTOS

A mi tutor, Ramiro, por el tiempo y esfuerzo dedicado en este proyecto. También, por haber sido la persona que me mostró que la investigación científica puede ser parte de mi vida.

A La Banda de los Gatos, por ser mis compañeros incondicionales en esta etapa universitaria, sin su apoyo este camino no hubiese sido el mismo. Gracias por todas las historias y logros que hemos conseguido juntos.

A mis abuelos, tíos y primos, por los consejos y reflexiones que han podido transmitirme. Sepan que han sido de gran ayuda y me han permitido llegar hasta este momento.

"Ya no quiero ser sólo un sobreviviente
quiero elegir el día para mi muerte.
Tengo la carne joven, roja la sangre,
la dentadura buena y mi esperma urgente."

(VICTOR HEREDIA)

DEDICATORIA

A mi madre, Maggy, por enseñarme que la perseverancia y el amor se cultivan día a día en todos los ámbitos y etapas de la vida. A mi padre, José, por enseñarme a cuestionar todo a mi alrededor mientras me mostraba el maravilloso camino de la matemática.

A mis hermanos, José y Darío, que con su ejemplo me muestran que el valor de la familia es incalculable y que la vida es hermosa cuando somos felices. A mis cuñadas, Tami y Cris, que me han brindado su afecto y cuidados a lo largo de estos años.

A mis sobrinos José, Alejo, Martina y Felipe, que con su alegría y carisma me concedieron la fuerza para no rendirme nunca.

A Pamela, que es partícipe de esta travesía brindándome su amor y paciencia en todo momento.

*"Yo bailo con mi canción
y no con la que me tocan.
Yo no soy la libertad
pero sí el que la provoca.
Si ya conozco el camino
para que voy andar acostado.
Si la libertad me gusta
para que he de vivir de esclavo."*

(FACUNDO CABRAL)

Índice general

Resumen	IX
Abstract	X
Notaciones	XI
1. Introducción	1
2. Definiciones Preliminares	4
2.1. Teoría de grafos	4
2.2. Programación Lineal Entera	7
2.3. Particionamiento de Grafos	9
3. Problema de equiparticionamiento de grafos	12
3.1. Grafos Completos	13
3.1.1. Primer modelo	13
3.1.2. Segundo modelo	14
3.2. Grafos Generales	16
3.2.1. Primer modelo	17
3.2.2. Segundo modelo	18
3.2.3. Tercer modelo	20
4. Desigualdades válidas	24
5. Resultados Computacionales	30
5.1. Equipartición de grafos	31

5.2. Equipartición de grafos en componentes conexas	32
6. Conclusiones y Recomendaciones	42
Bibliografía	44
Anexos	46

Índice de cuadros

3.1. Información de tamaño de los modelos \mathcal{FC} -1 y \mathcal{FC} -2.	16
3.2. Información de tamaño de los modelos \mathcal{FG} -1, \mathcal{FG} -2 y \mathcal{FG} -3.	23
5.1. Comparación de los modelos \mathcal{FC} -1 y \mathcal{FC} -2.	31
5.2. Instancia: $n = 15, k = 6, d = 0.92$	34
5.3. Instancia: $n = 15, k = 6, d = 0.31$	35
5.4. $n = 20, k = 5, d = 0.91$	36
5.5. $n = 20, k = 5, d = 0.36$	37
5.6. Comparación de los modelos \mathcal{FG} -1 y \mathcal{FG} -3.	37
5.7. Modelo \mathcal{FG} -1 para $n = 40$	38
5.8. Modelo \mathcal{FG} -1 para $n = 45$	39
5.9. Modelo \mathcal{FG} -1 para $n = 50$	39
5.10. Modelo \mathcal{FG} -1 para $n = 55$	40
5.11. Modelo \mathcal{FG} -1 para $n = 60$	40

Resumen

En el presente trabajo, se estudia problema de particionamiento de grafos en componentes conexas. El problema consiste en particionar un grafo no dirigido con costos sobre las aristas en un número fijo de componentes conexas, tal que el número de nodos en cada componente difiera en a lo más una unidad y el costo total de las aristas con nodos finales en el mismo subconjunto de nodos que induce la componente sea minimizado. Se presentan varios modelos de programación lineal entera usando diferentes enfoques (maximización de los costos de las aristas del corte y minimización de los costos de las aristas en cada componente conexa) y sus resultados son comparados. Además, se exponen familias de desigualdades válidas asociadas a los poliedros de estas formulaciones, junto con un algoritmo exacto tipo Branch & Cut para el problema estudiado. Se reportan resultados computacionales basados en instancias simuladas de diferentes tamaños y densidades. Finalmente, se presentan conclusiones sobre el presente trabajo.

Palabras clave: Equipartición de grafos, conectividad, programación lineal entera, desigualdades válidas.

Abstract

In the present work, the graph partitioning problem in connected components is studied. The problem consists of partitioning an undirected graph with cost on the edges into a fixed number of connected components, such that the number of nodes in each component differs by at most one unit and the total cost of the edges with end-nodes in the same subset of nodes that induces a component is minimized. Several integer linear programming models using different approaches (maximizing the edges in the cut or minimizing the edges in the connected components) are presented and the results are compared. Moreover, several families of valid inequalities associated to the polytope of these formulations are exposed, together with a Branch & Cut algorithm for the studied problem. Computational results based on simulated instances of different sizes and densities are reported. Finally, conclusions about the present work are presented.

Notaciones

V	Conjunto de nodos.
E	Conjunto de aristas.
A	Conjunto de arcos.
n	Número de nodos ($ V $).
m	Número de aristas ($ E $).
k	Número de particiones.
$[k] = \{1, \dots, k\}$	Conjunto de particiones.
$G = (V, E)$	Grafo no dirigido.
$D = (V, A)$	Grafo dirigido.
$K_n = (V_n, E_n)$	Grafo completo.
$\{u, v\}$	Arista entre el nodo u y el nodo v .
(u, v)	Arco donde el nodo u es el predecesor del nodo v .
$\delta_u := \{\{u, v\} : \{u, v\} \in E\}$	Conjunto de aristas incidentes al nodo $u \in V$.
$\delta_u^+ := \{(u, v) : (u, v) \in A\}$	Conjunto de arcos entrantes al nodo $u \in V$.
$\delta_u^- := \{(u, v) : (u, v) \in A\}$	Conjunto de arcos salientes del nodo $u \in V$.

Capítulo 1

Introducción

A través de los años, el mundo se ha visto en la obligación de trasladar los problemas reales al universo de las matemáticas y mediante todas las técnicas que esta dispone, buscar las mejores soluciones a los mismos. Una herramienta ampliamente utilizada es la teoría de grafos, donde muchos problemas de varias disciplinas (transporte, deporte, redes, logística, entre otras) pueden ser representados como una de estas estructuras. Por ejemplo, en el caso de un sistema de transporte público es posible asociar las estaciones de buses como nodos del grafo y se incluye una arista si existe una ruta entre las estaciones asociadas a un par de vértices. De igual forma, en el transporte aéreo un nodo ofrece la posibilidad de representar un aeropuerto en alguna ciudad del mundo, y en este caso, la inclusión de una arista implica la existencia de una ruta entre dos terminales aéreas.

Un problema clásico de esta teoría es el problema de particionar un grafo en varias partes. Durante los últimos años, muchas variantes han aparecido dependiendo del número de partes requeridas, pesos sobre los nodos o aristas, o características que se deben respetar en cada una de las particiones. Un problema específico consiste en particionar los nodos de un grafo de modo que el número de nodos en cada partición no exceda al resto de partes en más de una unidad y que en cada parte exista al menos un camino entre cada par de nodos. A esta variante se la conoce como el problema de equipartición en componentes conexas. Enfrentar este problema desde el punto de vista matemático es una tarea compleja, ya que se conoce que es un problema NP-duro, incluso en el caso en el que se elimine la restricción de la cantidad de nodos en cada subconjunto de la partición (Hojny *et al.*, 2020).

Por otro lado, la evidencia teórica diferencia si el grafo inicial es completo o no. En el primer caso, la condición de conexidad en cada uno de los subconjuntos de

nodos de la partición es trivial. Por otro lado, si el grafo considerado no es completo, entonces la conexidad de cada una de las componentes no puede ser asegurada. Por tal motivo, varios autores se refieren a la conexidad de la solución en varios problemas relacionados. Así, Wang *et al.* (2017) estudian desigualdades válidas que inducen facetas para el problema de encontrar el subgrafo conexo con peso máximo en un grafo. Jünger *et al.* (1985) proveen de condiciones necesarias para obtener conexidad en un subgrafo y de una herramienta para identificar si un grafo admite un particionamiento donde el peso en cada partición no exceda un valor fijo y se induzca un subgrafo conexo. El problema de equipartición en componentes conexas puede ser considerado como una generalización de varios problemas presentes en la literatura. De este modo, si se estudia el problema sobre grafos completos sin fijar el número de componentes conexas, entonces aparece el problema de particionamiento en cliques (Grötschel, M., Wakabayashi, 1989). De forma similar, si se relaja la condición del número de nodos en cada conjunto, el problema de k -particionamiento con restricciones de tamaño (Labbé y Özsoy, 2010) es revelado. En el caso en que se fije el número de particiones a dos, el problema de bisección puede ser identificado en Dellling *et al.* (2015).

Aplicaciones concretas al problema mencionado se pueden encontrar en Recalde *et al.* (2018) donde los autores realizan la construcción de grupos de equipos para el campeonato de fútbol profesional en la segunda categoría en el fútbol ecuatoriano. En este trabajo, los autores particionan el grafo completo que se forma al asociar las sedes de los equipos con nodos y los costos de las aristas con las distancias recorridas entre cada par de sedes, tal que la distancia total de viaje entre los equipos de todos los grupos sea minimizada. Por otro lado, Dilkina y Gomes (2010) estudian un problema ambiental relacionado al diseño de corredores de vida silvestre. Los autores plantean encontrar un subgrafo conexo de beneficio total máximo, tal que el subgrafo contenga un subconjunto específico de nodos y se respete ciertas restricciones presupuestarias.

En resumen, en este proyecto se propone estudiar el problema de equiparticionamiento en componentes conexas usando Programación Lineal Entera. Para la solución, varias familias de desigualdades válidas fueron encontradas y un método exacto de solución tipo Branch & Cut es expuesto. Este trabajo de titulación se encuentra organizada de la siguiente forma: En el Capítulo 2 se presentan las definiciones necesarias para abordar el problema. A continuación, en el Capítulo 3 se encuentra la formulación de los modelos de programación lineal entera mixta que describen el problema de la equipartición de grafos en k componentes conexas con

costos en las aristas. Por su parte, el Capítulo 4 detalla las desigualdades válidas para las formulaciones diseñadas. En el Capítulo 5 se exponen ampliamente los resultados computacionales obtenidos en este trabajo, mismo que finaliza con el Capítulo 6 donde se presentan las conclusiones y recomendaciones.

Capítulo 2

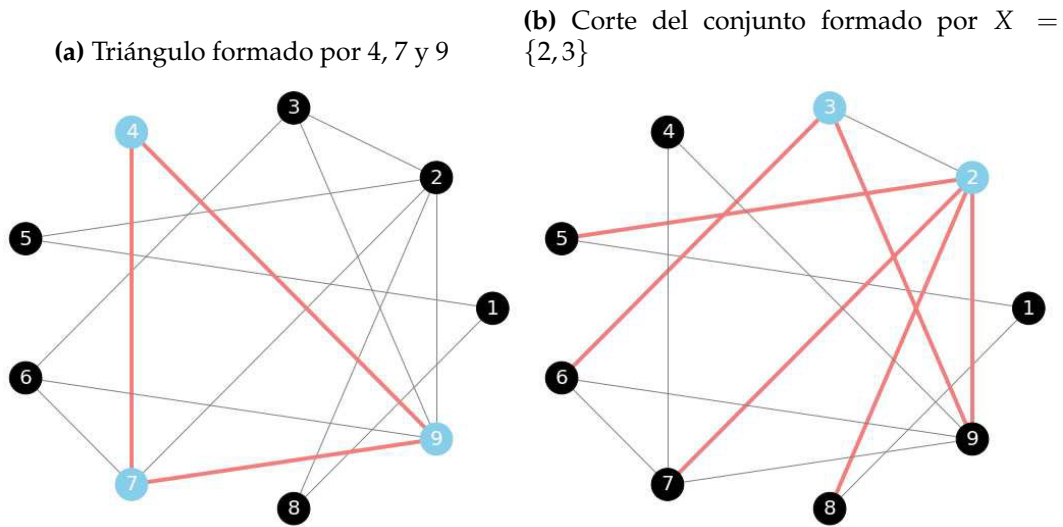
Definiciones Preliminares

2.1. Teoría de grafos

Un grafo es un objeto matemático que se compone de un conjunto de nodos y otro de aristas, donde una arista representa una asociación entre dos nodos. Dicho esto se define formalmente un grafo no dirigido simple como el par $G := (V, E)$, donde $V := \{v_1, \dots, v_n\}$ es un conjunto finito de puntos denominados nodos y $E := \{\{u, v\} : u, v \in V, u \neq v\}$ es el conjunto de pares de nodos llamados aristas. Se nota a $n = |V|$ a la cantidad de nodos y $m = |E|$ al número de aristas en el grafo. Adicionalmente, se define un grafo dirigido simple, el cual es un par compuesto por un conjunto finito de puntos $V = \{v_1, \dots, v_n\}$ llamados nodos y un subconjunto de pares ordenados de nodos $A = \{(u, v) : u, v \in V, u \neq v\}$ llamados arcos. Introducidos los conceptos de grafo dirigido y no dirigido, se menciona algunas características que serán importantes para los problemas en los que se enfoca este trabajo. Si entre cada par de nodos del grafo existe una arista, el grafo se lo denomina grafo completo K_n , mientras que si le hacen falta aristas se dice que es un grafo general con densidad $d := 2|E|/n(n-1)$. Por otro lado, si $W \subseteq V$ y $F \subseteq E$ se puede crear un grafo $H = (W, F)$ que se dice subgrafo de G . Con esto, si H es creado a partir de W , es decir, posee el conjunto de aristas $E(W)$ cuyos extremos están en W , entonces se dice que $H = (W, E(W))$ es un subgrafo inducido por W . De igual modo, si H es construido a partir de F , se expresa que $H = (V(F), F)$ es un subgrafo generado por F . Adicionalmente, se dice que H es una clique si es un subgrafo completo de G . Por ejemplo, podemos ver que un triángulo es una clique con tres nodos K_3 (Figura 2.1a).

Para un nodo $v \in V$, el conjunto de todas las aristas incidentes se representa por

Figura 2.1: Triángulos y Cortes



Elaboración propia

$\delta(v) := \{\{v, u\} : \{v, u\} \in E\}$ y la cardinalidad de $\delta(v)$ se conoce como grado del nodo. También se lo puede notar como δ_v . Extendiendo el concepto anterior a un subconjunto $X \subseteq V$, se define el corte $\delta(X) := \{\{u, v\} : u \in X, v \in V \setminus X\}$, es decir, el conjunto contiene todas las aristas que tienen uno de sus extremos en X y el otro fuera de dicho conjunto (Figura 2.1b).

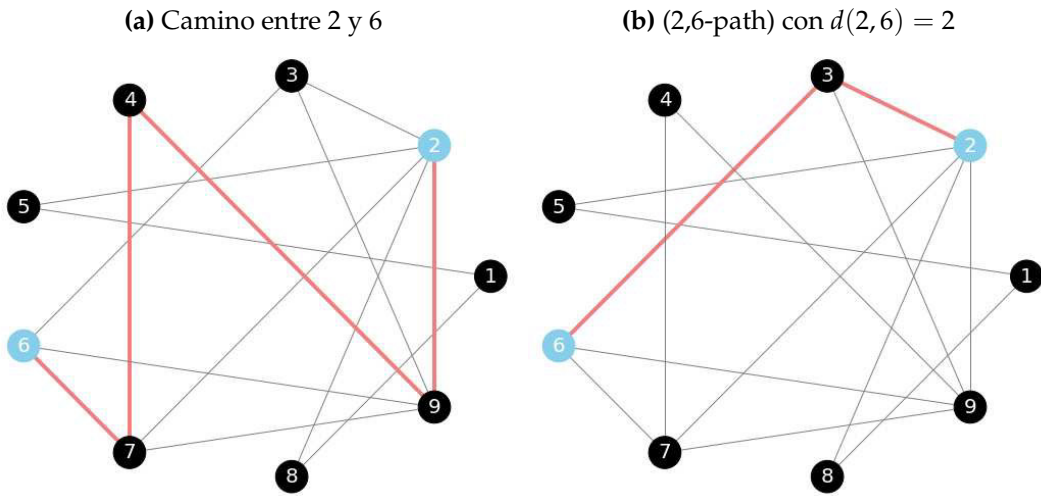
Un camino simple de v_i a v_k es un grafo no vacío $P = (V, E)$ de la forma:

$$V = \{v_i, \dots, v_k\} \quad \text{y} \quad E = \{\{v_i, v_{i+1}\}, \{v_{i+1}, v_{i+2}\}, \dots, \{v_{k-1}, v_k\}\},$$

donde los elementos de V son distintos y es representada por (*uv-path*) (Figura 2.2a). Los nodos v_i y v_k son unidos por P y son llamados nodos extremos y v_{i+1}, \dots, v_{k-1} los nodos internos de P . Notemos que a través del camino P el nodo v_k es alcanzable desde el nodo v_i . El número de aristas se conoce como la longitud del camino. Así, para dos nodos $u, v \in V$ se define $d(u, v)$ a la longitud más corta desde el nodo u hasta el nodo v (Figura 2.2b). Si no existe un camino para un par de nodos $v, w \in V$, entonces se fija $d(v, w) = \infty$. Si P es un camino de v_i a v_k es un camino de longitud mayor o igual a dos, entonces el grafo $C = P \cup \{v_i, v_k\}$ es un ciclo o circuito.

La noción de camino nos permite introducir la idea de conectividad o conexidad en grafos. Un grafo no vacío se llama conexo si cualquier par de sus nodos puede ser unido por un camino. Si el grafo no es conexo, entonces se llamará desconexo. Un conjunto de nodos $X \subseteq V$ es llamado conexo o componente conexa del grafo si el subgrafo inducido por X es conexo.

Figura 2.2: Caminos



Elaboración propia

Es fácil notar que existe una relación entre el concepto de componente conexa y el corte de un grafo. Así, si consideramos un grafo conexo $G = (V, E)$ y tomamos un subconjunto $X \subset V$, entonces podemos ver que si eliminamos el conjunto de aristas $\delta(X)$ se generan al menos dos componentes conexas. De este modo, diremos que un conjunto de aristas $F \subset E$ separa los conjuntos X y $V \setminus X$ si cualquier nodo $u \in X$ es alcanzable desde $v \in V \setminus X$ en el grafo G , pero no en el grafo $G' = (V, E \setminus F)$. Lo anterior nos permite definir la idea de conjuntos separadores.

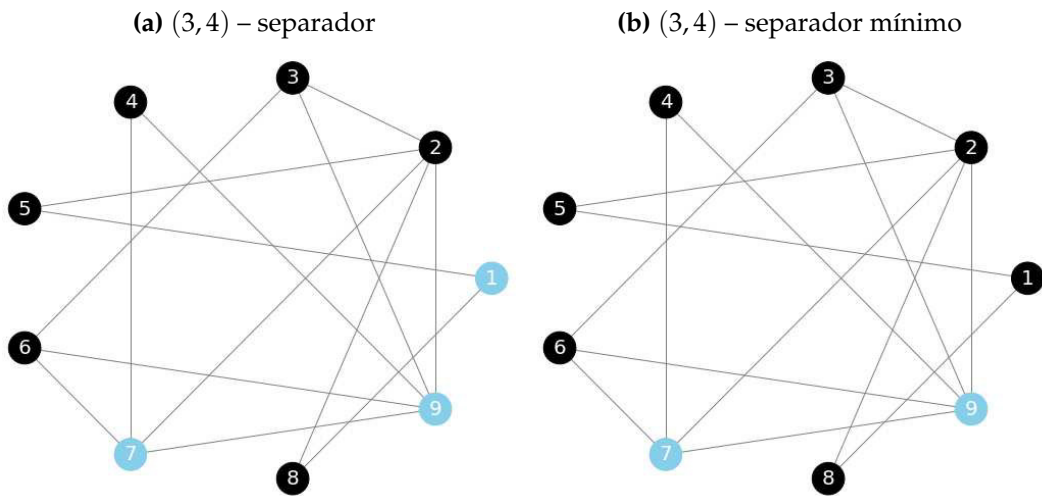
DEFINICIÓN 2.1. (Miyazawa et al., 2019) Sea $G = (V, E)$ un grafo no dirigido con u y v dos nodos no adyacentes. Diremos que el conjunto $S \subseteq V \setminus \{u, v\}$ es un (u, v) -separador si, u y v pertenecen a diferentes componentes conexas en el subgrafo inducido por $V \setminus S$. Así, definimos a $\Gamma_1(u, v)$ como el conjunto de todos los (u, v) -separadores mínimos en G .

