ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE CIENCIAS

PROGRAMACIÓN SEMIDEFINIDA PARA LA SOLUCIÓN DEL PROBLEMA DE PARTICIONAMIENTO DE GRAFOS K-WAY BALANCEADO CON RESTRICCIONES DE PESO

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO MATEMÁTICO

SANTIAGO ROMÁN FLORES BRAVO

santy_rflb@hotmail.com

DIRECTOR: DIEGO FERNANDO RECALDE CALAHORRANO

diego.recalde@epn.edu.ec

DMQ, SEPTIEMBRE 2022

CERTIFICACIONES

Yo, SANTIAGO ROMÁN FLORES BRAVO, declaro que el trabajo de integración curricular aquí descrito 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.

SANTIAGO ROMÁN FLORES BRAVO

Certifico que el presente trabajo de integración curricular fue desarrollado por SANTIAGO ROMÁN FLORES BRAVO, bajo mi supervisión.

DIEGO FERNANDO RECALDE CALAHORRANO

DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como los productos resultantes del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

SANTIAGO ROMÁN FLORES BRAVO

DIEGO FERNANDO RECALDE CALAHORRANO

Dedicatoria

A Román, Lucía, Gaby, Juan, Vero y Ruth.

Agradecimiento

Agradezco a mi familia, por el apoyo, la paciencia y el cariño que me han brindaron durante toda mi vida, en especial la ayuda y consejos que me dieron durante esta carrera.

A Gaby, quien logró darme el impulso final y la fuerza para completar este trabajo, su amor fue la clave para lograr encontrar la motivación de seguir.

A todos los profesores, compañeros y autoridades con los que tuve el privilegio de trabajar, me guiaron en lo académico y compartimos grandes momentos en todos los ámbitos de la vida universitaria.

A todos mis amigos, que me empujaron en los momentos de desasosiego, con total desinterés me ayudaron a continuar mis estudios y me acompañaron a completarlos.

A Diego, por su guía, su tiempo y paciencia, a pesar de mi intermitencia, me ayudó a terminar este trabajo.

Santiago

RESUMEN

La Programación Semidefinida es útil para resolver distintos problemas de optimización convexa; en este trabajo la empleamos para encontrar una solución al problema de particionamiento de grafos k-way balanceado con restricciones de peso. Se desarrolla dos formulaciones del problema propuesto, una de ellas iterativa, basadas en emplear una relajación lineal a la formulación semidefinida construida para el problema. Se simula varias instancias para la experimentación computacional, se presentan los resultados de la implementación y finalmente se presentan conclusiones y recomendaciones.

Palabras clave: Programación Semidefinida, Particionamiento de Grafos.

ABSTRACT

Semidefinite Programming is useful for solving different convex optimization problems; in this work we use it to find a solution to the weight-constrained balanced k-way graph partitioning problem. Two formulations for the problem are developed, one of them iterative, based on a linear relaxation of the semi-definite formulation built for the problem. Several instances are simulated for computational experimentation, the results of the implementation are presented, as well as conclusions and recommendations.

Keywords: Semidenite Programming, Graph Partitioning.

Índice general

Índice de figuras

Capítulo 1

Descripción del componente desarrollado

La Programación Semidefinida, SDP, por sus siglas en inglés (*Semidefinite Programing*), es un subcampo de la Optimización Convexa [18], que tiene por objeto optimizar (maximizar o minimizar) una función lineal sujeta a restricciones caracterizadas por una combinación afín de matrices simétricas semidefinidas positivas [16].

La teoría que cubre a la Programación Semidefinida puede abordarse desde distintas perspectivas, al ser parte de la Optimización Convexa puede estudiarse desde el Análisis Convexo, así mismo, puede estudiarse como una extensión de la Programación Lineal [18].

Dentro de la Optimización Combinatoria, la importancia del estudio de la Programación Semidefinida radica en su utilidad para resolver relajaciones convexas no lineales de problemas NP-difíciles, al igual que su uso para derivar límites para problemas de optimización cuadráticos no convexos [18].

Además de la Optimización Combinatoria, la Programación Semidefinida encuentra muchas aplicaciones [16]; por ejemplo: ingeniería financiera, química cuántica, diseño de redes, industria de telecomunicaciones [5], etc. Además, se usa en la teoría de control y en la optimización estructural [18].

Entre los problemas de optimización en los que podemos emplear la Programación Semidefinida para su resolución se destacan los problemas

de particionamiento de grafos [19][5], problemas de asignación cuadrática [20], problemas de cartera con restricciones de cardinalidad [17], etc.