La definición anterior se puede extender al caso en que se considere un conjunto de aristas.

DEFINICIÓN 2.2. Sea $G = (V, E)$ un grafo no dirigido con u y v dos nodos no adyacentes. Diremos que el conjunto $S \subseteq E$ es un (u, v) -separador si, u y v pertenecen a diferentes componentes conexas en el subgrafo generado por $E \setminus S$. Así, definimos a $\Gamma_2(u, v)$ como el conjunto de todos los (u, v) -separadores mínimos en G .

Usando conjuntos separadores se tiene la posibilidad de conocer los conjuntos de nodos y aristas indispensables para asegurar que entre un par de nodos siempre exista un camino. Un ejemplo de separador se encuentra en la Figura 2.3a, misma

Figura 2.3: Separadores



Elaboración propia

que muestra cuales nodos son necesarios para conectar los nodos 3 y 4, es decir, si se eliminan los nodos 1,7 y 9 del grafo, no existiría un camino entre dichos nodos. De forma similar, si solo se consideran los nodos 7 y 9, entonces entre los nodos 3 y 4 sigue sin existir un camino entre ellos. Esto produce que el conjunto $S = \{7, 9\}$ forme un separador mínimo para 3 y 4 ya que su conjunto no puede ser reducido (Figura 2.3b).

2.2. Programación Lineal Entera

La Programación Lineal consiste en optimizar (maximizar o minimizar) una función lineal sujeto a un conjunto de restricciones lineales para obtener el valor de las variables que den solución al problema. Así, en la presente sección introduciremos algunos conceptos básicos relacionados con la Programación Lineal y la Programación Lineal Entera.

DEFINICIÓN 2.3. *Dados los vectores $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ y una matriz $A \in \mathbb{R}^{m \times n}$. Un problema de programación lineal (LP) consiste en encontrar $x \in \mathbb{R}^n$ tal que $Ax \leq b$ y $c^T x$ sea maximizado, o decidir que el problema no tienen solución o es no acotado.*

A $c^T x$ la denominaremos función objetivo y $Ax \leq b$, $x \geq 0$ el conjunto de restricciones. También, podemos decir que la maximización de $c^T x$ está sujeta a $Ax \leq b$ y

$x \geq 0$ y tiene la siguiente notación:

$$\begin{aligned} & \text{máx } c^T x \\ & \text{sujeto a:} \\ & Ax \leq b, \\ & x \geq 0. \end{aligned}$$

Ahora definamos lo que es un poliedro. Un conjunto $P = \{x \in \mathbb{R}^n : Ax \leq b, x \geq 0\}$ es llamado un poliedro donde $A \in \mathbb{R}^{n \times m}$ y $b \in \mathbb{R}^m$. Equivalentemente, P es un poliedro si y solo si es la intersección de un número finito de subespacios afines. Si un poliedro es acotado es también conocido como un polítopo. Además, una desigualdad $a^T x \leq \gamma$ se llama válida para P si para cada $x \in P$, se verifica $a^T x \leq \gamma$. Dicho esto, podemos decir que un problema de programación lineal también puede escribirse como: $\text{máx}\{c^T x : x \in P\}$.

Para resolver un LP tenemos varios métodos. Por ejemplo, el Método del Simplex desarrollado por Dantzig (1951) que en la práctica funciona de forma eficiente, aunque se puedan construir instancias en las que se comporte en forma inadecuada. El primer algoritmo polinomial para este tipo de problemas fue reportado por Khachiyan (1979) adaptando el método de la elipsoide usado para resolver problemas no lineales, aunque el método es inaplicable en la práctica. Karmarkar (1984) presentó el Método de Punto Interior que resuelve problemas lineales en tiempo polinomial y además menciona que dispone de una implementación computacional que compite con el Simplex.

Por otro lado, si las variables solo pueden tomar valores enteros, la teoría clásica de programación lineal no es aplicable. Dicho esto, las siguientes definiciones particularizan a un programa lineal y permiten describir problemas más específicos que poseen mayor dificultad.

DEFINICIÓN 2.4. *Un programa lineal entero (IP) es un LP con la característica que las variables toman valores enteros, es decir, $\text{máx}\{c^T x : Ax \leq b, x \in \mathbb{Z}_n^+\}$.*

Ningún algoritmo polinomial es conocido para resolver problemas enteros en general. De hecho, la programación entera es un problema NP-completo. Entonces, si deseamos resolver el problema entero existen varios métodos que nos pueden ayudar en esta difícil tarea. Una idea natural consiste en cortar iterativamente ciertas partes del poliedro relajado sin perder las soluciones enteras. Este es el procedimiento del conocido algoritmo de Planos Cortantes propuesto por Dantzig *et al.* (1954),

quienes aplicaron esta idea al problema del agente viajero. Además, Gomory (1958) formuló un método para encontrar desigualdades válidas para todos los programas enteros, de modo que permita llegar a la solución óptima en un número finito pero exponencial de iteraciones.

Otro método es el algoritmo de Branch & Bound, el cual está incluido en el conjunto de técnicas de solución exactas. Dicho algoritmo realiza una enumeración de todas las posibles soluciones sin tener que considerarlas todas. Finalmente, podemos mencionar el método Branch & Cut que combina los métodos anteriores, es decir, ejecutar al algoritmo Branch & Bound y utilizar desigualdades válidas para ajustar las relajaciones lineales en cada uno de los nodos del árbol de búsqueda.

2.3. Particionamiento de Grafos

Es momento de definir formalmente el problema en el que se enfoca el presente trabajo, el problema de equiparticionamiento de grafos en componentes conexas. Así, dado un grafo con costos sobre las aristas, la tarea consiste en particionar el conjunto de nodos del grafo tal que dos características deban ser satisfechas en cada subconjunto: la primera impone la condición de que cada uno de los subconjuntos sea una componente conexa, mientras que en la segunda se debe garantizar que el número de nodos en cada subconjunto difiera a lo más en una unidad.

Iniciamos presentando formalmente el problema de equipartición:

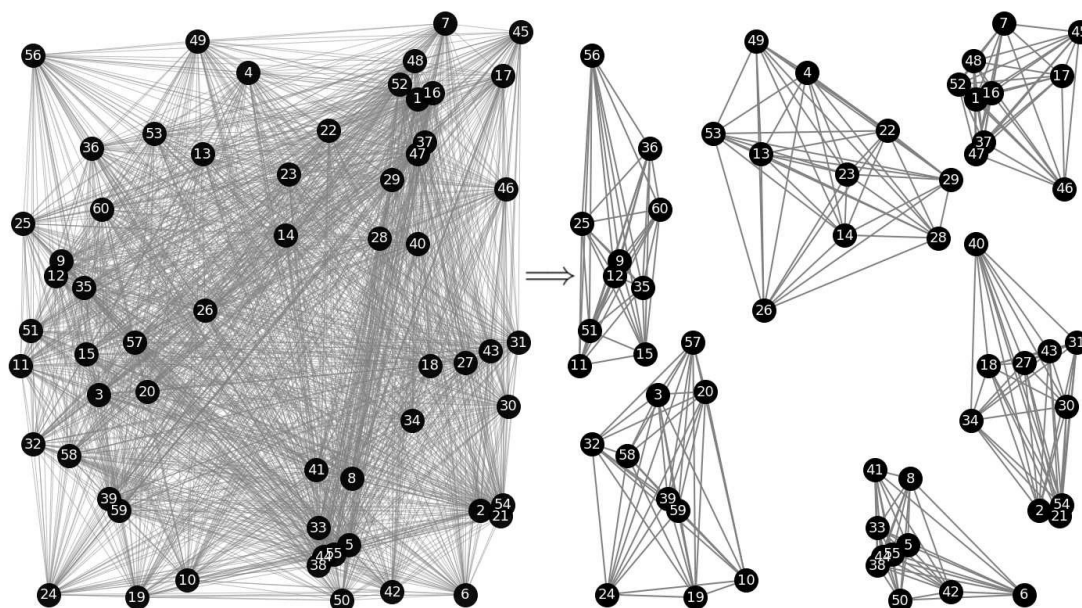
DEFINICIÓN 2.5. Sea $G = (V, E)$ un grafo completo con función de costos $c : E \rightarrow \mathbb{R}^+$ y un entero fijo $k \geq 2$. Así, equiparticionar un grafo en k componentes significa encontrar k subconjuntos no vacíos $V_1, \dots, V_k \subseteq V$ tal que:

- $\bigcup_{i=1}^k V_i = V$,
- $V_i \cap V_j = \emptyset \quad \forall i, j \in [k], i \neq j$,
- $\left| |V_i| - |V_j| \right| \leq 1 \quad \forall i, j \in [k], i \neq j$,

donde $[k] = \{1, 2, \dots, k\}$.

Es importante notar que todos los subgrafos $(V_i, E(V_i))$ son cliques y por tanto son componentes conexas ya que el grafo de entrada es un grafo completo. En la Figura 2.4 podemos visualizar un ejemplo del problema anterior, considerando una instancia con un grafo completo de 60 nodos particionado en $k = 6$ subconjuntos.

Figura 2.4: Equiparticionamiento en un Grafo Completo



Elaboración propia

A continuación se presenta una segunda definición que toma en cuenta grafos generales. En este caso, la condición de conexidad sobre cada una de los subconjuntos también debe ser considerada.

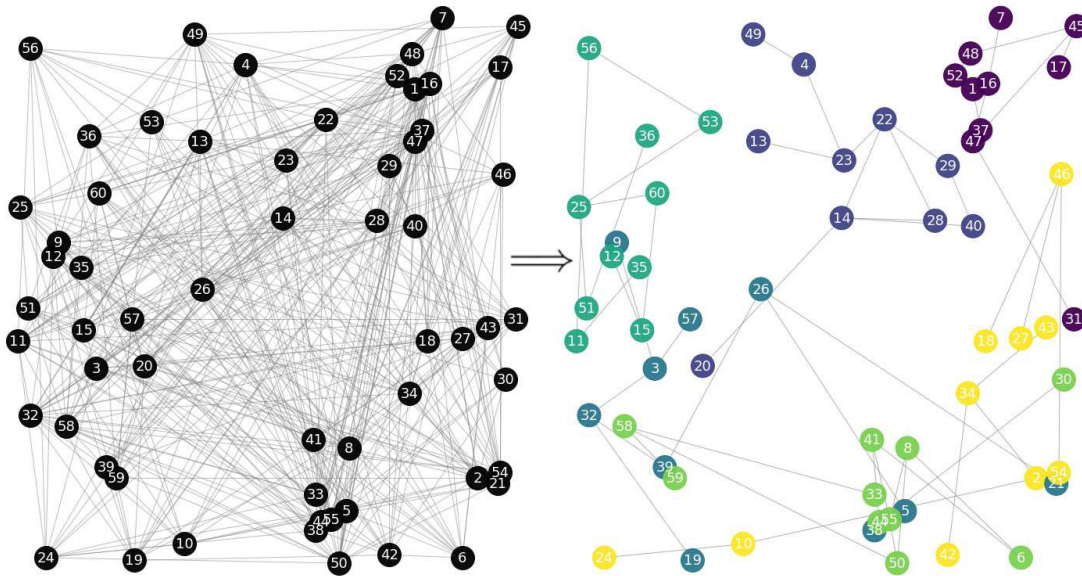
DEFINICIÓN 2.6. Sea $G = (V, E)$ un grafo conexo con función de costos $c : E \rightarrow \mathbb{R}^+$ y un entero fijo $k \geq 2$. Así, equiparticionar un grafo en k componentes conexas significa equiparticionar un grafo en k componentes tal que cada subgrafo inducido por los subconjuntos no vacíos $V_1, \dots, V_k \subseteq V$ sea conexo.

En la Figura 2.5 se muestra una instancia de un grafo general con densidad igual a 0,24 y con el mismo número de nodos e igual valor de k que la Figura 2.4. En ambas figuras podemos ver que existen notables diferencias respecto a los nodos que van juntos en una misma partición, ya que por las aristas faltantes no todo subconjunto de nodos puede ser una componente conexa.

Es fácil notar que, si el número de particiones divide exactamente al número de nodos del grafo, todas las particiones tendrán la misma cantidad de nodos y una partición tipo k -way es revelada. Por otro lado, si dicha división no es exacta, entonces existirá una o varias particiones que posean un nodo adicional. Es por ello que se demuestra el siguiente resultado que permite conocer cuántos nodos están contenidos en cada subconjunto de la partición.

TEOREMA 2.1. Dado un grafo $G = (V, E)$ con n nodos para ser particionado en

Figura 2.5: Equiparticionamiento en un Grafo General



Elaboración propia

k componentes. Si $r = n \bmod k$ es el resto de la división n/k , entonces existen r componentes con $\lceil n/k \rceil$ nodos y $k - r$ componentes con $\lfloor n/k \rfloor$ nodos.

Demostración. Empecemos tomando el caso en que k divide exactamente a n , puesto que $r = 0$, se tiene un resultado trivial ya que se obtendrán k componentes de n/k nodos en cada subconjunto. Por otro lado, supongamos que n/k no es entero, entonces existirán dos número enteros q y r tal que $n = kq + r$ y $r < k$. Ahora, si tomamos $q = \lfloor n/k \rfloor$ tenemos que:

$$\begin{aligned}
 n &= k \left\lfloor \frac{n}{k} \right\rfloor + r \\
 &= k \left\lfloor \frac{n}{k} \right\rfloor + r + r \left\lfloor \frac{n}{k} \right\rfloor - r \left\lfloor \frac{n}{k} \right\rfloor \\
 &= r \left(\left\lfloor \frac{n}{k} \right\rfloor + 1 \right) + (k - r) \left\lfloor \frac{n}{k} \right\rfloor \\
 &= r \left\lceil \frac{n}{k} \right\rceil + (k - r) \left\lfloor \frac{n}{k} \right\rfloor
 \end{aligned}$$

Esto implica que, el conjunto de n nodos puede ser particionado en r subconjuntos de tamaño $\lceil n/k \rceil$ y $k - r$ subconjuntos de tamaño $\lfloor n/k \rfloor$, demostrando el teorema. \square

Capítulo 3

Problema de equiparticionamiento de grafos

El problema de particionamiento de grafos ha sido modelizado usando diferentes técnicas como programación lineal entera, métodos heurísticos, programación cuadrática, entre otros. Por ejemplo, B.W. Kernighan y S. Lin (1970) presentaron un método heurístico que permite particionar grafos arbitrarios con costos sobre las aristas en subconjuntos de nodos de tamaño conocido. Extensiones del trabajo anterior pueden ser encontradas en métodos heurísticos recientes como METIS (Karypis y Kumar, 1998) o Kahip (Sanders y Schulz, 2012) que permiten particionar grafos de gran tamaño considerando costos sobre las aristas y múltiples pesos sobre los nodos. Usando programación cuadrática, Fan y Pardalos (2010) reportaron un modelo usando variables binarias e incluyeron varias reformulaciones y métodos de solución para aplicaciones prácticas.

En este trabajo nos enfocamos en métodos exactos basados en programación lineal entera. Así, Chopra y Rao (1993) presentaron un modelo de programación lineal entera para el problema de k -particionamiento junto con varias familias de desigualdades válidas y facetas para el poliedro asociado. Por otra parte, Recalde *et al.* (2018) presentaron un modelo de programación entera usando variables binarias que permite particionar un grafo completo con pesos sobre los nodos, tal que el costo total de las aristas con nodos finales en el mismo subconjunto sea minimizado.

Para el caso en el que se requiere componentes conexas trabajando sobre grafos generales, Hojny *et al.* (2020) buscan maximizar los costos de las aristas que pertenecen al corte. Los autores utilizan variables binarias para la elección de los arcos usados en la solución y variables continuas de flujo para garantizar la conexidad de

las componentes. Adicionalmente, Miyazawa *et al.* (2019) plantean dos formulaciones para una versión del k -particionamiento conexo balanceado, es decir, se busca particionar un grafo con pesos sobre los nodos en un número fijo de componentes conexas de pesos similares. Los autores proponen una formulación basada en conjuntos separadores que incluye un gran número de restricciones que potencialmente pueden ser implementadas en tiempo polinomial. El trabajo anterior es complementado con una segunda formulación basada en flujos.

En la presente sección, el problema de equiparticionamiento se aborda desde un punto de vista exacto. De esta forma, varias formulaciones basadas en programación lineal entera para resolver el problema son presentadas. Se pretende modelizar el problema con distintos enfoques: maximización de los costos de los arcos del corte y minimización de los costos de las aristas en cada partición. Además, consideramos analizar el problema sobre diferentes tipos de grafos, pues dependiendo de su estructura son necesarias distintas formulaciones para abordar el problema.

3.1. Grafos Completos

Es importante mencionar que al ser un grafo completo, cada uno de los subgrafos generados serán también completos, es decir, el presente caso particiona el grafo en cliques.

3.1.1. Primer modelo

La primera formulación considera en su función objetivo el enfoque de minimización del costo total de las aristas que conforman las cliques. Esta formulación es la reportada por Recalde *et al.* (2018) y puede ser escrita de la siguiente forma (\mathcal{FC} -1):

$$\text{mín } \sum_{\{u,v\} \in E} c_{uv} y_{uv} \quad (3.1)$$

sujeto a:

$$y_{uv} + y_{vw} - y_{uw} \leq 1, \quad \forall 1 \leq u < v < w \leq n, \quad (3.2)$$

$$y_{uv} - y_{vw} + y_{uw} \leq 1, \quad \forall 1 \leq u < v < w \leq n, \quad (3.3)$$

$$-y_{uv} + y_{vw} + y_{uw} \leq 1, \quad \forall 1 \leq u < v < w \leq n, \quad (3.4)$$

$$\left\lfloor \frac{n}{k} \right\rfloor \leq 1 + \sum_{\{u,v\} \in E} y_{uv} \leq \left\lceil \frac{n}{k} \right\rceil, \quad \forall u \in V, \quad (3.5)$$

$$y_{uv} \in \{0, 1\}, \quad \forall \{u, v\} \in E, \quad (3.6)$$

donde la variable y_{uv} toma el valor de uno cuando la arista $\{u, v\}$ pertenece a alguna clique, mientras que toma el valor de cero en caso contrario.

Como se mencionó anteriormente, la función objetivo (3.1) intenta minimizar el costo total de las aristas dentro del mismo subconjunto. Por otro lado, las restricciones (3.2), (3.3) y (3.4) denominadas desigualdades triangulares garantizan que todas las aristas que unen los nodos que se encuentran en el mismo subconjunto sean usadas. Así, si los nodos u, v y w se encuentran en el mismo subconjunto, entonces las variables asociadas a las aristas $\{u, v\}$, $\{u, w\}$ y $\{v, w\}$ toman el valor de uno. Finalmente, con la restricción (3.5) se controla el número de aristas adyacentes a cada nodo, haciendo que de cada nodo salgan al menos $\lfloor n/k \rfloor - 1$ y a lo más $\lceil n/k \rceil - 1$ aristas, es decir, la última restricción impone las condiciones para producir una equipartición.

3.1.2. Segundo modelo

Una segunda formulación para el problema de equiparticionamiento se deriva de la formulación reportada en Hojny *et al.* (2020). A diferencia del modelo \mathcal{FC} -1, el siguiente modelo intenta maximizar el costo total de las aristas en el multicorte. El modelo puede ser escrito de la siguiente forma (\mathcal{FC} -2):

$$\text{máx} \sum_{\{u,v\} \in E} c_{uv} y_{uv} \quad (3.7)$$

sujeto a:

$$\sum_{i \in [k]} x_{vi} = 1, \quad \forall v \in V, \quad (3.8)$$

$$\left\lfloor \frac{n}{k} \right\rfloor \leq \sum_{v \in V} x_{vi} \leq \left\lceil \frac{n}{k} \right\rceil, \quad \forall i \in [k], \quad (3.9)$$

$$x_{ui} + x_{vi} + y_{uv} \leq 2, \quad \forall \{u, v\} \in E, i \in [k], \quad (3.10)$$

$$x_{vi} \in \{0, 1\}, \quad \forall v \in V, i \in [k], \quad (3.11)$$

$$y_{uv} \in \{0, 1\}, \quad \forall \{u, v\} \in E. \quad (3.12)$$

En esta formulación tenemos dos conjuntos de variables. Las variables x_{vi} que toman el valor de uno si el nodo v pertenece a la partición i , mientras que la variable es igual a cero en otro caso. Además, se consideran variables y_{uv} asociadas a las aristas que se asignan a uno si la arista $\{u, v\}$ pertenece al corte, o se fija a cero en

otro caso.