Los problemas de particionamiento de grafos son problemas NP-difíciles y como se explica en [11] surgen del objetivo de agrupar en varios grupos a un conjunto de objetos caracterizados por sus atributos tal que cada grupo cumpla algunos requisitos, como cardinalidad, límites de peso, etc.

Los problemas de particionamiento de grafos tienen diversas utilidades, como son, el diseño de redes [5], la calendarización de torneos deportivos [10], la creación de rutas de vehículos y encuestadores [11].

En el presente trabajo se define un problema de particionamiento k-way balanceado con restricciones de peso tanto en los nodos como en las aristas, donde el objetivo es minimizar el peso total de las aristas internas de los conjuntos de partición.

En esta primera sección presentamos el objetivo general, los objetivos específicos, el alcance del trabajo y el marco teórico del mismo, en donde se expone un primer acercamiento al estado del arte de la Programación Semidefinida con un especial enfoque en la aplicación de ésta en la resolución de problemas de particionamiento de grafos.

En la Metodología, la segunda sección del trabajo, detallamos el proceso empleado en la modelización del problema propuesto, donde, como resultado obtenemos dos formulaciones, las cuales corresponden a dos relajaciones lineales de la formulación desarrollada al aplicar la Programación Semidefinida para resolver el problema de particionamiento de grafos planteado.

En la tercera sección, correspondiente a la experimentación computacional, se presentan detalles de las instancias simuladas, así como un resumen del desempeño de las formulaciones desarrolladas en el presente trabajo, así como una comparación entre ellas. Finalmente, en la cuarta sección se presentan conclusiones y recomendaciones.

1.1. Objetivo general

- Estudiar el problema de particionamiento de grafos k-way balanceado con restricciones de peso en nodos y aristas desde un enfoque basado en la Programación Semidefinida.

1.2. Objetivos específicos

OE1 Explicar de forma detallada la aplicación de la Programación Semidefinida en la solución del problema de particionamiento de grafos k-way balanceada con restricciones de peso tanto en los nodos como en las aristas.

OE2 Modelar el problema de particionamiento k-way balanceada con restricciones de peso en nodos y aristas como un modelo basado en la Programación Semidefinida.

OE3 Implementar el modelo planteado en el lenguaje de programación Python.

OE4 Generar instancias para la experimentación computacional y resolverlas.

1.3. Alcance

En el presente trabajo se estudia el estado del arte de la Programación Semidefinida, se describe su importancia, así como su utilidad para resolver distintos problemas de optimización, con énfasis en su uso para la resolución de problemas de particionamiento de grafos.

De la misma manera, se expone la importancia y utilidad de los problemas de particionamiento de grafos, en particular en el problema de particionamiento k-way balanceado con restricciones de peso en nodos y aristas.

Por medio de este trabajo abordamos el problema de particionamiento de grafos propuesto desde una óptica basada en la Programación Semide-

finida para la construcción del modelo del problema. Una vez modelado el problema construimos dos formulaciones basadas en una relajación lineal al modelo semidefinido construido.

Implementamos las formulaciones elaboradas con la ayuda del lenguaje de programación Python y para su estudio simulamos varias instancias para realizar pruebas computacionales. Para la simulación de estas instancias utilizamos el paquete de Python *igraph*, que nos permite la generación de grafos, mientras que para la resolución del problema empleamos a *Gurobi* como solver.

A partir de las instancias simuladas realizamos pruebas computacionales y evaluamos el modelo por medio de criterios, como por ejemplo, el tiempo de resolución exacta, el GAP alcanzado fijando un tiempo de cómputo dado. Comparamos los resultados computacionales, entre las dos formulaciones desarrolladas, mediante el valor de la función objetivo y el tiempo de resolución empleado en las distintas instancias. Por último, se presentan las conclusiones y recomendaciones elaboradas a partir del trabajo realizado.