La función objetivo (3.7) apunta a maximizar el costo total de las aristas en el corte. Esto implica intuitivamente que las aristas de menor costo no sean usadas y sean insertadas al interior de las cliques. El conjunto de restricciones están diseñadas para equiparticionar el grafo. Así, la restricción (3.8) garantiza que cada nodo sea asignado exactamente a un solo subconjunto. La restricción (3.9) define las condiciones para obtener un número balanceado de nodos en cada subconjunto, es decir, exige que el número de nodos en cada subconjunto se encuentre entre $\lfloor n/k \rfloor$ y $\lceil n/k \rceil$. Finalmente, la restricción (3.10) determina que si dos nodos se encuentran en distintas cliques, entonces la arista que los conecta debe encontrarse en el corte.

Podemos ver que los dos modelos anteriores utilizan criterios diferentes ya que el primero minimiza el uso de las aristas de acuerdo al costo, mientras que el segundo maximiza la cantidad de aristas fuera de las particiones respecto al costo. En otras palabras, intuitivamente el primer modelo desecha las aristas más costosas y el segundo las prefiere.

De las formulaciones anteriores podemos identificar sus poliedros. Así, definamos P^1 al conjunto de puntos que satisfacen las restricciones (3.2) - (3.6) del modelo \mathcal{FC} -1, es decir,

$$P^1 = \{y \in \{0, 1\}^{|E|} : (3.2) - (3.6) \text{ son satisfechas}\}.$$

En forma similar, denotamos por P^2 al conjunto de puntos que satisfacen las restricciones del modelo \mathcal{FC} -2, esto es,

$$P^2 = \{y \in \{0, 1\}^{|E|}, x \in \{0, 1\}^{|V| \times k} : (3.8) - (3.12) \text{ son satisfechas}\}.$$

El siguiente resultado establece una correspondencia entre las soluciones obtenidas en ambas formulaciones.

TEOREMA 3.1. *Un punto $y \in P^1$ si y sólo si el par $(1^{|E|} - y, x) \in P^2$.*

Demostración. Sea y un punto en P^1 el cual genera una solución de costo $C1$, entonces podemos construir una solución (t, x) en P^2 de costo $C2$. Además, encontramos las aristas del corte tomando $t_{uv} = 1 - y_{uv}$ para todo $\{u, v\} \in E$ y asignando el valor de uno a las variables asociadas a los nodos u y v siempre que la variable y_{uv} sea igual a uno ($x_{ui} = x_{vi} = 1$), para algún $i \in [k]$. Es decir,

$$y_{uv} = 1 \quad \implies \quad x_{ui} = 1 \quad \text{y} \quad x_{vi} = 1.$$

Por la restricción (3.8) cada nodo no puede ser asignado a más de una componente conexas respetando la condición de equipartición. Adicionalmente, se tiene la siguiente relación entre los costos:

$$\begin{aligned}
C1 &= \sum_{\{u,v\} \in E} c_{uv} y_{uv} \\
&= \sum_{\{u,v\} \in E} c_{uv} \cdot (1 - t_{uv}) \\
&= \sum_{\{u,v\} \in E} c_{uv} - \sum_{\{u,v\} \in E} c_{uv} t_{uv} \\
&= \sum_{\{u,v\} \in E} c_{uv} - C2.
\end{aligned}$$

Por otro lado, si se dispone de un punto factible (t, x) en P^2 con costo $C2$, es fácil notar que podemos elaborar un punto y en P^1 tomando $y_{uv} = 1 - t_{uv}$ y con costo $C1$, lo que concluye la demostración. \square

A continuación, se presenta el Cuadro 3.1 que resume los modelos presentados en esta sección.

Cuadro 3.1: Información de tamaño de los modelos \mathcal{FC} -1 y \mathcal{FC} -2.

Modelo	Cantidad de variables	Cantidad de restricciones	Criterio de optimización
\mathcal{FC} -1	$\frac{n(n-1)}{2}$	$n + \frac{n(n-1)(n-2)}{6}$	mín
\mathcal{FC} -2	$\frac{n(n-1)}{2} + nk$	$n + \frac{k}{2}(n(n-1) + 2)$	máx

Mediante el Cuadro 3.1 se evidencia que el enfoque de minimización produce un modelo con menor cantidad de variables, pero con mayor cantidad de restricciones respecto al enfoque de maximización.

3.2. Grafos Generales

En la presente sección consideramos la variante introducida en la Definición 2.6, es decir, los grafos a ser particionados tienen la característica de que su densidad es menor a uno. De este modo, al tener aristas faltantes, es necesario incluir restricciones adicionales para satisfacer las condiciones de conexidad. A continuación presentamos varias formulaciones para el problema de equiparticionamiento en componentes conexas sobre grafos generales.

3.2.1. Primer modelo

El primer modelo para esta variante se apoya en Hojny *et al.* (2020). La formulación inicia construyendo la versión dirigida del grafo de entrada de la instancia. El nuevo grafo dirigido posee el mismo conjunto de nodos y sus arcos son creados de la siguiente manera: si la arista $\{u, v\}$ se encuentra en el grafo original, entonces los arcos anti-paralelos (u, v) y (v, u) están en el grafo dirigido. Así, una arista del grafo original se encuentra asociada con dos arcos en el grafo dirigido.

La formulación usa variables similares a las utilizadas por el modelo \mathcal{FC} -2. Se definen variables x_{vi} que toman el valor de uno si el nodo v pertenece a la partición i , o cero si esto no sucede, y variables y_{uv} que se les asigna un valor igual a uno si la arista $\{u, v\}$ pertenece al corte, o cero en otro caso. Para satisfacer el requerimiento de conexidad, se propone enviar unidades de flujo entre los nodos de la misma componente conexa, es decir, se plantea identificar nodos sumideros artificiales. Dicho esto, z_{ui} toma el valor de uno si el nodo u tiene la característica de ser un nodo sumidero artificial, o cero si no lo es. Además, sobre la versión dirigida se definen variables f_{uv} que representan la cantidad de flujo sobre el arco (u, v) , mismo que puede ser un número real positivo incluido el cero.

El modelo puede ser escrito de la siguiente forma (\mathcal{FG} -1):

$$\text{máx} \sum_{\{u,v\} \in E} c_{uv} y_{uv} \quad (3.13)$$

sujeto a:

$$\sum_{i \in [k]} x_{vi} = 1, \quad \forall v \in V, \quad (3.14)$$

$$\left\lfloor \frac{n}{k} \right\rfloor \leq \sum_{v \in V} x_{vi} \leq \left\lceil \frac{n}{k} \right\rceil, \quad \forall i \in [k], \quad (3.15)$$

$$x_{ui} - x_{vi} \leq y_{uv}, \quad \forall \{u, v\} \in E, i \in [k], \quad (3.16)$$

$$x_{vi} - x_{ui} \leq y_{uv}, \quad \forall \{u, v\} \in E, i \in [k], \quad (3.17)$$

$$x_{ui} + x_{vi} + y_{uv} \leq 2, \quad \forall \{u, v\} \in E, i \in [k], \quad (3.18)$$

$$\sum_{u \in V} z_{ui} = 1, \quad \forall i \in [k], \quad (3.19)$$

$$z_{ui} \leq x_{ui}, \quad \forall u \in V, i \in [k], \quad (3.20)$$

$$f_{uv} + f_{vu} \leq M(1 - y_{uv}), \quad \forall \{u, v\} \in E, \quad (3.21)$$

$$\sum_{(u,v) \in \delta_u^-} f_{uv} - \sum_{(v,u) \in \delta_u^+} f_{vu} \geq 1 - M \sum_{i \in [k]} z_{ui}, \quad \forall u \in V, \quad (3.22)$$

$$x_{vi} \in \{0, 1\}, \quad \forall v \in V, i \in [k], \quad (3.23)$$

$$y_{uv} \in \{0, 1\}, \quad \forall \{u, v\} \in E, \quad (3.24)$$

$$z_{ui} \in \{0, 1\}, \quad \forall u \in V, i \in [k], \quad (3.25)$$

$$f_{uv}, f_{vu} \in \mathbb{R}_{\geq 0}, \quad \forall \{u, v\} \in E, \quad (3.26)$$

donde $M = n - k + 1$.

Así, la función objetivo (3.13) maximiza el costo total de las aristas en el corte. Lo anterior implica que las aristas de menor costo no son seleccionadas y se encuentran en los subgrafos inducidos por la partición. Dentro del conjunto de restricciones se pueden identificar un bloque de ellas que se concentran en equiparticionar el grafo. Así, (3.14) asegura que cada nodo sea asignado exactamente a un solo subconjunto, (3.15) limita la cantidad de nodos en cada subconjunto entre $\lfloor n/k \rfloor$ y $\lceil n/k \rceil$, (3.16) y (3.17) obligan a que si la arista $\{u, v\}$ no pertenece al corte, entonces los nodos u y v se encuentren en el mismo subconjunto, mientras que (3.18) afirma que si los nodos u y v se encuentran en la misma componente, entonces la arista $\{u, v\}$ no debe encontrarse en el corte. Por otro lado, el grupo de restricciones que asegura la conexidad de la partición empieza con la restricción (3.19) exigiendo que exactamente un solo nodo sea asignado como sumidero para cada subconjunto, mientras que (3.20) es una restricción de acoplamiento de variables e indica que únicamente los nodos que pertenecen a un subconjunto son candidatos para ser sumidero. Finalmente, las restricciones (3.21) y (3.22) permiten calcular el flujo en los arcos. En (3.21) se tiene que si la arista $\{u, v\}$ pertenece al corte, entonces no debe existir un flujo por sus arcos asociados en el grafo dirigido. Por otro lado, si la arista no es asignada al corte, entonces puede existir flujo sobre dichos arcos y la suma no debe superar $n - k + 1$ unidades. Finalmente, (3.22) es una restricción de conservación de flujo. Las restricciones antes mencionadas producen que los nodos en un mismo subconjunto de la partición se encuentren conectados y se conviertan en componentes conexas a través del flujo.

3.2.2. Segundo modelo

La siguiente formulación plantea otra forma de modelar la conexidad a través de nuevas desigualdades basadas en la estructura del grafo usando el concepto de conjuntos separadores. Así, las variables usadas en esta formulación son las mismas que se presentaron en la sección anterior, es decir, x_{vi} se fijan en uno si el nodo v pertenece a la partición i , o cero si no, y las variables y_{uv} toman en valor de uno si la arista $\{u, v\}$ pertenece al corte, cero en otro caso.

El segundo modelo para el problema de equiparticionamiento en componentes conexas (\mathcal{FG} -2) puede ser escrito:

$$\text{máx} \sum_{\{u,v\} \in E} c_{uv} y_{uv} \quad (3.27)$$

sujeto a:

$$\sum_{i \in [k]} x_{vi} = 1, \quad \forall v \in V, \quad (3.28)$$

$$\left\lfloor \frac{n}{k} \right\rfloor \leq \sum_{v \in V} x_{vi} \leq \left\lceil \frac{n}{k} \right\rceil, \quad \forall i \in [k], \quad (3.29)$$

$$x_{ui} - x_{vi} \leq y_{uv}, \quad \forall \{u, v\} \in E, i \in [k], \quad (3.30)$$

$$x_{vi} - x_{ui} \leq y_{uv}, \quad \forall \{u, v\} \in E, i \in [k], \quad (3.31)$$

$$x_{ui} + x_{vj} + y_{uv} \leq 2, \quad \forall \{u, v\} \in E, i, j \in [k], i \neq j, \quad (3.32)$$

$$x_{ui} + x_{vi} - \sum_{w \in S} x_{wi} \leq 1, \quad \forall \{u, v\} \notin E, S \in \Gamma_1(u, v), i \in [k], \quad (3.33)$$

$$x_{vi} \in \{0, 1\}, \quad \forall v \in V, i \in [k], \quad (3.34)$$

$$y_{uv} \in \{0, 1\}, \quad \forall \{u, v\} \in E. \quad (3.35)$$

De forma similar que en el modelo \mathcal{FG} -1, la función objetivo (3.27) utiliza las aristas más costosas en el corte. Las restricciones (3.28) - (3.32) son similares a las reportadas en \mathcal{FG} -1 y equiparticionan el grafo asignando cada nodo a un solo subconjunto en la partición, limitando la cantidad de nodos por subconjunto e identificando que si la arista $\{u, v\}$ se encuentra en el corte, entonces los nodos u, v deben ser asignados a subconjuntos distintos. Por otro lado, la restricción (3.33) se basa en el concepto de conjuntos separadores sobre un grafo (Definición 2.2). Dicha restricción muestra que si dos nodos no adyacentes $u, v \in V$ se encuentran en el mismo subconjunto, entonces necesariamente debe existir al menos un nodo de cada conjunto separador $S \in \Gamma_1(u, v)$ en el mismo subconjunto para asegurar que exista comunicación entre ellos.

Se propone una variante del modelo anterior al modificar la restricción de separadores y utilizar el conjunto $\Gamma_2(u, v)$ formado por las aristas. Con esto, la restricción (3.33) se expresa de la siguiente manera:

$$x_{ui} + x_{vi} + \sum_{\{r,s\} \in S} y_{rs} \leq 2 + (|S| - 1), \quad \forall \{u, v\} \notin E, S \in \Gamma_2(u, v), i \in [k], \quad (3.36)$$

con la idea de que al menos una arista de cada conjunto separador no sea incluida en el corte.

3.2.3. Tercer modelo

En esta última formulación, a diferencia de los modelos \mathcal{FG} -1 y \mathcal{FG} -2, se utiliza un enfoque de minimización intentando seleccionar las aristas menos costosas para construir las componentes conexas. La modificación del sentido de la optimización cambia la naturaleza de las variables con respecto a las dos formulaciones anteriores, pues ahora las variables y_{uv} toman el valor de uno si la arista $\{u, v\}$ pertenece a alguna de las componentes conexas, o cero en otro caso. Al igual que en los modelos anteriores, las variables binarias x_{vi} determinan si el nodo v se encuentra en la partición i . De igual forma, las variables z_{ui} identifican si el nodo u es un nodo fuente o no y las variables f_{uv} representan la cantidad de flujo enviado por el arco (u, v) .

El modelo puede ser expresado de la siguiente forma (\mathcal{FG} -3):

$$\text{mín } \sum_{\{u,v\} \in E} c_{uv} y_{uv} \quad (3.37)$$

sujeto a:

$$\sum_{i \in [k]} x_{vi} = 1, \quad \forall v \in V, \quad (3.38)$$

$$x_{ui} + x_{vi} - y_{uv} \leq 1, \quad \forall \{u, v\} \in E, i \in [k], \quad (3.39)$$

$$x_{ui} + x_{vj} + y_{uv} \leq 2, \quad \forall \{u, v\} \in E, i, j \in [k], i \neq j, \quad (3.40)$$

$$\sum_{u \in V} z_{ui} = 1, \quad \forall i \in [k], \quad (3.41)$$

$$z_{ui} \leq x_{ui}, \quad \forall u \in V, i \in [k], \quad (3.42)$$

$$f_{uv} + f_{vu} \leq M y_{uv}, \quad \forall \{u, v\} \in E, \quad (3.43)$$

$$\sum_{(u,v) \in \delta_u^-} f_{uv} - \sum_{(v,u) \in \delta_u^+} f_{vu} = \left\lfloor \frac{n}{k} \right\rfloor \sum_{i=1}^r z_{ui} + \left\lceil \frac{n}{k} \right\rceil \sum_{i=r+1}^k z_{ui} - 1, \quad \forall u \in V, \quad (3.44)$$

$$x_{ui} \in \{0, 1\}, \quad \forall u \in V, i \in [k], \quad (3.45)$$

$$y_{uv} \in \{0, 1\}, \quad \forall \{u, v\} \in E, \quad (3.46)$$

$$z_{ui} \in \{0, 1\}, \quad \forall u \in V, i \in [k], \quad (3.47)$$

$$f_{uv}, f_{vu} \in \mathbb{R}_{\geq 0}, \quad \forall \{u, v\} \in E, \quad (3.48)$$

donde $M = n - k + 1$ y $r = n \bmod k$.

En la función objetivo (3.37) se apunta a minimizar el costo total de las aristas con nodos finales en la misma componente conexa. La restricción (3.38) garantiza que cada nodo sea asignado exactamente a una sola componente conexa. La desigualdad (3.39) muestra que si dos nodos incidentes u y v son asignados a un mis-

mo subconjunto, entonces la variable asociada a esa arista (si existe) debe tomar el valor de uno, indicando que se encuentra en la solución. Adicionalmente, en (3.40) se asegura que si dos nodos incidentes se encuentran en diferentes subconjuntos de la partición, entonces la variable asociada a dicha arista debe tomar el valor de cero. Con estas dos últimas restricciones describimos completamente la relación existente entre nodos y aristas dentro y fuera de una componente conexa. La restricción (3.41) asigna un único nodo fuente en cada subconjunto de la partición, mientras que la restricción (3.42) permite a un nodo ser fuente si y solo si primero pertenece a dicho subconjunto. Finalmente, (3.43) es una restricción de conservación de flujo y muestra que se seleccionan r nodos fuente que envían $\lceil n/k \rceil - 1$ unidades de flujo y $k - r$ nodos sumidero que envían $\lfloor n/k \rfloor - 1$ unidades de flujo. Además, si un nodo no es fuente, entonces será un sumidero con demanda de una unidad de flujo. El lado derecho de la restricción claramente es justificado por el Teorema 2.1. Notemos que si n divide exactamente a k , entonces r es igual a cero y la restricción (3.44) se escribe de la siguiente forma:

$$\sum_{(u,v) \in \delta_u^-} f_{uv} - \sum_{(v,u) \in \delta_u^+} f_{vu} = \frac{n}{k} \sum_{i=1}^k z_{u,i} - 1, \quad \forall u \in V. \quad (3.49)$$

El Teorema 3.1 puede ser extendido para obtener la equivalencia entre los modelos presentados para grafos generales. Para ello, es necesario presentar los poliedros asociados a cada una de las formulaciones. Definimos:

$$\begin{aligned} P^3 &= \{y \in \{0,1\}^{|E|}, x, z \in \{0,1\}^{|V| \times k}, f \in \mathbb{R}_{\geq 0} : (3.14) - (3.26) \text{ son satisfechas}\}, \\ P^4 &= \{y \in \{0,1\}^{|E|}, x \in \{0,1\}^{|V| \times k} : (3.28) - (3.35) \text{ son satisfechas}\}, \\ P^5 &= \{y \in \{0,1\}^{|E|}, x, z \in \{0,1\}^{|V| \times k}, f \in \mathbb{R}_{\geq 0} : (3.38) - (3.48) \text{ son satisfechas}\}, \end{aligned}$$

donde el poliedro P^{i+2} está asociado a la formulación $\mathcal{FG}-i$, para $i = 1, 2, 3$. Los siguientes resultados muestran la relación entre ellos:

TEOREMA 3.2. *Un punto $(x, y, z, f) \in P^3$ si y sólo si $(x, y) \in P^4$.*

Demostración. Sea (x, y, z, f) un punto en P^3 que genera una solución de costo $C3$, entonces podemos elaborar una solución (x', y') para P^4 con costo $C4$. Dado que las variables sobre nodos y aristas poseen la misma interpretación se tiene una relación directa, es decir, $x' = x$ y $y' = y$. Notemos que utilizando dicha relación las variables x' satisfacen la restricción (3.33), ya que x garantiza conectividad en P^3 . Esto se tiene ya que si $x'_{ui} = x'_{vi} = 1$, para nodos u, v no adyacentes y algún

$i \in [k]$, entonces cualquier conjunto separador S debe contener al menos un nodo en $V_i \setminus \{u, v\}$, asegurando de este modo que $\sum_{w \in S} x'_{wi} \geq 1$ y por tanto el cumplimiento de la restricción mencionada. Además, se tiene que $C4 = C3$.

La otra implicación sigue inmediatamente tomando $x = x', y = y'$ y seleccionando aleatoriamente un nodo sumidero en cada V_i , para $i \in [k]$. Finalmente, los valores de las variables de flujo pueden ser polinomialmente obtenidas asegurando la conservación de flujo sobre cada una de las componentes conexas, lo que concluye la demostración. \square

TEOREMA 3.3. *Un punto $(x, y, z, f) \in P^3$ si y sólo si $(x, 1^{|E|} - y, z, f) \in P^5$.*

Demostración. Sea (x, y, z, f) un punto en P^3 que genera una solución de costo $C3$, entonces podemos construir una solución (x', y', z', f') para P^5 con costo $C5$. Así, se pueden encontrar las aristas en la solución de P^5 tomando $y'_{uv} = 1 - y_{uv}$ para todo $\{u, v\} \in E$, y para los nodos utilizando $x'_{ui} = x_{ui}$, para todo $u \in V$ y $i \in [k]$. Además, los nodos fuente pueden ser identificados a partir de los nodos sumideros eligiendo $z'_{ui} = z_{ui}$, para todo $u \in V$ y $i \in [k]$. Finalmente, las variables f'_{uv} son reveladas mediante $f'_{uv} = f_{vu}$ para todo $\{u, v\} \in E$. De esta forma encontramos una equivalencia entre ambos poliedros utilizando la idea de que P^3 posee nodos sumidero y P^5 nodos fuente, es por ello que sus variables de flujo cambian de sentido dependiendo si el nodo es sumidero o fuente. Adicionalmente, en el valor de la función objetivo se tiene la misma relación que el caso completo, pues los costos solo dependen de las variables asociadas a las aristas. Así:

$$C3 = \sum_{\{u,v\} \in E} c_{uv} y_{uv} \quad (3.50)$$

$$= \sum_{\{u,v\} \in E} c_{uv} \cdot (1 - y'_{uv}) \quad (3.51)$$

$$= \sum_{\{u,v\} \in E} c_{uv} - \sum_{\{u,v\} \in E} c_{uv} y'_{uv} \quad (3.52)$$

$$= \sum_{\{u,v\} \in E} c_{uv} - C5 \quad (3.53)$$

Por otro lado, si se dispone de un punto factible (x', y', z', f') para P^5 con costo $C5$, se puede notar que mediante las igualdades anteriores es posible construir un punto (x, y, z, f) en P^3 con costo $C3$, lo que concluye la demostración. \square

Por el Teorema 3.2 y 3.3 se tiene el siguiente resultado:

COROLARIO 3.4. *Un punto $(x, y) \in P^4$ si y sólo si $(x, 1^{|E|} - y, z, f) \in P^5$.*

A continuación, se presenta el Cuadro 3.2 que resume los modelos presentados en esta sección.

Cuadro 3.2: Información de tamaño de los modelos \mathcal{FG} -1, \mathcal{FG} -2 y \mathcal{FG} -3.

Modelo	Cantidad de variables	Cantidad de restricciones	Criterio de optimización	Criterio de conectividad
\mathcal{FG} -1	$3m + 2nk$	$m + 2n + 2k + k(3m + n)$	máx	flujo
\mathcal{FG} -2	$m + nk$	$n + k + \frac{3}{2}mk + \frac{1}{2}mk^2 + \alpha\beta k$	máx	separadores
\mathcal{FG} -3	$3m + 2nk$	$m + 2n + k + \frac{1}{2}mk + \frac{1}{2}mk^2 + nk$	mín	flujo

con $\alpha = \frac{n(n-1)}{2} - m$ y $\beta = \sum_{\{u,v\} \notin E} |\Gamma_1(u,v)|$.

El Cuadro 3.2 muestra que el modelo \mathcal{FG} -2, el cual usa un enfoque de maximización y separadores para resolver la conectividad, posee la menor cantidad de variables y la mayor cantidad de restricciones ya que depende de la cardinalidad de los conjuntos separadores. Así, el modelo \mathcal{FG} -1 que utiliza el flujo para solventar la conectividad es el que posee la menor cantidad de restricciones.

Capítulo 4

Desigualdades válidas

En el presente capítulo, varias familias de desigualdades válidas asociadas a los poliedros del problema de equiparticionamiento de grafos en componentes conexas son presentadas, mismas que serán incluidas en los modelos $\mathcal{FG}\text{-}1$, $\mathcal{FG}\text{-}2$ y $\mathcal{FG}\text{-}3$ en un esquema de Branch & Cut. Nótese que al ser las formulaciones equivalentes, las desigualdades presentadas pueden ser transformadas desde el caso de minimización al caso de maximización y viceversa. Además, los poliedros definidos por la envolvente convexa de las soluciones enteras están únicamente determinados por la estructura del grafo general $G = (V, E)$ y los parámetros Γ_1, Γ_2 y $k \geq 2$.

Los primeros resultados se derivan de las desigualdades válidas introducidas en Hojny *et al.* (2020) y Miyazawa *et al.* (2019). Iniciamos introduciendo el concepto de un nodo de articulación sobre un grafo. Un nodo de articulación es un nodo $u \in V$ tal que si u es eliminado del grafo G , entonces G es disconexo. Usando este concepto podemos presentar el primer tipo de desigualdades válidas.

TEOREMA 4.1. *Sea $\ell \in V$ un nodo de articulación y sean $u, v \in V$ nodos en diferentes componentes conexas en el subgrafo inducido por $V \setminus \{\ell\}$. Entonces las desigualdades:*

$$x_{ui} + x_{vi} - x_{\ell i} \leq 1, \quad \forall i \in [k], \quad (4.1)$$

son válidas para $\mathcal{FG}\text{-}1$ y $\mathcal{FG}\text{-}3$.

El resultado anterior puede ser generalizado usando la noción de conjuntos separadores al extender la idea de un nodo de articulación. Miyazawa *et al.* (2019) demuestran que (3.33) es una desigualdad válida para el caso de maximización, mostrando que si dos nodos no adyacentes se encuentran en una misma componente, entonces debe existir al menos un nodo de cada conjunto separador asignado al

mismo subconjunto de la partición. Esto asegura que exista comunicación entre dos nodos no adyacentes en un mismo subconjunto. Con esto se presenta el siguiente resultado:

TEOREMA 4.2. *Sean u y v dos nodos no adyacentes en V y $\Gamma_1(u, v)$ y $\Gamma_2(u, v)$ sus conjuntos separadores de nodos y aristas, respectivamente. Entonces las desigualdades:*

$$x_{ui} + x_{vi} - \sum_{w \in S_1} x_{wi} \leq 1, \quad \forall S_1 \in \Gamma_1(u, v), \quad (4.2)$$

$$x_{ui} + x_{vi} + \sum_{\{r,s\} \in S_2} y_{rs} \leq 2 + (|S_2| - 1), \quad \forall S_2 \in \Gamma_2(u, v), \quad (4.3)$$

son válidas para \mathcal{FG} -1 y las desigualdades:

$$x_{ui} + x_{vi} - \sum_{w \in S_1} x_{wi} \leq 1, \quad \forall S_1 \in \Gamma_1(u, v), \quad (4.4)$$

$$x_{ui} + x_{vi} - \sum_{\{r,s\} \in S_2} y_{rs} \leq 1, \quad \forall S_2 \in \Gamma_2(u, v), \quad (4.5)$$

son válidas para \mathcal{FG} -3.

Notemos que su validez es consecuencia de la restricción (3.33) que asegura conexidad de las componentes.

El tercer tipo de desigualdades determinan el número mínimo de aristas necesarias para asegurar la conexidad de una solución factible, lo que permite mejorar la cota inferior que proviene de la relajación lineal en los distintos modelos. La desigualdad tiene la siguiente estructura:

TEOREMA 4.3. *La desigualdad:*

$$\sum_{\{u,v\} \in E} y_{uv} \leq m - n + k, \quad (4.6)$$

es válida para \mathcal{FG} -1 y \mathcal{FG} -2, y la desigualdad:

$$\sum_{\{u,v\} \in E} y_{uv} \geq n - k, \quad (4.7)$$

es válida para \mathcal{FG} -3.

Demostración. Sea n_i el número de nodos de la componente i , para $i \in [k]$, tal que $\sum_{i \in [k]} n_i = n$ y $\lfloor n/k \rfloor \leq n_i \leq \lceil n/k \rceil$. Es conocido que se necesitan al menos $n_i - 1$

aristas para que la componente i sea conexa. Para la formulación \mathcal{FG} -3 se tiene:

$$\sum_{\{u,v\} \in E} y_{uv} \geq \sum_{i=1}^k (n_i - 1) = \sum_{i=1}^k n_i - \sum_{i=1}^k 1 = n - k,$$

es decir, se necesitan al menos $n - k$ aristas para obtener un equiparticionamiento conexo. Por otro lado, las desigualdades para los casos de maximización (\mathcal{FG} -1 y \mathcal{FG} -2) se obtienen mediante las equivalencias del Teorema 3.3 y del Corolario 3.4 usando la relación $y' = 1^{|E|} - y$. Así,

$$\begin{aligned} \sum_{\{u,v\} \in E} y_{uv} &\geq n - k, \\ \sum_{\{u,v\} \in E} (1 - y'_{uv}) &\geq n - k, \\ m - \sum_{\{u,v\} \in E} y'_{uv} &\geq n - k, \\ \sum_{\{u,v\} \in E} y'_{uv} &\leq m - n + k. \end{aligned}$$

□

El cuarto tipo de desigualdades se basan en el uso de algoritmos de camino más corto entre dos nodos del grafo. La desigualdad identifica pares de nodos que no pueden ser incluidos en la misma componente conexa, es decir, si existe una componente que incluya dicho par de nodos, entonces se violaría el requerimiento de equipartición. Puede ser expresada por el siguiente resultado:

TEOREMA 4.4. *Sean u y v dos nodos de G tal que $d(u, v) \geq \lceil n/k \rceil$, entonces las desigualdades:*

$$x_{ui} + x_{vi} \leq 1, \quad \forall i \in [k], \quad (4.8)$$

son válidas para \mathcal{FG} -1, \mathcal{FG} -2 y \mathcal{FG} -3.

Demostración. Tomemos dos nodos u y v y se conoce que el tamaño máximo de una componente es $\lceil n/k \rceil$. Como se tiene que $d(u, v) \geq \lceil n/k \rceil$, entonces cualquier camino que permite conectar a los nodos u y v en el grafo utiliza al menos $\lceil n/k \rceil$ aristas. Esto implica que si alguna componente conexa contiene a los nodos u y v , entonces dicha componente debe tener al menos $\lceil n/k \rceil + 1$ nodos violando de este modo la condición de equipartición.

Obsérvese que este tipo de desigualdades, al depender solo de las variables sobre los nodos, es idéntica para los casos de maximización y minimización ya que la

interpretación de dichas variables es la misma en ambos casos. □

El quinto tipo de desigualdades usa la idea de que si tres nodos se encuentran dentro de un mismo subconjunto, entonces las aristas que los conectan también deben ser usadas en la solución. Grötschel, M., Wakabayashi (1989) mencionan que dados tres nodos u, v y w que forman un triángulo, las desigualdades válidas asociadas a dichos nodos tienen la siguiente forma:

$$-y_{vw} + y_{wu} + y_{uv} \leq 1, \quad (4.9)$$

$$y_{vw} - y_{wu} + y_{uv} \leq 1, \quad (4.10)$$

$$+y_{vw} + y_{wu} - y_{uv} \leq 1, \quad (4.11)$$

Por la estructura de (4.9)-(4.11) son desigualdades válidas para \mathcal{FG} -3. Utilizando la equivalencia entre las formulaciones se tiene:

$$y_{vw} - y_{wu} - y_{uv} \leq 0, \quad (4.12)$$

$$-y_{vw} + y_{wu} - y_{uv} \leq 0, \quad (4.13)$$

$$-y_{vw} - y_{wu} + y_{uv} \leq 0, \quad (4.14)$$

y podemos concluir que son desigualdades válidas para \mathcal{FG} -1 y \mathcal{FG} -2.

El sexto tipo de desigualdades son útiles cuando el grafo es denso. La idea consiste en identificar un subconjunto de nodos en el que se pueda asegurar que varios nodos de dicho subconjunto deben coincidir en la misma componente conexa. Esto se logra encontrando subgrafos completos con $k + 1$ nodos y por el hecho de que el grafo debe ser particionado en k componentes, entonces al menos 2 nodos de la clique deben ser incluidos en el mismo subconjunto. El resultado es presentado en el siguiente teorema:

TEOREMA 4.5. *Sea $T = (T', E')$ una clique en G con $|T'| = k + 1$ nodos, entonces la desigualdad:*

$$\sum_{\{u,v\} \in E'} y_{uv} \leq \frac{(k+1)k}{2} - 1, \quad (4.15)$$

es válida para \mathcal{FG} -1 y \mathcal{FG} -2. Además, la desigualdad:

$$\sum_{\{u,v\} \in E'} y_{uv} \geq 1, \quad (4.16)$$

es válida para \mathcal{FG} -3.

Demostración. Sea $T = (T', E')$ una clique con $k + 1$ nodos contenida en G . Es fácil

notar que jamás podría presentarse el caso de que cada elemento de la clique pertenezca a una componente diferente ya que se tiene $k + 1$ nodos en T' y tan solo k componentes. Esto implica que al menos dos nodos en T' deben ir juntos en un mismo subconjunto de la partición, es decir, al menos una arista perteneciente a E' debe ser elegida dentro de una componente conexa. Para la formulación \mathcal{FG} -3 se tiene:

$$\sum_{\{u,v\} \in E'} y_{uv} \geq 1.$$

Las desigualdades para los casos de maximización (\mathcal{FG} -1 y \mathcal{FG} -2) se obtienen mediante las equivalencias del Teorema 3.3 y del Corolario 3.4 usando la relación $y' = 1^{|E|} - y$. Así,

$$\begin{aligned} \sum_{\{u,v\} \in E'} y_{uv} &\geq 1, \\ \sum_{\{u,v\} \in E'} (1 - y'_{uv}) &\geq 1, \\ |E'| - \sum_{\{u,v\} \in E'} y'_{uv} &\geq 1, \\ \frac{(k+1)k}{2} - \sum_{\{u,v\} \in E'} y'_{uv} &\geq 1, \\ \sum_{\{u,v\} \in E'} y'_{uv} &\leq \frac{(k+1)k}{2} - 1. \end{aligned}$$

□

Hojny *et al.* (2020) reportan que las desigualdades del teorema anterior pueden ser separadas por un algoritmo de fuerza bruta en un tiempo $\mathcal{O}(|V|^{k+1})$. Además, aconseja que solo se utilice para instancias con $k \leq 3$ puesto que la cantidad de cliques crece exponencialmente dependiendo de k .

Finalmente, el séptimo tipo de desigualdades válidas se centran en identificar subconjuntos de nodos que no satisfacen las condiciones de equiparticionamiento, es decir, realizar una búsqueda de aquellos subconjuntos que disponen de máximo $\lfloor n/k \rfloor - 1$ nodos y asegurar que exista una arista en el corte para garantizar la conexidad. El resultado es formalizado de la siguiente forma:

TEOREMA 4.6. *Sea $Q \subset V$ tal que $1 \leq |Q| \leq \lfloor n/k \rfloor - 1$, entonces la desigualdad:*

$$\sum_{\{u,v\} \in \delta(Q)} y_{uv} \leq |\delta(Q)| - 1, \quad (4.17)$$

es válida para \mathcal{FG} -1 y \mathcal{FG} -2. Además, la desigualdad:

$$\sum_{\{u,v\} \in \delta(Q)} y_{uv} \geq 1, \quad (4.18)$$

es válida para \mathcal{FG} -3.

Demostración. Sea $Q \subset V$ tal que $1 \leq |Q| \leq \lfloor n/k \rfloor - 1$. Dado que cada componente debe tener al menos $\lfloor n/k \rfloor$ nodos, entonces al menos un nodo en la vecindad debe ser incluido en el conjunto Q para alcanzar el número mínimo de nodos en la componente conexa y satisfacer la condición de equipartición. Esto implica que para la formulación \mathcal{FG} -3 debe existir al menos una arista en el corte de Q . Por otro lado, para las formulaciones \mathcal{FG} -1 y \mathcal{FG} -2 se puede usar el mismo argumento, es decir, para asegurar que el conjunto Q se transforme en una componente conexa, a lo más $|\delta(Q)| - 1$ aristas deben ser incluidas en el corte. Si Q contiene nodos de varios subconjuntos de la partición, entonces más de una arista en $\delta(Q)$ será incluida en la solución.

□

Capítulo 5

Resultados Computacionales

En el presente capítulo se reporta los resultados computacionales obtenidos al implementar los modelos desarrollados en el presente trabajo para el problema de equiparticionamiento de grafos en componentes conexas. Además, para grafos generales se incluyen resultados alcanzados al incluir las desigualdades válidas en un algoritmo tipo Branch & Cut.

Las instancias utilizadas para el desarrollo de este trabajo fueron simuladas generando n puntos (x, y) en el plano, donde $x \in [0; 200]$ e $y \in [0; 200]$. La posición de cada nodo es asociada a cada punto (x, y) . Para cada par de puntos $u, v \in V$ con $u \neq v$, se genera una arista con costo c_{uv} igual a la distancia euclidiana entre los puntos extremos de dicha arista. Para el caso de grafos generales, un número real en el intervalo $[0, 1]$ es considerado, de forma que la arista exista basada en una probabilidad igual a dicho valor.

Para comparar de mejor forma las soluciones de los modelos con distintos enfoques, se propone transformar el valor de la función objetivo de los modelos de maximización a su valor equivalente en los modelos de minimización. Se utiliza el Teorema 5.1 para las instancias que usan grafos completos y los Teoremas 3.2 y 3.3 para cuando los grafos de entrada son generales. Así por ejemplo, si $\mathcal{FC}-2$ encuentra una equipartición con costo $C2$, entonces en los diferentes reportes se incluirá el valor:

$$C1 = \sum_{\{u,v\} \in E} c_{uv} - C2.$$

Todos los experimentos computacionales fueron realizados utilizando un computador con procesador Intel Core i7-9700K CPU 3.60Ghz con 32 GB de memoria RAM en Windows 10 Education. Además, se utilizó la versión 9.1.1 del solver de optimi-

zación Gurobi a través de la distribución Anaconda. Adicionalmente, se programó en la aplicación Jupyter Notebook mediante el lenguaje de programación Python.

5.1. Equipartición de grafos

En esta sección se presenta la comparación entre el modelo $\mathcal{FC}-1$ con enfoque de minimización y el modelo $\mathcal{FC}-2$ con enfoque de maximización (Cuadro 5.1). Para este caso no se propusieron desigualdades válidas, por lo que todas las instancias fueron resueltas usando Gurobi en su configuración por defecto.

En la comparación se observan 13 instancias que varían dependiendo del número de nodos n y la cantidad de subconjuntos k . En este caso, el tiempo máximo de ejecución es de 1800 segundos. Para cada modelo se muestra el valor de la función objetivo, el número de nodos explorados, el tiempo de ejecución en segundos y la brecha de optimalidad en porcentaje (GAP).

Cuadro 5.1: Comparación de los modelos $\mathcal{FC}-1$ y $\mathcal{FC}-2$.

Instancia		$\mathcal{FC}-1$				$\mathcal{FC}-2$			
n	k	F. Obj.	Nodos	Tiempo	GAP	F. Obj.	Nodos	Tiempo	GAP
31	10	1129.74	1	0.42	0	1129.74	12697	21.87	0
40	10	1738.27	1	1.16	0	1738.27	359081	1291.29	0
	4	17072.86	15	149.22	0	17072.86	14389	1800.05	1.54
54	6	9001.55	1	44.55	0	9001.55	50808	1800.07	0.32
	8	9714.44	1	56.09	0	9714.44	26065	1800.07	0.63
	9	5702.90	296	237.65	0	6502.5	30041	1800.1	1.47
60	21	1118.42	1	5.53	0	1150.69	149182	1800.3	0.26
	23	984.21	1	2.18	0	984.21	93575	1800.19	0.18
	10	5529.63	2329	387.66	0	5572.33	30854	1800.1	0.7
65	11	5008.36	5579	1760.1	0	6512.43	24543	1800.14	2.3
	25	861.46	1	4.62	0	892.15	99670	1800.31	0.19
70	12	4521.70	1	30.26	0	6362.79	18690	1800.11	2.02
	27	966.49	1	8.34	0	985.1	66734	1800.35	0.16

El Cuadro 5.1 muestra la facilidad del modelo $\mathcal{FC}-1$ para resolver el problema ya que en todas las instancias encontró la solución óptima, en una baja cantidad de nodos explorados y en corto tiempo respecto al otro modelo presentado. Por otro lado, modelo $\mathcal{FC}-2$ tan solo en dos instancias obtuvo una solución óptima, y en

el resto de ellas usaron el tiempo máximo permitido y un valor de GAP promedio igual a 0.814 %. Adicionalmente, se puede notar que el modelo $\mathcal{FC}-2$ tiene problemas al momento de disminuir la cota que proviene de resolver su relajación lineal, puesto que en varias instancias obtiene la misma función objetivo que el modelo $\mathcal{FC}-1$, pero no logra concluir que es la óptima. Además, sin importar el tamaño de la instancia, el modelo $\mathcal{FC}-2$ explora una gran cantidad de nodos, a diferencia del modelo $\mathcal{FC}-1$ que encuentra la mayoría de las soluciones en el nodo raíz. Esto puede ser explicado por las restricciones triangulares del modelo $\mathcal{FC}-1$ que acercan el valor de la solución de la relajación lineal a la solución entera de mejor forma que las restricciones del modelo $\mathcal{FC}-2$.