1.4. Marco teórico

Para aproximarnos a la Programación Semidefinida iniciamos definiendo de manera general un problema de optimización. Tal como en [13] decimos que todos los problemas de optimización de variable real se puede formular como:

Encontrar el máximo o el mínimo de $f_0(x) : D \in \mathbb{R}^n \to \mathbb{R}$ (función objetivo) dentro de un subconjunto S definido en el dominio de la función, sujeto a una serie de restricciones, formuladas mediante ecuaciones e inecuaciones $\{f_i, h_i\}$.

$$\begin{aligned} &\begin{matrix}\text{mín}\\ \text{máx}\end{matrix} \quad f_0(x) \\ sujeto\, a: \\ &f_i(x) \leq 0\,, \quad i \in [1, m_1] \\ &f_i(x) \geq 0\,, \quad i \in (m_1, m] \\ &h_i(x) = 0\,, \quad i \in [1, l] \\ &x \in S \subseteq \mathbb{R}^n \end{aligned} \tag{1.1}$$

Notemos que todo problema de maximización puede reescribirse como un problema de minimización, dado que $\text{mín}\, f_0(x) = \text{máx} - f_0(x)$ y que $f_i(x) \leq 0$ se puede escribir como $-f_i(x) \geq 0$. Por otro lado, se puede sustituir las restricciones de igualdad con dos restricciones de desigualdad, entonces, sin perder generalidad se reescribe (1.1) como un problema de minimización:

$$\begin{aligned} &\text{mín} \quad f_0(x) \\ sujeto\, a: \\ &f_i(x) \leq 0\,, \quad i \in [1, m_1] \\ &x \in S \subseteq \mathbb{R}^n \end{aligned} \tag{1.2}$$

Así mismo, es bueno recordar que la convexidad es una propiedad ampliamente utilizada en matemática aplicada, más en concreto en la optimización [13]. Es así que escribimos varias definiciones asociadas a la convexidad.

Función Convexa:
Se dice que una función real de variable real $f(x) : \mathbb{R}^n \to \mathbb{R}$ es convexa si y solo si:

$$\forall x\,, y \in \mathbb{R}^n, \forall \alpha \in [0, 1]: \quad f(\alpha\, x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha) f(y) \tag{1.3}$$

El que una función sea convexa le provee de varias propiedades útiles;

en [13] se enuncian algunas de estás propiedades. Así mismo, la noción de convexidad es aplicada también en la teoría de conjuntos, donde:

Conjunto Convexo:
Se dice que un conjunto S es convexo si y solo si:

$$\forall x\,,\, y \in S, \forall \alpha \in [0,1]: \quad \alpha\, x + (1-\alpha) y \in S \tag{1.4}$$

Al igual que en el caso de las funciones convexas, los conjuntos convexos tienen propiedades importantes, que las podemos revisar en [13].

Ahora, es importante destacar que la Optimización Convexa es una de las ramas más desarrolladas hasta el momento dentro del campo de la optimización [13], por las propiedades que posee. Este caso especial de problema de optimización se lo puede definir como:

Problema de Optimización Convexa:
Un problema de optimización se dice que es convexo si y solo si:

1. La función objetivo $f_0(x)$ es una función convexa; y,
2. La región factible S es un conjunto convexo.

Notemos que para que S sea un conjunto convexo es necesario que las restricciones de igualdad sean funciones afines y las restricciones de desigualdad sean convexas.

De forma concreta, el problema de optimización convexa es el problema de encontrar algún punto $\tilde{x} \in S$ tal que este punto sea el $inf\{f_0(x) : x \in S\}$ donde f_0 y S son convexos [14].

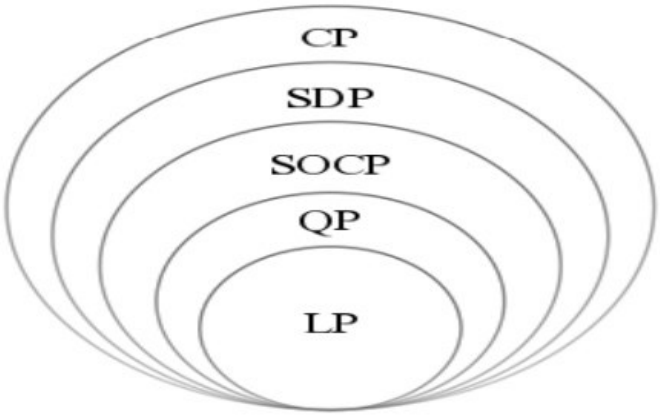
Los problemas de Optimización Convexa pueden clasificarse por las características que tomen tanto la función objetivo como las restricciones. Es decir, si todas estás son afines, o la función objetivo es cuadrática, etc.