5.2. Equipartición de grafos en componentes conexas

En esta sección se experimenta con los modelos formulados para equiparticionar un grafo donde cada subconjunto de la partición deba ser conexo. Las formulaciones $\mathcal{FG}-1$ y $\mathcal{FG}-3$ solo dependen de los conjuntos de nodos V y aristas E por lo que su tamaño crece polinomialmente en función de la instancia de entrada. Por otra parte, el modelo $\mathcal{FG}-2$ precisa de los conjuntos separadores Γ_1 o Γ_2 , y por tanto el número de restricciones puede crecer exponencialmente dependiendo de los mismos. Hojny *et al.* (2020) mencionan que, si bien dichos conjuntos son fáciles de encontrar, la cantidad de conjuntos es grande. Debido a este hecho, el modelo $\mathcal{FG}-2$ es manejado utilizando desigualdades tipo lazy asociadas a las restricciones que se encuentran relacionadas con los conjuntos separadores. Lo anterior implica que las restricciones no son incluidas al inicio de la optimización, si no que se añaden cuando una solución entera viola alguna de ellas. En ese caso se incluyen las desigualdades violadas y la optimización continúa. Esto permite una disminución del tamaño del modelo y un manejo adecuado de la instancia desde el punto de vista computacional.

Dado que las formulaciones $\mathcal{FG}-1$, $\mathcal{FG}-2$ y $\mathcal{FG}-3$ pueden ser difíciles de resolver, se plantea diseñar un algoritmo tipo Branch & Cut. La ejecución de los modelos enteros incluyendo diferentes rutinas de separación es comparado inicialmente utilizando las desigualdades planteadas en el capítulo anterior. Para las formulaciones se incluirán las desigualdades de tipo Clique (Teorema 4.5), Corte (Teorema 4.6), Cota (Teorema 4.3), Camino (Teorema 4.4), Separador (Teorema 4.2) y Triángulo (desigualdades 4.9 - 4.11). Para todas las instancias, los algoritmos de separación utilizados son del tipo fuerza bruta y las desigualdades válidas del solver Gurobi fueron deshabilitadas.

Se realiza un primer análisis en instancias con baja cantidad de nodos ($n \leq 20$) con la finalidad de encontrar las desigualdades válidas que más aporten a la solución de los modelos, es decir, identificar el modelo que mejor se comporte respecto al tiempo de ejecución, la cantidad de nodos explorados y el valor del GAP. Con estos resultados, en un segundo experimento se considerarán instancias de mayor tamaño con la configuración de desigualdades que vuelvan más eficientes a las formulaciones.

Los Cuadros 5.2 - 5.6 muestran resultados asociados a grafos de diferente tamaño, densidades y número de componentes conexas en las que debe ser particionado el grafo. Para cada instancia se resolvieron los modelos $\mathcal{FG}-1$, $\mathcal{FG}-2$ y $\mathcal{FG}-3$ incluyendo por separado las desigualdades válidas planteadas.

En cada uno de los cuadros se reporta el resultado de la optimización de las tres formulaciones incluyendo los casos en que se considera solo las desigualdades tipo Clique, solo con desigualdades tipo Corte, agrupando las desigualdades tipo Cota y Camino, solo las desigualdades tipo Separador y solo las desigualdades tipo Triángulo. Finalmente, en las dos últimas filas de cada modelo se plantean los experimentos de no utilizar ninguna desigualdad y el de utilizar todas las desigualdades. Por otro lado, en las cuatro primeras columnas se incluyen el valor de la función objetivo, el número de nodos explorados, el tiempo de ejecución en segundos y la brecha de optimalidad en porcentaje. El resto de valores representan el número de desigualdades válidas incluidas de cada tipo.

Los Cuadros 5.2 y 5.3 difieren por la densidad de su grafo de entrada, esta diferencia permite evidenciar que la desigualdad tipo Camino existe cuando el grafo es disperso. Por el contrario, las desigualdades tipo Clique aparecen cuando el grafo es denso, es decir, cuando la densidad es alta. Además, en estas dos instancias no existieron desigualdades tipo Corte para ser introducidas. Respecto a la desigualdad tipo Cota se puede ver que su inclusión es favorable, pues en los tres modelos con ambas densidades existe una disminución en el tiempo de ejecución respecto al caso sin desigualdades. Usando las desigualdades tipo Separador, se evidencia la gran cantidad de desigualdades que son incluidas en una instancia, y que no existe una mejora respecto al tiempo de ejecución o a la cantidad de nodos explorados, es decir, estas desigualdades son incluidas pero no tienen un efecto positivo sobre los modelos. Finalmente, las desigualdades tipo Triángulo no tienen un gran impacto en el tiempo de ejecución, pero en algunas instancias ayudan a encontrar la solución óptima explorando una menor cantidad de nodos.

Cuadro 5.2: Instancia: $n = 15, k = 6, d = 0.92$.

Desigualdad Válida	F. Obj.	Nodos	Tiempo	GAP	Clique	Corte	Cota	Camino	Separador	Triángulo
<i>FG-1</i>										
Clique	468.65	1860	59.7	0	805	0	0	0	0	0
Corte	468.65	5993	5.56	0	0	0	0	0	0	0
Cota y Camino	468.65	3256	4	0	0	0	1	0	0	0
Separador	468.65	5993	5.88	0	0	0	0	0	0	0
Triángulo	468.65	6151	10.92	0	0	0	0	0	0	50
Sin desigualdades	468.65	5993	5.87	0	0	0	0	0	0	0
Todas las desigualdades	468.65	4726	159.65	0	708	0	1	0	0	364
<i>FG-2</i>										
Clique	468.65	59861	1820.31	0.71	1970	0	0	0	0	0
Corte	468.65	678187	418.35	0	0	0	0	0	0	0
Cota y Camino	468.65	387297	168.25	0	0	0	1	0	0	0
Separador	468.65	687831	567.73	0	0	0	0	0	1227229	0
Triángulo	468.65	689214	961.57	0	0	0	0	0	0	125375
Sin desigualdades	468.65	678187	358.44	0	0	0	0	0	0	0
Todas las desigualdades	468.65	56157	1814.98	0.61	2021	0	1	0	86327	85726
<i>FG-3</i>										
Clique	468.65	33324	1167.73	0	4109	0	0	0	0	0
Corte	468.65	104321	128.51	0	0	0	0	0	0	0
Cota y Camino	468.65	102014	67.04	0	0	0	1	0	0	0
Separador	468.65	103680	138.99	0	0	0	0	0	7	0
Triángulo	468.65	125236	297.9	0	0	0	0	0	0	698
Sin desigualdades	468.65	104321	128.58	0	0	0	0	0	0	0
Todas las desigualdades	468.65	49469	1824.38	17.35	7201	0	1	0	0	980

Por otro lado, el modelo que mejor se comporta respecto al tiempo de ejecución en estas dos instancias es el modelo $FG-1$, ya que independientemente de la desigualdad aplicada y la densidad de la instancia, se resuelve en un menor tiempo que los modelos $FG-2$ y $FG-3$. Además, en todos los modelos existe al menos una desigualdad válida que mejora el tiempo de ejecución respecto al caso sin desigualdades. Por tal motivo, podemos afirmar que todas las desigualdades válidas diseñadas en el presente trabajo son efectivas para resolver el problema de equiparticionamiento de grafos. Cabe mencionar que el caso con todas las desigualdades es el que en general más tiempo se demora, ya que lo reportado incluye el tiempo de los algoritmos de separación de todas las desigualdades válidas. Por otro lado, esta configuración es la que menor cantidad de nodos explorados posee, pues tiene más desigualdades válidas para incluir. Esto demuestra el correcto funcionamiento del algoritmo Branch & Cut y su beneficio.

Para validar los resultados anteriores, dos nuevas instancias donde la cantidad de nodos aumenta y la cantidad de subconjuntos disminuye son consideradas. Así, se tienen dos instancias, con 20 nodos y 5 subconjuntos. La primera usa un grafo con densidad igual a 0.91 (Cuadro 5.4) y la segunda con 0.36 de densidad (Cuadro 5.5). Estas instancias revelan la complejidad del problema, pues con aumentar tan solo cinco nodos, la mayoría de los experimentos no obtienen la solución óptima. Hay que añadir que, cuando el grafo tiene densidad baja el problema se dificulta

Cuadro 5.3: Instancia: $n = 15, k = 6, d = 0.31$.

Desigualdad Válida	F.Obj.	Nodos	Tiempo	GAP	Clique	Corte	Cota	Camino	Separador	Triángulo
<i>FG-1</i>										
Clique	594.68	57062	26.67	0	0	0	0	0	0	0
Corte	594.68	57062	27.41	0	0	0	0	0	0	0
Cota y Camino	594.68	441	0.39	0	0	0	1	114	0	0
Separador	594.68	24804	21.16	0	0	0	0	0	30	0
Triángulo	594.68	42933	24.85	0	0	0	0	0	0	87
Sin desigualdades	594.68	57062	24.05	0	0	0	0	0	0	0
Todas las desigualdades	594.68	556	0.58	0	0	0	1	114	10	17
<i>FG-2</i>										
Clique	594.68	4962	1801.21	16.29	0	0	0	0	0	0
Corte	594.68	4907	1801.2	16.29	0	0	0	0	0	0
Cota y Camino	594.68	80733	149.21	0	0	0	1	114	0	0
Separador	594.68	4487	1801.18	16.29	0	0	0	0	70	0
Triángulo	594.68	4841	1801.11	16.29	0	0	0	0	0	164
Sin desigualdades	594.68	4603	1801.05	16.29	0	0	0	0	0	0
Todas las desigualdades	594.68	78891	120.47	0	0	0	1	114	310	975
<i>FG-3</i>										
Clique	594.68	200681	64.89	0	0	0	0	0	0	0
Corte	594.68	200681	65.55	0	0	0	0	0	0	0
Cota y Camino	594.68	23167	11.12	0	0	0	1	114	0	0
Separador	594.68	191641	124.6	0	0	0	0	0	180	0
Triángulo	594.68	184329	79.77	0	0	0	0	0	0	6865
Sin desigualdades	594.68	200681	48.46	0	0	0	0	0	0	0
Todas las desigualdades	594.68	18915	17.58	0	0	0	1	114	30	2163

aún más. Esto se puede evidenciar en el Cuadro 5.4 y en el Cuadro 5.5 ya que hay más experimentos en los que no se obtiene la solución óptima. Este incremento en la dificultad se produce ya que la cantidad de aristas disminuye y existen menos posibilidades de encontrar una equipartición conexas, dificultando la búsqueda de la misma.

Una diferencia importante respecto a las instancias con 15 nodos es la existencia de las desigualdades válidas de tipo Corte. Estas desigualdades muestran ser de ayuda cuando el grafo es poco denso, pues en el Cuadro 5.5 se ve una disminución en el valor del GAP respecto al caso sin desigualdades. Por otro lado, se sigue manteniendo la dificultad de resolver un grafo disperso frente a un grafo denso. Otra consecuencia en los grafos dispersos, es que el modelo $\mathcal{FG}-2$ explora una baja cantidad de nodos, pues al tener pocas aristas, existen más soluciones que no verifican las desigualdades tipo lazy. En este sentido, el modelo $\mathcal{FG}-2$ invierte su tiempo en incluir las desigualdades necesarias para que la solución sea conexas.

Por otro lado, en el caso de densidad baja, el modelo $\mathcal{FG}-3$ se presenta como el menos eficiente. Esto es evidente, ya que explora gran cantidad de nodos, incluye desigualdades válidas y aun así reporta valores del GAP altos. Además, el modelo $\mathcal{FG}-2$ queda descartado, pues al depender de los conjuntos separadores, su tamaño crece al punto en que se vuelve computacionalmente intratable. Adicionalmente, debido al número de elementos en los conjuntos Γ_1 y Γ_2 en instancias de mayor

Cuadro 5.4: $n = 20, k = 5, d = 0.91$.

Desigualdad Válida	E. Obj.	Nodos	Tiempo	GAP	Clique	Corte	Cota	Camino	Separador	Triángulo
<i>FG-1</i>										
Clique	1144.51	15457	1053.04	0	3857	0	0	0	0	0
Corte	1144.51	22020	471.3	0	0	624	0	0	0	0
Cota y Camino	1144.51	13124	30.36	0	0	0	1	0	0	0
Separador	1144.51	25403	36.21	0	0	0	0	0	0	0
Triángulo	1144.51	26383	81.61	0	0	0	0	0	0	23
Sin desigualdades	1144.51	25403	28.67	0	0	0	0	0	0	0
Todas las desigualdades	1144.51	17857	1771.98	0	3544	620	1	0	0	1397
<i>FG-2</i>										
Clique	1144.51	25038	1800.75	2.8	3976	0	0	0	0	0
Corte	1144.51	83268	1813.53	3.2	0	804	0	0	0	0
Cota y Camino	1144.51	1125214	1079.27	0	0	0	1	0	0	0
Separador	1144.51	914992	1065.71	0	0	0	0	0	113	0
Triángulo	1144.51	638454	1800.08	1.64	0	0	0	0	0	1075
Sin desigualdades	1144.51	914367	814.76	0	0	0	0	0	0	0
Todas las desigualdades	1144.51	18263	1808.23	3.18	4090	794	1	0	4	1851
<i>FG-3</i>										
Clique	1231.77	24410	1828.5	40.91	34267	0	0	0	0	0
Corte	1194.18	95524	1803.78	37.13	0	160378	0	0	0	0
Cota y Camino	1144.51	84285	109.17	0	0	0	1	0	0	0
Separador	1144.51	141732	265.34	0	0	0	0	0	6	0
Triángulo	1144.51	137594	569.98	0	0	0	0	0	0	1283
Sin desigualdades	1144.51	150453	236.3	0	0	0	0	0	0	0
Todas las desigualdades	1179.13	18824	1815.76	27.22	26644	24366	1	0	9	1505

tamaño, las desigualdades válidas tipo Separador también quedan excluidas para los siguientes análisis.

A continuación, se procede a mostrar instancias donde la cantidad de nodos varía entre 22 y 30, y la cantidad de subconjuntos cambia entre 5 y 10. De la misma forma que en las instancias pasadas, se consideran experimentos con grafos usando densidad alta y baja, pues de esto depende la existencia de varios tipos de desigualdades válidas. Estos experimentos se realizan con los modelos $FG-1$ y $FG-3$, cada uno con las desigualdades válidas que mejoran su eficiencia según los análisis anteriores. En las primeras columnas del Cuadro 5.6 se muestran las características de la instancia a resolver. Además, la función objetivo, la cantidad de nodos explorados, el tiempo de ejecución en segundos, la brecha de optimalidad y la cantidad de desigualdades válidas son incluidas para cada instancia. El tiempo máximo de ejecución para los experimentos mencionados se ha extendido a 3600 segundos.

Cuadro 5.5: $n = 20, k = 5, d = 0.36$.

Desigualdad Válida	F.Obj.	Nodos	Tiempo	GAP	Clique	Corte	Cota	Camino	Separador	Triángulo
<i>FG-1</i>										
Clique	765.99	3259549	1800.1	7.6	0	0	0	0	0	0
Corte	765.99	95522	856.02	0	0	726	0	0	0	0
Cota y Camino	765.99	12661	8.71	0	0	0	1	0	0	0
Separador	765.99	1915191	1800.28	8.12	0	0	0	0	360	0
Triángulo	765.99	2376724	1800.32	7.45	0	0	0	0	0	1587
Sin desigualdades	765.99	4152452	1800.02	7.21	0	0	0	0	0	0
Todas las desigualdades	765.99	6243	134.34	0	0	646	1	0	30	184
<i>FG-2</i>										
Clique	1071.83	1460	1802.27	19.71	0	0	0	0	0	0
Corte	1036.71	1201	1861.65	18.86	0	10316	0	0	0	0
Cota y Camino	765.99	3153	1801.4	2.74	0	0	1	0	0	0
Separador	789.8	1477	1802.21	13.81	0	0	0	0	10	0
Triángulo	1071.83	1459	1804.01	19.71	0	0	0	0	0	17
Sin desigualdades	1071.83	1458	1804.32	19.71	0	0	0	0	0	0
Todas las desigualdades	775.96	1952	1802.7	2.95	0	13116	1	0	60	6208
<i>FG-3</i>										
Clique	765.99	2483875	1800.14	61.69	0	0	0	0	0	0
Corte	775.96	202218	1803.86	46.48	0	1206	0	0	0	0
Cota y Camino	765.99	3276018	1569.98	0	0	0	1	0	0	0
Separador	817.49	1520811	1800.15	67.69	0	0	0	0	200	0
Triángulo	815.9	1740087	1800.04	58	0	0	0	0	0	954
Sin desigualdades	765.99	2375132	1800.04	62.05	0	0	0	0	0	0
Todas las desigualdades	765.99	192932	1806.02	12.06	0	2028	1	0	60	442

Cuadro 5.6: Comparación de los modelos $\mathcal{FG}-1$ y $\mathcal{FG}-3$.

Instancia			$\mathcal{FG}-1$					$\mathcal{FG}-3$					Cota y Camino*	
n	k	d	F. Obj.	Nodos	Tiempo	GAP	Clique	F. Obj.	Nodos	Tiempo	GAP	Clique		Corte
22	5	0.93	1438.39	20498	1940.86	0	4569	2247.68	28960	3712.11	66.68	10299	2710	1
		0.33	1037.18	49064	42.37	0	0	1037.18	310550	3608.74	15.95	0	107584	1
24	8	0.93	685.08	21560	3600.43	0.26	4994	1421.02	15376	3676.21	74.36	10381	6496	1
		0.36	872.23	760954	1225.68	0	0	876.5	178079	3608.9	33.14	0	67676	97
26	8	0.9	749.01	22086	3645.93	0.77	6113	1986.66	15327	3667.67	81.21	4177	2756	1
		0.37	691.64	27546	75.12	0	0	873.4	117183	3603.64	34.48	0	45222	1
28	10	0.87	628.71	14960	3654.93	0.38	3628	903.44	11786	3623.22	57.44	5924	0	1
		0.35	802.78	900187	3600.11	0.11	0	843.35	333383	3600.44	31.96	0	0	101
30	10	0.89	806.05	14610	3683.62	0.35	4067	2466.39	8814	3608.25	80.48	3704	2180	1
		0.31	1075.49	215847	715.35	0	0	1507	79030	3614.71	49.54	0	15030	101

El uso del caracter * representa que se aplicaron el mismo número de desigualdades válidas en ambos modelos en el nodo raíz.

Al comparar los modelos $\mathcal{FG}-1$ y $\mathcal{FG}-3$ se puede observar un mejor comportamiento del modelo $\mathcal{FG}-1$, ya que en muchas instancias se obtiene la solución óptima y en el resto de ellas el valor del GAP se aproxima a cero. Por otro lado, la segunda formulación reporta altos valores del GAP en todas las instancias, con un valor promedio del 52.52 %. Un dato interesante al momento de ejecutar el proceso de optimización es que al modelo $\mathcal{FG}-3$ le resulta complicado encontrar tan solo una solución factible. Para concluir se puede ver que el modelo $\mathcal{FG}-3$ no presenta un desempeño satisfactorio para resolver el problema de equiparticionamiento en componentes conexas y es descartado para los experimentos con instancias mayores a 30 nodos.

Gracias a los análisis anteriores se ha podido filtrar el modelo más eficiente entre los formulados para el problema de equiparticionamiento. Esto implica aumentar aun más la cantidad de nodos en las instancias para llevar al extremo al modelo.

En los siguientes cuadros se consideran instancias con $40 \leq n \leq 60$ nodos y $6 \leq k \leq 21$ subconjuntos, que son resueltas con el modelo $\mathcal{FG}-1$ utilizando dos tipos de configuraciones respecto a las desigualdades válidas a las que notaremos por \mathcal{D}_1 y \mathcal{D}_2 . En ambas configuraciones se incluyen las desigualdades tipo Cota, tipo Camino y tipo Clique, ya que los experimentos anteriores mostraron que ayudan a disminuir la cantidad de nodos explorados y el tiempo de ejecución. Además, la configuración \mathcal{D}_1 admite la desigualdad tipo Triángulo, mientras que \mathcal{D}_2 admite a la desigualdad tipo Corte. Estas dos configuraciones son comparadas con el experimento que no incluye desigualdades válidas de ningún tipo. Para cada instancia y configuración, el valor de la función objetivo, el número de nodos explorados, el tiempo de ejecución en segundos, la brecha de optimalidad en porcentaje y el número de desigualdades de cada tipo incluidas en el proceso de solución son reportadas en los siguientes cuadros. Con estas aclaraciones se presentan los resultados para el modelo $\mathcal{FG}-1$.