Esto nos permite decir que la Optimización Convexa tiene varios subcampos de estudio como la Programación Lineal (LP) por sus siglas en inglés *Linear Program*, la Programación Cuadrática (QP) *Quadratic Program*, la Programación de Cono de Segundo Orden (SOCP) *Second-Order Cone Program*, la Programación Semidefinida (SDP) *Semidefinite Program* y la Programación de Cono (CP) *Cone Program*.

En [14] podemos ver una relación jerárquica entre este tipo de problemas, donde obtenemos que:

$$LP \subseteq QP \subseteq SOCP \subseteq SDP \subseteq CP \tag{1.5}$$

Figura 1.1: Jerarquía de problemas de optimización convexa



Ahora, tal como se expresa en [18] decimos que la Programación Semidefinida se refiere a los problemas de Optimización Convexa que se pueden expresar en la forma:

$$\begin{aligned} &\text{mín } C \bullet X \\ &sujeto\, a: \\ &\quad A_i \bullet X = b_i\,, \quad \forall\, i = 1, \ldots, m \\ &\quad X \succeq 0 \end{aligned} \tag{1.6}$$

donde $X \in S^n$ es la variable a optimizar, $b \in R^m$, $A_i \in S^n$ y $C \in S^n$ son los parámetros del problema y S^n representa al espacio de las matrices reales simétricas $n \times n$.

La notación $X \succeq 0$ significa que X es una matriz semidefinida positiva; recordemos que una matriz simétrica X se dice que es semidefinida positiva si solo si: $1) \forall y \neq 0 : y^T X y \geq 0 \quad y \quad 2) \forall i : \lambda_i(X) \geq 0$.

Mientras que la notación $C \bullet X$ representa el producto interno:

$$C \bullet X = \sum_{j=1}^{n} c_j x_j \tag{1.7}$$

Como se representa en las relaciones expuestas en (1.5) la Programación Semidefinida se puede considerar una extensión de la Programación Lineal [16], una generalización de la Programación Cuadrática y también puede verse como un caso especial de la Programación Cónica [14].

Según [3] la Programación Semidefinida es el desarrollo más emocionante en la programación matemática de la década de 1990, criterio que se comparte en [18], donde se dice que la Programación Semidefinida ha sido una de las áreas de investigación más emocionantes y activas en la optimización durante dicha década.

El interés en la Programación Semidefinida nace por sus aplicaciones en diversos campos de la matemática como la optimización con restricciones convexa tradicional [3], la Teoría de Control y la Optimización Combinatoria [18]. Por este interés han trabajado en ella expertos en programación convexa, álgebra lineal, optimización numérica, optimización combinatoria, teoría de control y estadística [18].

Considerando a la Programación Semidefinida como una extensión de la Programación Lineal, se puede ver una similitud en su teoría entre la Programación Lineal y la Programación Semidefinida, pero existen diferencias importantes. Los resultados de dualidad son más débiles y no existe un método Simplex directo o práctico para los programas semidefinidos [16], aunque existen extensiones [8] [4].

Así mismo, el estudio de la Programación Semidefinida ha permitido desarrollar algoritmos de punto interior eficientes para resolver problemas SDP [18], donde la mayoría de estos generalmente se pueden resolver de manera muy eficiente en la práctica y en la teoría [3].

También se puede aplicar el método del elipsoide de Yudin, Nemirovsky y Shor, o los métodos generales para la optimización convexa no diferenciable desarrollados por Shor, Kiwiel e Hiriart-Urruty y Lemaréchal [16].

Es importante remarcar que pese a que los programas semidefinidos son mucho más generales que los programas lineales, no son mucho más

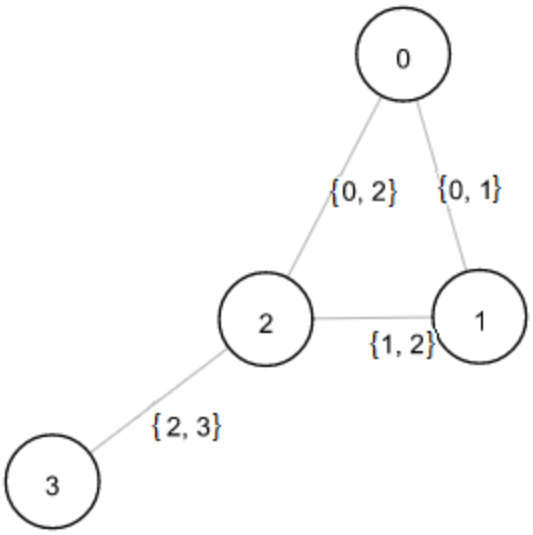
difíciles de resolver y la mayoría de los métodos de punto interior para Programación Lineal se han generalizado a programas semidefinidos [16].