Cuadro 5.7: Modelo $\mathcal{FG}-1$ para $n = 40$.

$\mathcal{FG}-1$									
Configuración	F. Obj.	Nodos	Tiempo	GAP	Cota	Camino	Clique	Corte	Triángulo
$n = 40, k = 6, d = 0.77$									
\mathcal{D}_1	3037.34	22969	3687.5	3.2	1	0	4124	0	1446
\mathcal{D}_2	4584.8	2986	3661.15	6	1	0	1909	569	0
Sin desigualdades	3405.9	468005	3600.26	3.73	0	0	0	0	0
$n = 40, k = 6, d = 0.15$									
\mathcal{D}_1	2064.95	1212407	3600.09	5.2	1	0	0	0	280
\mathcal{D}_2	2328.2	17800	3730.05	7.98	1	0	0	1581	0
Sin desigualdades	2057.14	3184309	3600.07	19.95	0	0	0	0	0
$n = 40, k = 14, d = 0.8$									
\mathcal{D}_1	1075.73	4402	3601.8	0.86	1	0	0	0	490
\mathcal{D}_2	825.95	4776	3600.42	0.44	1	0	0	0	0
Sin desigualdades	1171.13	5300	3600.35	1.9	0	0	0	0	0
$n = 40, k = 14, d = 0.16$									
\mathcal{D}_1	1593.43	434658	3600.26	2.65	1	2996	0	0	382
\mathcal{D}_2	1593.43	694265	3600.49	2.46	1	2996	0	0	0
Sin desigualdades	1720.69	531709	3600.09	13.54	0	0	0	0	0

Cuadro 5.8: Modelo \mathcal{FG} -1 para $n = 45$.

\mathcal{FG} -1									
Configuración	F. Obj.	Nodos	Tiempo	GAP	Cota	Camino	Clique	Corte	Triángulo
$n = 45, k = 7, d = 0.78$									
\mathcal{D}_1	4134.76	3967	3608.34	4.23	1	0	3016	0	896
\mathcal{D}_2	4942.46	1327	3612.59	5.52	1	0	1111	252	0
Sin desigualdades	3581.96	74674	3600.38	3.88	0	0	0	0	0
$n = 45, k = 7, d = 0.16$									
\mathcal{D}_1	2237.83	702371	3600.23	2.4	1	0	0	0	308
\mathcal{D}_2	-	13286	3600	-	1	0	0	-	0
Sin desigualdades	2504.33	1103065	3600.2	15.68	0	0	0	0	0
$n = 45, k = 16, d = 0.77$									
\mathcal{D}_1	1139.45	1396	3600.79	0.77	1	0	0	0	135
\mathcal{D}_2	1159.47	1491	3601.02	0.78	1	0	0	0	0
Sin desigualdades	1161.23	7675	3600.43	1.43	0	0	0	0	0
$n = 45, k = 16, d = 0.15$									
\mathcal{D}_1	1291.68	194871	3600.43	0.57	1	4832	0	0	260
\mathcal{D}_2	1291.68	168869	3600.52	0.64	1	4832	0	0	0
Sin desigualdades	1626.83	63712	3600.08	12.2	0	0	0	0	0

El uso del caracter - significa que en la optimización no se encontró ninguna solución factible.

Cuadro 5.9: Modelo \mathcal{FG} -1 para $n = 50$.

\mathcal{FG} -1									
Configuración	F. Obj.	Nodos	Tiempo	GAP	Cota	Camino	Clique	Corte	Triángulo
$n = 50, k = 8, d = 0.78$									
\mathcal{D}_1	3834.48	10388	3603.67	3.01	1	0	4365	0	579
\mathcal{D}_2	4968.8	1858	3723.45	4.37	1	0	695	174	0
Sin desigualdades	3689.61	66473	3600.3	3.42	0	0	0	0	0
$n = 50, k = 8, d = 0.15$									
\mathcal{D}_1	2647.99	427477	3600.33	3.41	1	0	0	0	454
\mathcal{D}_2	-	11645	3600	-	1	0	0	-	0
Sin desigualdades	2863.69	1302728	3600.09	16.78	0	0	0	0	0
$n = 50, k = 17, d = 0.79$									
\mathcal{D}_1	1028.3	1522	3600.93	0.49	1	0	0	0	133
\mathcal{D}_2	1068.79	1387	3600.5	0.54	1	0	0	0	0
Sin desigualdades	2927.52	4158	3600.49	2.99	0	0	0	0	0
$n = 50, k = 17, d = 0.28$									
\mathcal{D}_1	2922	4013	3600.34	5.62	1	102	0	0	256
\mathcal{D}_2	1316.06	5581	3600.29	0.8	1	102	0	0	0
Sin desigualdades	-	4116	3600	-	0	0	0	0	0

El uso del caracter - significa que en la optimización no se encontró ninguna solución factible.

Cuadro 5.10: Modelo \mathcal{FG} -1 para $n = 55$.

\mathcal{FG} -1									
Configuración	F. Obj.	Nodos	Tiempo	GAP	Cota	Camino	Clique	Corte	Triángulo
$n = 55, k = 9, d = 0.8$									
\mathcal{D}_1	4595.42	4956	3601.55	2.86	1	0	2500	0	331
\mathcal{D}_2	5965.41	1453	3628.28	4.03	1	0	670	162	0
Sin desigualdades	5741.28	10130	3600.14	4.49	0	0	0	0	0
$n = 55, k = 9, d = 0.15$									
\mathcal{D}_1	2912.69	241584	3600.24	5.13	1	0	0	0	768
\mathcal{D}_2	-	9500	3600	-	1	0	0	-	0
Sin desigualdades	-	640197	3600	-	0	0	0	0	0
$n = 55, k = 19, d = 0.81$									
\mathcal{D}_1	1650.44	466	3600.77	0.86	1	0	0	0	74
\mathcal{D}_2	1609.95	479	3600.81	0.84	1	0	0	0	0
Sin desigualdades	1870.52	4772	3600.49	1.47	0	0	0	0	0
$n = 55, k = 19, d = 0.27$									
\mathcal{D}_1	-	2177	3600	-	1	8	0	0	-
\mathcal{D}_2	-	2456	3600	-	1	8	0	-	0
Sin desigualdades	-	3430	3600	-	0	0	0	0	0

El uso del caracter - significa que en la optimización no se encontró ninguna solución factible.

Cuadro 5.11: Modelo \mathcal{FG} -1 para $n = 60$.

\mathcal{FG} -1									
Configuración	F. Obj.	Nodos	Tiempo	GAP	Cota	Camino	Clique	Corte	Triángulo
$n = 60, k = 10, d = 0.8$									
\mathcal{D}_1	6239.52	2960	3608.13	3.49	1	0	1697	0	161
\mathcal{D}_2	6892.72	1161	3701.93	3.99	1	0	462	90	0
Sin desigualdades	7427.4	4838	3600.7	4.82	0	0	0	0	0
$n = 60, k = 10, d = 0.16$									
\mathcal{D}_1	3622.15	89836	3601.04	7.11	1	0	0	0	1123
\mathcal{D}_2	3189.67	6695	3668.83	5.18	1	0	0	269	0
Sin desigualdades	4190.93	151338	3600.09	18.26	0	0	0	0	0
$n = 60, k = 21, d = 0.81$									
\mathcal{D}_1	1801.44	181	3602.53	0.94	1	0	0	0	22
\mathcal{D}_2	2115.23	167	3601.44	1.18	1	0	0	0	0
Sin desigualdades	2029.76	2824	3600.42	1.47	0	0	0	0	0
$n = 60, k = 21, d = 0.28$									
\mathcal{D}_1	-	1800	3600	-	1	13	0	0	-
\mathcal{D}_2	-	2382	3600	-	1	13	0	-	0
Sin desigualdades	-	3338	3600	-	0	0	0	0	0

El uso del caracter - significa que en la optimización no se encontró ninguna solución factible.

De las 20 instancias presentadas, dos de ellas no se resolvieron mediante ninguna configuración. En 15 instancias sucedió que alguna de las configuraciones \mathcal{D}_1 y \mathcal{D}_2 fue superior al caso sin desigualdades, ya que encontró una solución con menor valor en la función objetivo. En las tres instancias faltantes, el caso sin desigualdades obtuvo una mejor solución, pero en dos de ellos el valor del GAP es mayor. Cabe notar que en la mayoría de las instancias, las configuraciones \mathcal{D}_1 y \mathcal{D}_2 obtuvieron sus soluciones con menor cantidad de nodos explorados respecto al caso sin desigualdades.

En los reportes anteriores se evidencia que en instancias grandes las desigualdades válidas propuestas en el presente trabajo producen una mejora en la reducción del valor del GAP y reducción en la cantidad de nodos explorados. Comparando las configuraciones \mathcal{D}_1 y \mathcal{D}_2 , se muestra que \mathcal{D}_1 en 12 ocasiones encontró una mejor solución y menor valor del GAP que \mathcal{D}_2 . Esto pone en evidencia que las desigualdades tipo Triángulo tienen un impacto positivo en el modelo. En efecto, la cantidad de nodos explorados por la configuración \mathcal{D}_2 es menor respecto a \mathcal{D}_1 , lo que apoya la idea de necesitar un mejor algoritmo de separación. Para finalizar, es evidente la funcionalidad del modelo $\mathcal{FG}-1$ ya que los valores en la columna GAP asociados a cada una de las instancias con las configuraciones \mathcal{D}_1 y \mathcal{D}_2 son bajos, en promedio 3%, y en algunos casos menor que 1%. Esto implica que se dispone de una formulación estable y un algoritmo exacto tipo Branch & Cut adecuado para resolver el problema de equiparticionamiento de grafos generales en componentes conexas.

Capítulo 6

Conclusiones y Recomendaciones

En el presente trabajo se estudia el problema de equiparticionamiento de grafos en un número fijo de componentes conexas. En ese sentido, se presentaron cinco modelos de programación lineal entera, dos de ellos para el caso sobre grafos completos y tres para el caso sobre grafos generales. Dado que el caso sobre grafos generales es más complejo se produjeron varias familias de desigualdades válidas y fueron incluidas en un algoritmo exacto tipo Branch & Cut para resolver el problema.

En el caso de grafos completos se evidenció que la formulación $\mathcal{FC}-1$ que utiliza variables sobre las aristas y usa desigualdades triangulares fue superior a la formulación $\mathcal{FC}-2$ que posee variables sobre nodos y aristas. Esto se concluye ya que la formulación $\mathcal{FC}-1$ permite resolver el problema de equiparticionamiento de grafos completos reportando mejores valores en el tiempo de ejecución, cantidad de nodos explorados y valor de las brechas de optimalidad en las diferentes instancias experimentadas.

Como se mencionó anteriormente, para el caso de grafos generales se propuso un algoritmo tipo Branch & Cut, incluyendo varias familias de desigualdades válidas. La efectividad de todas ellas fueron probadas al incluirlas en el proceso de solución y obtener una significativa reducción en el número de nodos explorados, tiempo de ejecución y brecha de optimalidad. Al comparar las tres formulaciones se pudo concluir que la formulación $\mathcal{FG}-1$ es la que se comporta de mejor forma, al permitir resolver instancias de hasta 60 nodos. Con esta formulación, diferentes configuraciones fueron probadas, permitiendo identificar la mejor estrategia de solución y la efectividad del algoritmo exacto de solución.

En resumen, se puede concluir que la inclusión de las desigualdades válidas presentadas en este trabajo son útiles para abordar el problema de equiparticionamiento

to de grafos en componentes conexas. Asimismo, las formulaciones $\mathcal{FC}-1$ y $\mathcal{FG}-1$ son las que se comportan de mejor forma respecto a las presentadas. Finalmente, este trabajo puede ser extendido al problema de equiparticionamiento adicionando pesos sobre los nodos, de modo que no solo se busque un equiparticionamiento minimizando el costo total sobre las aristas, si no también un equiparticionamiento donde las componentes conexas sean balanceadas respecto al peso impuesto sobre los nodos.

Bibliografía

- Bernhard, K. y Vygen, J. (2001). *Combinatorial optimization: Theory and algorithms*, volumen 41.
- B.W. Kernighan y S. Lin (1970). An effective heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307.
- Chopra, S. y Rao, M. R. (1993). The partition problem. *Mathematical Programming*, 59(1-3):87–115.
- Dantzig, G. (1951). Application of the simplex method to a transportation problem. *Activity Analysis of Production and Allocation*. Koopmans, T.C., Ed., John Wiley and Sons, New York, pp. 359–373.
- Dantzig, G., Fulkerson, R., y Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Operations Research*, 2(4):393–410.
- Delling, D., Fleischman, D., Goldberg, A. V., Razenshteyn, I., y Werneck, R. F. (2015). An exact combinatorial algorithm for minimum graph bisection. *Mathematical Programming*, 153(2):417–458.
- Dilkina, B. y Gomes, C. P. (2010). Solving connected subgraph problems in wildlife conservation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6140 LNCS:102–116.
- Fan, N. y Pardalos, P. M. (2010). Linear and quadratic programming approaches for the general graph partitioning problem. pp. 57–71.
- Gomory, R. E. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278.
- Grötschel, M., Wakabayashi, Y. (1989). A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 45:59–96.

- Hojny, C., Joormann, I., Lüthen, H., y Schmidt, M. (2020). Mixed-integer programming techniques for the connected max-k-cut problem. *Mathematical Programming Computation*.
- Jünger, M., Reinelt, G., y Pulleyblank, W. R. (1985). On partitioning the edges of graphs into connected subgraphs. *Journal of Graph Theory*, 9(4):539–549.
- Karmarkar, N. (1984). A new polynomial time algorithm for linear programming. *Combinatorica*, (4):373–395.
- Karypis, G. y Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal of Scientific Computing*, 20(1):359–392.
- Khachiyan, L. (1979). Polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, pp. 191–194.
- Labbé, M. y Özsoy, F. A. (2010). Size-constrained graph partitioning polytopes. *Discrete Mathematics*, 310(24):3473–3493.
- Miyazawa, F. K., Moura, P. F. S., Ota, M. J., y Wakabayashi, Y. (2019). Integer programming approaches to balanced connected k-partition. pp. 1–16.
- Recalde, D., Severín, D., Torres, R., y Vaca, P. (2016). Balanced partition of a graph for football team realignment in ecuador. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9849 LNCS:357–368.
- Recalde, D., Severín, D., Torres, R., y Vaca, P. (2018). An exact approach for the balanced k-way partitioning problem with weight constraints and its application to sports team realignment. *Journal of Combinatorial Optimization*, 36:916–936.
- Sanders, P. y Schulz, C. (2012). Distributed evolutionary graph partitioning. *Proceedings of the Workshop on Algorithm Engineering and Experiments*, pp. 16–29.
- Shin, Y. Y. y Koh, J. S. (1998). An algorithm for generating minimal cutsets of undirected graphs. *Journal of Applied Mathematics and Computing*, 5(3):681–693.
- Wang, Y., Buchanan, A., y Butenko, S. (2017). *On imposing connectivity constraints in integer programs*, volumen 166. Springer Berlin Heidelberg.

Anexos

Código 6.1: Paquetes.

```
1 from gurobipy import *
2 import matplotlib.pyplot as plt
3 import random as rm
4 from math import *
5 import networkx as nx
6 import pylab
7 from networkx.algorithms.approximation import connectivity
8 from networkx.algorithms.connectivity import minimum_st_node_cut
9 from networkx.algorithms.connectivity import minimum_st_edge_cut
10 from networkx.algorithms.connectivity import minimum_node_cut
11 from networkx.algorithms.flow import maximum_flow
12 import itertools
13 import pandas as pd
```

Código 6.2: Formulación \mathcal{FC} -1.

```
1
2 Instancia = [(60,9),(60,21),(65,10),(60,23),
3 (65,11),(65,25),(70,12),(70,27)]
4
5 def Modelo_FC1(Instancia):
6     h = 60
7     Reportes_Modelo_FC1={}
8     for n1,k1 in Instancia:
9         for i in range(4):
10             print(h+1)
11             n = n1
12             k = k1
13             sem = h
14             rm.seed(sem)
15             h=h+1
16             # Nodos del grafo
17             V = tuplelist(range(1,n+1))
```

```

18     d_prim = {}
19     d_barra = {}
20     coordx={}
21     d1 = {}
22     d1_barra={}
23     coordy={}
24     for i in V:
25         a = rm.randint(0,200)
26         b = rm.randint(0,200)
27         coordx[i] = a
28         coordy[i] = b
29
30     for i in V:
31         for j in range(i,n+1):
32             if i != j:
33                 if rm.randint(0,100)<101:
34                     d_prim[(i,j)]=
35                     round(math.sqrt((coordx[i] - coordx[j])**2
36                     + (coordy[i] - coordy[j])**2),2)
37                     d1[(i,j)] =
38                     round(math.sqrt((coordx[i] - coordx[j])**2
39                     + (coordy[i] - coordy[j])**2),2)
40                     d_prim[(j,i)]=
41                     round(math.sqrt((coordx[i] - coordx[j])**2
42                     + (coordy[i] - coordy[j])**2),2)
43                 else:
44                     d_barra[(i,j)] = 0
45                     d1_barra[(i,j)] =
46                     round(math.sqrt((coordx[i] - coordx[j])**2
47                     + (coordy[i] - coordy[j])**2),2)
48                     d_barra[(j,i)] = 0
49
50
51     posicion = {}
52     for i in V:
53         posicion[i]=[coordx[i],coordy[i]]
54     d={**d_prim,**d_barra}
55     d = tupledict(d)
56     E = d.keys()
57     E_barra = d_barra.keys()
58     E_prim = d_prim.keys()
59     E_1 = d1.keys()
60
61     G_prim = nx.Graph()

```

```

62     for i in V:
63         G_prim.add_node(i)
64     for i in E_prim:
65         G_prim.add_edge(i[0],i[1],weight=d[i])
66     for i in E_prim:
67         G_prim.edges[i[0],i[1]]["Costo"] = d_prim[i]
68
69
70     ## MODELO
71     mC_1 = Model()
72
73     ## Variables
74     x = mC_1.addVars(E, name="x", vtype=GRB.BINARY);
75     ## Funcion Objetivo
76
77     mC_1.setObjective( x.prod(d, '*'), GRB.MINIMIZE);
78
79     ## Restricciones
80
81     mC_1.addConstrs( ( x[i,j]+x[j,l]-x[i,l] <= 1
82     for i in V for j in V for l in V if i<j if j<l ),
83     name="res_4");
84     mC_1.addConstrs( ( x[i,j]-x[j,l]+x[i,l] <= 1
85     for i in V for j in V for l in V if i<j if j<l ),
86     name="res_5");
87     mC_1.addConstrs( ( -x[i,j]+x[j,l]+x[i,l] <= 1
88     for i in V for j in V for l in V if i<j if j<l ),
89     name="res_6");
90
91     if n % k == 0:
92         mC_1.addConstrs( ( x.sum(i, '*') == (n/k)-1 for i in V),
93         name="res_10");
94     else:
95         FL = math.floor(n/k)
96         FU = math.ceil(n/k)
97         r = n - k*FL
98         betha_nk = ((r*FU*(FU-1))/2) + (((k-r)*FL*(FL-1))/2)
99         mC_1.addConstrs( ( x.sum(i, '*') + 1 <= FU for i in V),
100         name="res_7_1");
101         mC_1.addConstrs( ( x.sum(i, '*') + 1 >= FL for i in V),
102         name="res_7_2");
103         mC_1.addConstr( ( x.sum(*, '*') == 2*betha_nk ),
104         name="res_12");
105

```

```

106     mC_1.addConstrs( ( x[i,j] - x[j,i] == 0
107     for i in V for j in V if i!=j), name="res_extra");
108     mC_1.params.TimeLimit = 1800
109
110
111     mC_1.update()
112     mC_1.optimize()
113     vx = mC_1 .getAttr('x', x)
114
115     Tiempo= round(mC_1.runtime,2)
116     p1 = round(sum(d_prim[i,j] for i,j in E_prim)/2,2)
117     Valor_optimo = round(mC_1.getObjective().getValue()/2,2)
118     Nodos_explorados = mC_1.NodeCount
119     Iteraciones_simplex = mC_1.iterCount
120     GAP = round(mC_1.MIPGap*100,2)
121     Reportes_Modelo_FC1[n1,k1,sem]=[p1,Valor_optimo,
122     Nodos_explorados,Iteraciones_simplex,Tiempo,GAP]
123
124     del mC_1
125     objeto = pd.DataFrame.from_dict(Reportes_Modelo_FC1,
126     orient='index').rename(columns={0: 'Peso_Maximo',
127     1: 'Funcion_Objetivo',2: 'Nodos_explorados',
128     3: 'Iteraciones_simplex',4: 'Tiempo',5: 'GAP'})
129     objeto.to_excel('Reportes_Modelo_FC1_supergrandes.xlsx',
130     sheet_name='Reporte')
131     return(Reportes_Modelo_FC1)
132
133 Reportes_Modelo_FC1 = Modelo_FC1(Instancia)

```

Código 6.3: Formulación $FC-2$.

```

1 Modelo_FC2 = {}
2 Instancia = [(60,9),(60,21),(65,10),(60,23),
3 (65,11),(65,25),(70,12),(70,27)]
4 h = 60
5 for n1,k1 in Instancia:
6     print("Instancia",n1,k1)
7     for i in range(4):
8         print(h+1)
9         n = n1
10        k = k1
11        sem = h
12        rm.seed(h)
13        h=h+1
14

```

```

15     # Nodos del grafo
16     V = tuplelist(range(1,n+1))
17
18     d_prim = {}
19     d_barra = {}
20     coordx={}
21     d1 = {}
22     d1_barra={}
23     coordy={}
24     for i in V:
25         a = rm.randint(0,200)
26         b = rm.randint(0,200)
27         coordx[i] = a
28         coordy[i] = b
29
30     for i in V:
31         for j in range(i,n+1):
32             if i != j:
33                 if rm.randint(0,100)<101:
34                     d_prim[(i,j)]=
35                     round(math.sqrt((coordx[i] - coordx[j])**2
36                     + (coordy[i] - coordy[j])**2),2)
37                     d1[(i,j)] =
38                     round(math.sqrt((coordx[i] - coordx[j])**2
39                     + (coordy[i] - coordy[j])**2),2)
40                     d_prim[(j,i)]=
41                     round(math.sqrt((coordx[i] - coordx[j])**2
42                     + (coordy[i] - coordy[j])**2),2)
43                 else:
44                     d_barra[(i,j)] = 0
45                     d1_barra[(i,j)] =
46                     round(math.sqrt((coordx[i] - coordx[j])**2
47                     + (coordy[i] - coordy[j])**2),2)
48                     d_barra[(j,i)] = 0
49
50
51     posicion = {}
52     for i in V:
53         posicion[i]=[coordx[i],coordy[i]]
54
55     d={**d_prim,**d_barra}
56     d = tupledict(d)
57     E = d.keys()
58     E_barra = d_barra.keys()

```

```

59 E_prim = d_prim.keys()
60 E_1 = d1.keys()
61
62 G_prim = nx.Graph()
63 for i in V:
64     G_prim.add_node(i)
65 for i in E_prim:
66     G_prim.add_edge(i[0],i[1],weight=d[i])
67 for i in E_prim:
68     G_prim.edges[i[0],i[1]]["Costo"] = d_prim[i]
69
70
71 ## MODELO
72 K = range(1,k+1)
73 mC_2 = Model()
74 x=mC_2.addVars(V,K, name="x", vtype=GRB.BINARY);
75 y=mC_2.addVars(E_1, name="y", vtype=GRB.BINARY);
76
77 mC_2.setObjective( ( quicksum( y[u,v]*d_prim[u,v]
78 for u,v in E_1) ) , GRB.MAXIMIZE);
79 mC_2.addConstrs( ( x.sum(v, '*') == 1 for v in V ),
80 name="res_1b");
81 mC_2.addConstrs( ( x[u,i]+x[v,i]+y[u,v] <= 2
82 for u,v in E_1 for i in K ), name="res_1e");
83
84 if n %k == 0:
85     mC_2.addConstrs( ( x.sum('*',i) == (n/k) for i in K),
86 name="res_3parti");
87 else:
88     mC_2.addConstrs( ( x.sum('*',i) <= math.ceil(n/k)
89 for i in K), name="res_3partil");
90     mC_2.addConstrs( ( x.sum('*',i) >= math.floor(n/k)
91 for i in K), name="res_3parti2");
92
93 mC_2.update()
94 mC_2.params.TimeLimit = 1800
95 mC_2.optimize()
96 vy = mC_2.getAttr('x', y)
97
98 Tiempo= round(mC_2.runtime,2)
99 p1 = round(sum(d_prim[i,j] for i,j in E_prim)/2,2)
100 Valor_optimo = round(p1-mC_2.getObjective().getValue(),2)
101 Nodos_explorados = mC_2.NodeCount
102 Iteraciones_simplex = mC_2.iterCount

```

```

103     GAP = round(mC_2.MIPGap*100,2)
104     Modelo_FC2[n1,k1,sem]=[Valor_optimo ,
105     Nodos_explorados ,Iteraciones_simplex ,Tiempo,GAP]
106
107     del mC_2
108
109 Reportes_Modelo_FC2 = pd.DataFrame.from_dict(Modelo_FC2 ,
110 orient='index').rename(columns={0: 'Valor_optimo' ,
111 1: 'Nodos_explorados' ,2: 'Iteraciones_simplex' ,3: 'Tiempo' ,4: 'GAP' })
112 Reportes_Modelo_FC1.to_excel('Reportes_Modelo_FC1_supergrandes.xlsx'
113 , sheet_name='Reporte')

```

Código 6.4: Algoritmo para generar Γ_1 .

```

1
2 def fun_aux(Grafo_primal ,k1 ,u ,v):
3     ayuda = []
4     conjunto = list(range(1 ,len(Grafo_primal.nodes)+1))
5     conjunto.remove(u)
6     conjunto.remove(v)
7     combi = list(itertools.combinations(conjunto ,k1))
8     for separador in combi:
9         aux = nx.Graph()
10        for i1 in range(1 ,len(Grafo_primal.nodes)+1):
11            aux.add_node(i1)
12        for i2 ,j2 in Grafo_primal.edges():
13            aux.add_edge(i2 ,j2)
14        aux.remove_nodes_from(list(separador))
15        if nx.has_path(aux , source=u , target=v) == False:
16            ayuda.append(list(separador))
17        del aux
18    del separador ,conjunto ,combi
19    return ayuda
20
21
22 def Gamma(Grafo_primal ,Grafo_faltante):
23     final = {}
24     print("Cantidad de aristas faltantes " ,len(Grafo_faltante.edges()))
25     contar = 1
26     for u ,v in Grafo_faltante.edges():
27         primero = []
28         k1 = connectivity.local_node_connectivity(Grafo_primal , u , v)
29         maximo_de = len(Grafo_primal.nodes())-2
30         for i in range(k1 ,maximo_de+1):
31             aux = fun_aux(Grafo_primal ,i ,u ,v)

```

```

32     primero = primero + aux
33     print("Arista actual ", contar, "uv", u, v, "k", k1)
34     contar = contar + 1
35     final[(u,v)] = primero
36     del primero, contar, k1, maximo_de, aux, u, v, i
37
38     return final
39
40 Gamma1 = Gamma(G_prim, G_falt)

```

Código 6.5: Algoritmo para generar Γ_2 propuesto en Shin y Koh (1998).

```

1
2 def test_comp_connect(Grafo, T):
3     Componentes = list(nx.connected_components(Grafo))
4     vacio = set()
5     for i in Componentes:
6         if T <= i:
7             return i
8     return vacio
9
10 def Adyacentes_conjunto(Grafo, V_s):
11     res = set()
12     for i in V_s:
13         if i in Grafo.nodes():
14             res.update(list(Grafo.adj[i]))
15     return res - V_s
16
17 def Incidentes_aristas(Grafo, V_s):
18     E_c = []
19
20     G_aux1 = nx.Graph()
21     for i1 in Grafo.nodes():
22         G_aux1.add_node(i1)
23     for i2, j2 in Grafo.edges():
24         G_aux1.add_edge(i2, j2)
25     G_aux1.remove_nodes_from(set(Grafo.nodes()) - V_s)
26
27     EV_s = G_aux1.edges()
28     for i, j in Grafo.edges():
29         if (i, j) not in EV_s:
30             if ({i} <= V_s or {j} <= V_s):
31                 E_c.append([i, j])
32     del G_aux1
33     return E_c

```



```

34
35 def EGCUT(Grafo1, V_s, T, separadores):
36     Grafo = nx.Graph()
37     for i3 in Grafo1.nodes():
38         Grafo.add_node(i3)
39     for i4, j4 in Grafo1.edges():
40         Grafo.add_edge(i4, j4)
41
42     V = set(Grafo.nodes())
43
44     V_x = Adyacentes_conjunto(Grafo, V_s)
45
46     ### GRAFO AUXILIAR
47     G_aux = nx.Graph()
48     for i1 in Grafo.nodes():
49         G_aux.add_node(i1)
50     for i2, j2 in Grafo.edges():
51         G_aux.add_edge(i2, j2)
52     G_aux.remove_nodes_from(V_s)
53
54     V_t = test_comp_connect(G_aux, T)
55     del G_aux
56     Z = (V-V_s)-V_t
57     if len(Z & T) == 0:
58         V_s = V_s | Z
59         V_x = V_x - Z
60         E_c = Incidentes_aristas(Grafo, V_s)
61         separadores.append(E_c)
62         T_prima = set()
63
64         while len(V_x - T) != 0:
65             v = [i for i in V_x-T][0]
66             V_x.discard(v)
67             EGCUT(Grafo, V_s | {v}, T | T_prima, separadores)
68             T_prima = T_prima | {v}
69     return "TERMINAMOS"
70
71 G = nx.Graph()
72 for i in range(1,19):
73     G.add_node(i)
74 aristas = [(1,9),(1,10),(1,3),(1,2),(2,11),(2,12),(11,12),
75 (2,5),(3,5),(3,4),(4,7),(5,7),(5,6),(6,8),(7,8),(8,14),
76 (8,13),(15,17),(15,16),(16,18),(17,18)]
77

```

```

78 for i,j in aristas:
79     G.add_edge(i,j)
80
81 Gamma2 = {}
82 for i in range(1,19):
83     for j in range(1,19):
84         if (i,j) not in G.edges():
85             separadores = []
86             EGCUT(G,{i},{j},separadores)
87             Gamma2[(i,j)] = separadores

```

Código 6.6: Cliques de tamaño k .

```

1 def cliques_tamaño_k(G, k, tope_cliques):
2     contador = 0
3     resultado = []
4     for clique in nx.find_cliques(G):
5         if len(clique) == k:
6             resultado.append(list(clique))
7             contador = contador + 1
8         elif len(clique) > k:
9             resultado = resultado +
10             list(itertools.combinations(clique, k))
11             contador = contador +
12             len(list(itertools.combinations(clique, k)))
13         if contador > tope_cliques:
14             return resultado
15     return resultado
16
17 maximo = 5000
18 cliques_k_mas_1 = cliques_tamaño_k(G_prim, k+1, maximo)

```

Código 6.7: Caminos con la cantidad de nodos mayor a $\lceil n/k \rceil$.

```

1 caminos = []
2 for u,v in G_falt.edges():
3     A = nx.shortest_path(G_prim, source=u, target=v)
4     if len(A) > math.ceil(n/k):
5         caminos.append([u,v])
6 print("Almacenados {} caminos".format(len(caminos)))

```

Código 6.8: Cortes de tamaño 2 y 3 y triángulos.

```

1 cortes_de_3 = []
2 cortes_de_2 = []

```

```

3 triadas = []
4 for t1 in G_prim.nodes():
5     for t2 in G_prim.nodes():
6         if t1 < t2:
7             cortes_de_2.append([t1, t2])
8             for t3 in G_prim.nodes():
9                 if t2 < t3:
10                    cortes_de_3.append([t1, t2, t3])
11                    if (t1, t2) in G_prim.edges()
12                    and (t2, t3) in G_prim.edges()
13                    and (t3, t1) in G_prim.edges():
14                        triadas.append([t1, t2, t3])

```

Código 6.9: Instancia para los modelos $\mathcal{FG-i}$.

```

1 n = 25
2 k = 5
3 densidad = 40
4 rm.seed(200)
5
6 # Nodos del grafo
7 V = tuplelist(range(1,n+1))
8
9 d_prim = {}
10 d_barra = {}
11 coordx={}
12 d1 = {}
13 d1_barra={}
14 coordy={}
15 for i in V:
16     a = rm.randint(0,200)
17     b = rm.randint(0,200)
18     coordx[i] = a
19     coordy[i] = b
20
21 for i in V:
22     for j in range(i,n+1):
23         if i != j:
24             if rm.randint(0,100)< densidad:
25                 d_prim[(i, j)]= round(math.sqrt((coordx[i]
26                 - coordx[j])**2 + (coordy[i] - coordy[j])**2),2)
27                 d1[(i, j)] = round(math.sqrt((coordx[i]
28                 - coordx[j])**2 + (coordy[i] - coordy[j])**2),2)
29                 d_prim[(j, i)]= round(math.sqrt((coordx[i]
30                 - coordx[j])**2 + (coordy[i] - coordy[j])**2),2)

```

```

31         else:
32             d_barra[(i, j)] = 0
33             d1_barra[(i, j)] = round(math.sqrt((coordx[i]
34             - coordx[j])**2 + (coordy[i] - coordy[j])**2), 2)
35             d_barra[(j, i)] = 0
36
37
38 posicion = {}
39 for i in V:
40     posicion[i] = [coordx[i], coordy[i]]
41
42 d = {**d_prim, **d_barra}
43 d = tupledict(d)
44 E = d.keys()
45 E_barra = d_barra.keys()
46 E_prim = d_prim.keys()
47 E_1 = d1.keys()
48 E_1_falt = d1_barra.keys()
49
50 G_prim = nx.Graph()
51 for i in V:
52     G_prim.add_node(i)
53 for i in E_prim:
54     G_prim.add_edge(i[0], i[1], capacity=1)
55 for i in E_barra:
56     G_prim.edges[i[0], i[1]]["Costo"] = d_barra[i]
57
58 G_falt = nx.Graph()
59 for i in V:
60     G_falt.add_node(i)
61 for i in E_1_falt:
62     G_falt.add_edge(i[0], i[1], capacity=1)
63
64 edge_labels = dict(d_prim)
65 densidad = (len(d_prim)) / (n * (n - 1))
66
67 pi = round(sum(d_prim[i, j] for i, j in E_prim) / 2, 2)

```

Código 6.10: Función callback para el modelo $FG-1$.

```

1 def callback_flujo(model, where):
2
3     if where == GRB.Callback.MIPNODE:
4         vy = model.cbGetNodeRel(model._y)
5         vx = model.cbGetNodeRel(model._x)

```

```

6      maximo_por_solucion = 3
7      contador1 = 0
8      contador2 = 0
9      contador3 = 0
10     contador4 = 0
11     contador5 = 0
12     tol = 0.1
13
14     # Tipo Corte
15     if math.floor(model._n/len(model._K)) >= 3:
16         for a,b in model._cortes_de_2:
17             if sum([(1-vy[u,v]) for u,v in model._E_prim
18                     if u in [a,b] and v not in [a,b] ])<1-tol:
19                 model.cbCut( (quicksum((1- model._y[u,v]) for u,v in
20 model._E_prim if u in [a,b] and v not in [a,b] )
21 >=1))
22                 model._res_lazy_flujo_corte =
23                 model._res_lazy_flujo_corte + 1
24                 contador5 = contador5 +1
25                 if contador5 >= maximo_por_solucion:
26                     break
27
28     if math.floor(model._n/len(model._K)) >= 4:
29         for a1,b1,c1 in model._cortes_de_3:
30             if sum([(1-vy[u,v]) for u,v in model._E_prim
31                     if u in [a1,b1,c1]
32                     and v not in [a1,b1,c1] ])<1-tol:
33                 model.cbCut( (quicksum((1- model._y[u,v]) for u,v in
34 model._E_prim if u in [a1,b1,c1]and v not in
35 [a1,b1,c1] )>=1))
36                 model._res_lazy_flujo_corte =
37                 model._res_lazy_flujo_corte + 1
38                 contador5 = contador5 +1
39                 if contador5 >= maximo_por_solucion:
40                     break
41
42
43     # Tipo Triángulo
44     if len(model._K)>2:
45         for v,w,u in model._triadas:
46             if vy[v,w]-vy[w,u]-vy[v,u] > 0+tol:
47                 model.cbCut(model._y[v,w]-model._y[w,u]-
48 model._y[v,u] <= 0)
49                 model._res_lazy_flujo_triángulos=

```

```

50         model._res_lazy_flujo_triangulos + 1
51         contador1 = contador1 + 1
52         if contador1 >= maximo_por_solucion:
53             break
54     if -vy[v,w]+vy[w,u]-vy[v,u] > 0+tol:
55         model.cbCut(-model._y[v,w]+model._y[w,u]-
56         model._y[v,u] <= 0)
57         model._res_lazy_flujo_triangulos=
58         model._res_lazy_flujo_triangulos + 1
59         contador1 = contador1 + 1
60         if contador1 >= maximo_por_solucion:
61             break
62     if -vy[v,w]-vy[w,u]+vy[v,u] > 0+tol:
63         model.cbCut(-model._y[v,w]-model._y[w,u]+
64         model._y[v,u] <= 0)
65         model._res_lazy_flujo_triangulos
66         =model._res_lazy_flujo_triangulos + 1
67         contador1 = contador1 + 1
68         if contador1 >= maximo_por_solucion:
69             break
70
71     # Tipo Clique
72     k1 = len(model._K)
73     cota = (factorial(k1+1) /
74     (factorial(2) * factorial((k1+1) - 2) ))-1
75     for clique in model._cliques_k_mas_1:
76         if sum([vy[u,v] for u in clique for v in clique
77         if (u,v) in model._E_prim ]) > (2*cota)+tol:
78             model.cbCut( quicksum(model._y[u,v] for u in clique
79             for v in clique if (u,v) in model._E_prim ) <= 2*cota)
80             model._res_lazy_flujo_cliques =
81             model._res_lazy_flujo_cliques + 1
82             contador2 = contador2 + 1
83             if contador2 >= maximo_por_solucion:
84                 break
85
86     # Tipo Separador
87     for i,j in model._Grafofaltante.edges():
88         for k in model._K:
89             if vx[i,k]+vx[j,k] >= 2+tol
90             and vx[i,k] > 0 and vx[j,k] > 0:
91                 for sep in model._Gamma_nodos[(i,j)]:
92                     if vx[i,k]+vx[j,k] -
93                     sum([vx[h,k] for h in sep ]) > 1+tol:

```

```

94         model.cbCut( model._x[i,k]+model._x[j,k]-
95         quicksum(model._x[a,k] for a in sep) <=1 );
96         model._res_lazy_flujo =
97         model._res_lazy_flujo + 1
98         contador3 = contador3 + 1
99         if contador3 >= maximo_por_solucion:
100             break
101         if contador3 >= maximo_por_solucion:
102             break
103     if contador3 >= maximo_por_solucion:
104         break
105
106 del contador1 ,contador2 ,contador3 ,contador4 ,contador5

```

Código 6.11: Función callback para el modelo \mathcal{FG} -2.

```

1 def callback_sep(model, where):
2     if where == GRB.Callback.MIPNODE:
3         vy = model.cbGetNodeRel(model._y)
4         vx = model.cbGetNodeRel(model._x)
5         maximo_por_solucion = 10
6         contador1 = 0
7         contador2 = 0
8         contador3 = 0
9         contador4 = 0
10        contador5 = 0
11        tol=0.1
12
13
14        # Tipo Corte
15        if math.floor(model._n/len(model._K)) >= 3:
16            for a in range(1,model._n):
17                for b in range(1,model._n):
18                    if a!=b:
19                        if sum([(1-vy[u,v]) for u,v in model._E_prim
20                            if u in [a,b] and v not in [a,b] ])<1-tol:
21                            model.cbCut( (quicksum((1- model._y[u,v])
22                                for u,v in model._E_prim
23                                if u in [a,b] and v not in [a,b] ) >=1))
24                            model._res_lazy_sep_corte =
25                            model._res_lazy_sep_corte + 1
26                            contador5 = contador5 +1
27                            if contador5 >= maximo_por_solucion:
28                                break
29                    if contador5 >= maximo_por_solucion:

```

```

30     break
31
32     if math.floor(model._n/len(model._K)) >= 4:
33         for c in range(1,model._n):
34             if b!=c:
35                 if sum([(1-vy[u,v]) for u,v in
36 model._E_prim if u in [a,b,c] and
37 v not in [a,b,c] ])<1-tol:
38                     model.cbCut((quicksum((1-
39 model._y[u,v]) for u,v in
40 model._E_prim if u in [a,b,c]
41 and v not in [a,b,c] )>=1))
42 model._res_lazy_sep_corte =
43 model._res_lazy_sep_corte + 1
44 contador5 = contador5 + 1
45                     if contador5 >= maximo_por_solucion:
46                         break
47                 if contador5 >= maximo_por_solucion:
48                     break
49             if contador5 >= maximo_por_solucion:
50                 break
51
52 # Tipo Triángulo
53 if len(model._K)>2:
54     for v,w,u in model._triadas:
55         if vy[v,w]-vy[w,u]-vy[v,u] > 0+tol:
56             model.cbCut(model._y[v,w]-model._y[w,u]-
57 model._y[v,u] <= 0)
58             model._res_lazy_sep_triángulos =
59 model._res_lazy_sep_triángulos + 1
60             contador1 = contador1 + 1
61             if contador1 >= maximo_por_solucion:
62                 break
63         if -vy[v,w]+vy[w,u]-vy[v,u] > 0+tol:
64             model.cbCut(-model._y[v,w]+model._y[w,u]-
65 model._y[v,u] <= 0)
66             model._res_lazy_sep_triángulos =
67 model._res_lazy_sep_triángulos + 1
68             contador1 = contador1 + 1
69             if contador1 >= maximo_por_solucion:
70                 break
71         if -vy[v,w]-vy[w,u]+vy[v,u] > 0+tol:
72             model.cbCut(-model._y[v,w]-model._y[w,u]+
73 model._y[v,u] <= 0)

```



```

74         model._res_lazy_sep_triangulos =
75         model._res_lazy_sep_triangulos + 1
76         contador1 = contador1 + 1
77         if contador1 >= maximo_por_solucion:
78             break
79
80     #Tipo Clique
81     k1 = len(model._K)
82     cota = (factorial(k1+1) /
83     (factorial(2) * factorial((k1+1) - 2) ))-1
84     for clique in model._cliques_k_mas_1:
85         if sum([vy[u,v] for u in clique for v in clique if (u,v)
86         in model._E_prim ]) > (2*cota)+tol:
87             model.cbCut( quicksum(model._y[u,v] for u in clique
88             for v in clique if (u,v) in model._E_prim ) <= 2*cota)
89             model._res_lazy_sep_cliques =
90             model._res_lazy_sep_cliques + 1
91             contador2 = contador2 + 1
92             if contador2 >= maximo_por_solucion:
93                 break
94
95     del contador1 , contador2 , contador3 , contador4 , contador5
96
97
98     if where == GRB.Callback.MIPSOL:
99         vx = model.cbGetSolution(model._x)
100        vy = model.cbGetSolution(model._y)
101        for i,j in model._Grafofaltante.edges():
102            for k in model._K:
103                if vx[i,k]+vx[j,k] > 1.9:
104                    for sep in model._Gamma_nodos[(i,j)]:
105                        if vx[i,k]+vx[j,k] -
106                        sum([vx[h,k] for h in sep ]) > 1:
107                            model.cbLazy( model._x[i,k]+model._x[j,k]-
108                            quicksum(model._x[a,k] for a in sep) <=1 );
109                            model._res_lazy_sep = model._res_lazy_sep + 1

```

Código 6.12: Función callback para el modelo $FG-3$.

```

1 def callback_G3(model, where):
2     if where == GRB.Callback.MIPNODE:
3         vy = model.cbGetNodeRel(model._y)
4         vx = model.cbGetNodeRel(model._x)
5         maximo_por_solucion = 5
6         contador1 = 0

```

```

7     contador2 = 0
8     contador3 = 0
9     contador4 = 0
10    contador5 = 0
11    tol = 0.1
12
13    # Tipo Corte
14    if math.floor(model._n/len(model._K)) >= 3:
15        for a in range(1,model._n):
16            for b in range(1,model._n):
17                if a!=b:
18                    if sum([vy[u,v] for u,v in model._E_prim
19                        if u in [a,b] and v not in [a,b] ])<1-tol:
20                        model.cbCut( (quicksum(model._y[u,v]
21                            for u,v in model._E_prim if u in [a,b]
22                                and v not in [a,b] ) >=1))
23                        model._res_lazy_G3_corte =
24                        model._res_lazy_G3_corte + 1
25                        contador5 = contador5 +1
26                        if contador5 >= maximo_por_solucion:
27                            break
28                if contador5 >= maximo_por_solucion:
29                    break
30
31    if math.floor(model._n/len(model._K)) >= 4:
32        for c in range(1,model._n):
33            if b!=c:
34                if sum([vy[u,v] for u,v in model._E_prim
35                    if u in [a,b,c]
36                    and v not in [a,b,c] ])<1-tol:
37                    model.cbCut( (quicksum(model._y[u,v]
38                        for u,v in model._E_prim
39                            if u in [a,b,c]
40                                and v not in [a,b,c] )>=1))
41                    model._res_lazy_G3_corte =
42                    model._res_lazy_G3_corte + 1
43                    contador5 = contador5 +1
44                    if contador5 >= maximo_por_solucion:
45                        break
46                if contador5 >= maximo_por_solucion:
47                    break
48    if contador5 >= maximo_por_solucion:
49        break
50

```

```

51 # Tipo Triángulo
52 if len(model._K)>2:
53     for v,w,u in model._triadas:
54         if (1-vy[v,w])-(1-vy[w,u])-(1-vy[v,u]) > 0+tol:
55             model.cbCut((1-model._y[v,w])-(1-model._y[w,u])-(
56                 (1-model._y[v,u]) <= 0)
57             model._res_lazy_G3_triangulos =
58             model._res_lazy_G3_triangulos + 1
59             contador1 = contador1 + 1
60             if contador1 >= maximo_por_solucion:
61                 break
62         if -(1-vy[v,w])+(1-vy[w,u])-(1-vy[v,u]) > 0+tol:
63             model.cbCut(-(1-model._y[v,w])+(1-model._y[w,u])-(
64                 (1-model._y[v,u]) <= 0)
65             model._res_lazy_G3_triangulos =
66             model._res_lazy_G3_triangulos + 1
67             contador1 = contador1 + 1
68             if contador1 >= maximo_por_solucion:
69                 break
70         if -(1-vy[v,w])-(1-vy[w,u])+(1-vy[v,u]) > 0+tol:
71             model.cbCut(-(1-model._y[v,w])-(1-model._y[w,u])+(
72                 (1-model._y[v,u]) <= 0)
73             model._res_lazy_G3_triangulos =
74             model._res_lazy_G3_triangulos + 1
75             contador1 = contador1 + 1
76             if contador1 >= maximo_por_solucion:
77                 break
78
79 #Tipo Clique
80 k1 = len(model._K)
81 cota = (factorial(k1+1) /
82         (factorial(2) * factorial((k1+1) - 2) ))-1
83 for clique in model._cliques_k_mas_1:
84     if sum([(1-vy[u,v]) for u in clique for v in clique
85         if (u,v) in model._E_prim ]) > (2*cota)+tol:
86         model.cbCut( quicksum((1-model._y[u,v]) for u in clique
87             for v in clique if (u,v) in model._E_prim ) <= 2*cota)
88         model._res_lazy_G3_cliques =
89         model._res_lazy_G3_cliques + 1
90         contador2 = contador2 + 1
91         if contador2 >= maximo_por_solucion:
92             break
93
94

```

```

95 # Tipo Separador
96 for i,j in model._Grafofaltante.edges():
97     for k in model._K:
98         if vx[i,k]+vx[j,k] >= 2+tol
99         and vx[i,k] > 0 and vx[j,k] >0:
100             for sep in model._Gamma_nodos[(i,j)]:
101                 if vx[i,k]+vx[j,k] -
102                 sum([vx[h,k] for h in sep ]) > 1+tol:
103                     model.cbCut( model._x[i,k]+model._x[j,k]-
104                     quicksum(model._x[a,k] for a in sep) <=1 );
105                     model._res_lazy_G3 = model._res_lazy_G3 + 1
106                     contador3 = contador3 + 1
107                     if contador3 >= maximo_por_solucion:
108                         break
109                 if contador3 >= maximo_por_solucion:
110                     break
111             if contador3 >= maximo_por_solucion:
112                 break
113
114 del contador1 , contador2 , contador3 , contador4 , contador5

```

Código 6.13: Formulación \mathcal{FG} -1.

```

1 res_lazy_flujo = 0
2 res_lazy_flujo_triangulos = 0
3 res_lazy_flujo_cliques = 0
4 res_lazy_flujo_cota_aristas = 0
5 res_lazy_flujo_caminos=0
6 res_lazy_flujo_corte=0
7 K = range(1,k+1)
8 M = n-k+1
9
10 mG_1 = Model()
11 x=mG_1.addVars(V,K, name="x", vtype=GRB.BINARY);
12 y=mG_1.addVars(E_prim, name="y", vtype=GRB.BINARY);
13 z=mG_1.addVars(V,K, name="z", vtype=GRB.BINARY);
14 f=mG_1.addVars(E_prim, name="f", vtype=GRB.CONTINUOUS, lb =0);
15
16 mG_1._x = x
17 mG_1._y = y
18 mG_1._n = n
19 mG_1._Grafofaltante = G_falt
20 mG_1._res_lazy_flujo = res_lazy_flujo
21 mG_1._res_lazy_flujo_cota_aristas = res_lazy_flujo_cota_aristas
22 mG_1._res_lazy_flujo_triangulos = res_lazy_flujo_triangulos

```

```

23 mG_1._res_lazy_flujo_cliques = res_lazy_flujo_cliques
24 mG_1._res_lazy_flujo_caminos = res_lazy_flujo_caminos
25 mG_1._res_lazy_flujo_corte = res_lazy_flujo_corte
26 mG_1._K = K
27 mG_1._m = m
28 #mG_1._Gamma_nodos = Gamma1
29 mG_1._E_prim = E_prim
30 mG_1._triadas = triadas
31 mG_1._cliques_k_mas_1 = cliques_k_mas_1
32 mG_1._caminos = caminos
33 mG_1._cortes_de_2 = cortes_de_2
34 mG_1._cortes_de_3 = cortes_de_3
35
36 mG_1.setObjective( ( quicksum( y[u,v]*d_prim[u,v]
37 for u,v in E_prim) ) , GRB.MAXIMIZE);
38 mG_1.addConstrs( ( x.sum(v, '*') == 1 for v in V ),
39 name="res_1b");
40 mG_1.addConstrs( ( x[u,i]-x[v,i] <= y[u,v]
41 for u,v in E_prim for i in K ), name="res_1c");
42 mG_1.addConstrs( ( x[v,i]-x[u,i] <= y[u,v]
43 for u,v in E_prim for i in K ), name="res_1d");
44 mG_1.addConstrs( ( x[u,i]+x[v,i]+y[u,v] <= 2
45 for u,v in E_prim for i in K ), name="res_1e");
46 mG_1.addConstrs( ( z.sum('*',i)==1 for i in K ),
47 name="res_2a");
48 mG_1.addConstrs( ( z[u,i] <= x[u,i] for u in V
49 for i in K ), name="res_2b");
50 mG_1.addConstrs( ( f[u,v]+f[v,u] <= M*(1-y[u,v])
51 for u,v in E_prim), name="res_2c");
52 mG_1.addConstrs( ( f.sum(u, '*')-f.sum('*',u) >=
53 1-M*z.sum(u, '*') for u in V ), name="res_2d");
54
55
56
57 if n % k == 0:
58     mG_1.addConstrs( ( x.sum('*',i) == (n/k)
59     for i in K), name="res_2e");
60 else:
61     mG_1.addConstrs( ( x.sum('*',i) <=
62     math.ceil(n/k) for i in K), name="res_3part1");
63     mG_1.addConstrs( ( x.sum('*',i) >=
64     math.floor(n/k) for i in K), name="res_3part2");
65
66 # Tipo Cota

```

```

67 max_aristas = (mG_1.m-mG_1.n+len(mG_1._K))*2
68 mG_1.addConstr( y.sum( '*' , '*' )<=
69 max_aristas ,name="cota" )
70 mG_1._res_lazy_flujo_cota_aristas =
71 mG_1._res_lazy_flujo_cota_aristas + 1
72
73 # Tipo Camino
74
75 for k in mG_1._K:
76     for i ,j in mG_1._caminos:
77         mG_1.addConstr( (x[i ,k]+x[j ,k] ) <=
78             1 ,name="camino" )
79         mG_1._res_lazy_flujo_caminos =
80         mG_1._res_lazy_flujo_caminos +1
81
82
83 mG_1.Params.LazyConstraints = 0
84 mG_1.Params.Cuts = 0
85 mG_1.Params.PreCrush = 1
86 mG_1.Params.TimeLimit = 3600
87 mG_1.update()
88 mG_1.optimize( callback_flujo )

```

Código 6.14: Formulación \mathcal{FG} -2.

```

1 res_lazy_sep = 0
2 res_lazy_sep_triangulos = 0
3 res_lazy_sep_cliques = 0
4 res_lazy_sep_cota_aristas = 0
5 res_lazy_sep_caminos=0
6 res_lazy_sep_corte=0
7 res_lazy_sep_cut=0
8
9 K = range(1,k+1)
10 mG_2 = Model()
11 x=mG_2.addVars(V,K, name="x", vtype=GRB.BINARY);
12 y=mG_2.addVars(E_prim, name="y", vtype=GRB.BINARY);
13
14
15 mG_2._x = x
16 mG_2._y = y
17 mG_2._n = n
18 mG_2._Grafofaltante = G_falt
19 mG_2._res_lazy_sep = res_lazy_sep
20 mG_2._res_lazy_sep_triangulos = res_lazy_sep_triangulos

```

```

21 mG_2._res_lazy_sep_cliques = res_lazy_sep_cliques
22 mG_2._res_lazy_sep_cota_aristas = res_lazy_sep_cota_aristas
23 mG_2._res_lazy_sep_caminos = res_lazy_sep_caminos
24 mG_2._res_lazy_sep_corte = res_lazy_sep_corte
25 mG_2._res_lazy_sep_cut = res_lazy_sep_cut
26 mG_2._K = K
27 mG_2._m = m
28 mG_2._E_prim = E_prim
29 mG_2._Gamma_nodos = Gammal
30 mG_2._triadas = triadas
31 mG_2._cliques_k_mas_1 = cliques_k_mas_1
32 mG_2._caminos = caminos
33
34
35
36 mG_2.setObjective( ( quicksum( y[u,v]*d_prim[u,v]
37 for u,v in E_prim) ) , GRB.MAXIMIZE);
38 mG_2.addConstrs( ( x.sum(v, '*') == 1 for v in V ),
39 name="res_1b");
40 mG_2.addConstrs( ( x[u,i]-x[v,i] <= y[u,v]
41 for u,v in E_prim for i in K ), name="res_1c");
42 mG_2.addConstrs( ( x[v,i]-x[u,i] <= y[u,v]
43 for u,v in E_prim for i in K ), name="res_1d");
44 mG_2.addConstrs( ( x[u,i]+x[v,i]+y[u,v] <=
45 2 for u,v in E_prim for i in K ), name="res_1e");
46
47 #for u,v in Gammal:
48 #    for S in Gammal[(u,v)]:
49 #        for i in K:
50 #            mG_2.addConstr( ( x[u,i]+x[v,i]-
51 quicksum( x[w,i] for w in S ) <=1 ) )
52 #            #mG_2.addConstr( ( x[u,i]+x[v,i]+
53 quicksum( y[w1,w2] for w1,w2 in S ) <=2+(len(S)-1) ) )
54
55
56
57 if n % k == 0:
58     mG_2.addConstrs( ( x.sum( '*',i) == (n/k)
59 for i in K), name="res_2e");
60 else:
61     mG_2.addConstrs( ( x.sum( '*',i) <=
62 math.ceil(n/k) for i in K), name="res_3part1");
63     mG_2.addConstrs( ( x.sum( '*',i) >=
64 math.floor(n/k) for i in K), name="res_3part2");

```

```

65
66 # Tipo Cota
67 max_aristas = (mG_2._m-mG_2._n+len(mG_2._K))*2
68 mG_2.addConstr( y.sum( '*' , '*' )<= max_aristas ,
69 name="cota" )
70
71 # Tipo Camino
72 for k in mG_2._K:
73     for i,j in mG_2._caminos:
74         mG_2.addConstr( (x[i,k]+x[j,k] ) <=
75             1,name="camino")
76
77
78 mG_2.Params.LazyConstraints = 1
79 mG_2.Params.PreCrush = 1
80 mG_2.Params.Cuts = 0
81 mG_2.params.TimeLimit = 3600
82 mG_2.update()
83 mG_2.optimize( callback_sep )

```

Código 6.15: Formulación \mathcal{FG} -3.

```

1 res_lazy_G3 = 0
2 res_lazy_G3_triangulos = 0
3 res_lazy_G3_cliques = 0
4 res_lazy_G3_cota_aristas = 0
5 res_lazy_G3_caminos=0
6 res_lazy_G3_corte=0
7
8 K = range(1,k+1)
9 M = n-k+1
10
11 mG_3 = Model()
12 x=mG_3.addVars(V,K, name="x", vtype=GRB.BINARY);
13 y=mG_3.addVars(E_prim, name="y", vtype=GRB.BINARY);
14 z=mG_3.addVars(V,K, name="z", vtype=GRB.BINARY);
15 f=mG_3.addVars(E_prim, name="f", vtype=GRB.CONTINUOUS, lb =0);
16 mG_3.update()
17 x.BranchPriority=4
18 y.BranchPriority=3
19 z.BranchPriority=2
20 f.BranchPriority=1
21 mG_3.update()
22
23

```



```

24 mG_3._x = x
25 mG_3._y = y
26 mG_3._n = n
27 mG_3._m = m
28 mG_3._Grafofaltante = G_falt
29 mG_3._res_lazy_G3 = res_lazy_G3
30 mG_3._res_lazy_G3_triangulos = res_lazy_G3_triangulos
31 mG_3._res_lazy_G3_cliques = res_lazy_G3_cliques
32 mG_3._res_lazy_G3_cota_aristas = res_lazy_G3_cota_aristas
33 mG_3._res_lazy_G3_caminos = res_lazy_G3_caminos
34 mG_3._res_lazy_G3_corte = res_lazy_G3_corte
35 mG_3._K = K
36 mG_3._E_prim = E_prim
37 #mG_3._Gamma_nodos = Gammal
38 #mG_3._triadas = triadas
39 mG_3._cliques_k_mas_1 = cliques_k_mas_1
40 mG_3._caminos = caminos
41
42
43 mG_3.setObjective( ( quicksum( y[u,v]*d_prim[u,v]
44 for u,v in E_prim) ) , GRB.MINIMIZE);
45 mG_3.addConstrs( ( x.sum(v, '*') == 1 for v in V ),
46 name="res_1b");
47
48 mG_3.addConstrs( ( x[u,i]+x[v,i]-y[u,v] <= 1
49 for u,v in E_prim for i in K ), name="res_1e");
50
51 mG_3.addConstrs( ( z.sum('*',i)==1 for i in K ),
52 name="res_2a");
53 mG_3.addConstrs( ( z[u,i] <= x[u,i] for u in V
54 for i in K ), name="res_2b");
55
56 mG_3.addConstrs( ( f[u,v]+f[v,u] <= M*y[u,v]
57 for u,v in E_prim), name="res_2c");
58 #mG_3.addConstrs( ( y[u,v]-y[v,u] == 0
59 for u,v in E_prim), name="res_adicional");
60
61 mG_3.addConstrs( ( x[u,i]+x[v,j]+y[u,v] <= 2
62 for u,v in G_prim.edges() for i in K
63 for j in K if i != j), name="res_2e");
64
65
66
67 if n % k == 0:

```

```

68     mG_3.addConstrs( ( f.sum(u, '*')-f.sum('* ',u) ==
69     (n/k)*z.sum(u, '*')-1 for u in V ), name="res_2d");
70 else:
71     mG_3.addConstrs( ( f.sum(u, '*')-f.sum('* ',u) ==
72     (math.ceil(n/k)*z.sum(u, range(1,(n %k)+1)))+
73     (math.floor(n/k)*z.sum(u, range((n %k)+1,k+1))) -
74     1 for u in V ), name="res_2d1");
75
76
77 # Tipo Cota
78 max_aristas = (mG_3._n-len(mG_3._K))*2
79 mG_3.addConstr( (y.sum('* ', '*') >= max_aristas)
80 ,name="cota")
81
82 # Tipo Camino
83 for k in mG_3._K:
84     for i, j in mG_3._caminos:
85         mG_3.addConstr( (x[i, k]+x[j, k] ) <= 1,
86         name="camino")
87
88
89 mG_3.Params.LazyConstraints = 0
90 mG_3.Params.PreCrush = 1
91 mG_3.Params.Cuts = 0
92 mG_3.params.TimeLimit = 3600
93 mG_3.update()
94 mG_3.optimize(callback_G3)

```