Por otro lado, en la literatura encontramos que el estudio de grafos se remonta a 1736 con Euler y el conocido problema de los puentes de Königsberg; en [6] podemos encontrar un breve resumen de la teoría de grafos, donde se destaca que esta rama de las matemáticas, que parte de las matemáticas discretas, ha evolucionado constantemente para dar soluciones a problemas concretos.

Se define un Grafo G como la dupla $G := (V\ ,\ E)$ donde el conjunto de nodos es $V = \{1, \ldots, n\}$ y $E = \{\{i, j\} : i, j \in V, i \neq j\}$ representa al conjunto de las aristas, donde $\{ij\} \in E$ indica que existe un camino adyacente entre el nodo i y el nodo j.

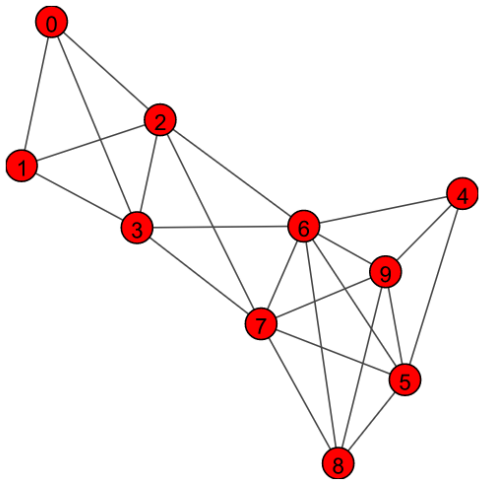
Gráficamente un grafo puede verse como:

Figura 1.2: Grafo



Existen distintos tipos de grafos, entre ellos podemos destacar a los árboles, los grafos conexos, grafos completos, grafos regulares, grafos dirigidos, grafos no dirigidos, etc. Para nuestro trabajo se dice que un grafo es no dirigido cuando sus aristas no tienen asociada ninguna dirección.

Figura 1.3: Grafo no dirigido

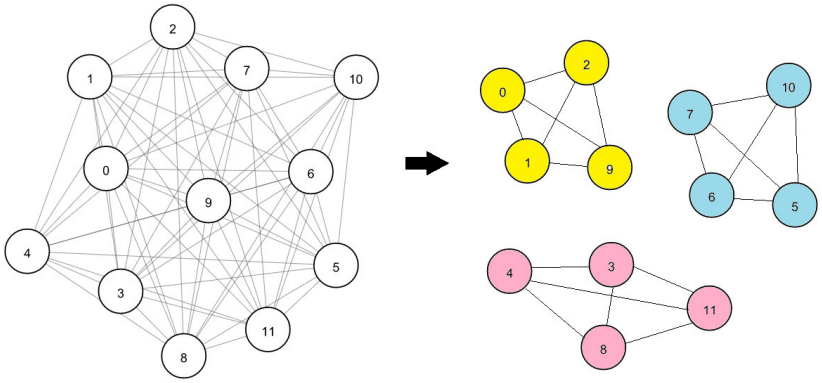


Como se muestra en [1], podemos ver que las aplicaciones de la teoría de grafos van desde el internet y temas científico-técnicos hasta los estudios sociales; inclusive, esta teoría puede ser importante en la educación a través de muchos juegos basados en los grafos.

Dentro de la teoría de grafos existen, entre otros, problemas clásicos como el problema de caminos más cortos, el problema de flujo de costo mínimo, el problema de particionamiento de grafos, etc. En nuestro trabajo nos centramos en el problema de particionamiento de grafos, el cual consiste en dividir el conjunto de nodos V en k conjuntos disjuntos, de tal forma que el conjunto $\{V_1, \ldots, V_k\}$ sea una *k-partición* del conjunto de nodos V [10], donde $\cup_{c=1}^{k} V_c = V$, $V_i \cap V_j = \emptyset$ y $V_c \neq \emptyset$ para todo $c \in [k]$. Donde $[k]$ representa el conjunto $\{1, \ldots, k\}$ con $k \geq 2$ un número entero fijo.

Estos conjuntos disjuntos forman k subgrafos $\{(V_1, E(V_1)) \ldots (V_k, E(V_k))\}$ de G que cumplen distintas características dependiendo del tipo de problema de particionamiento que se esté resolviendo.

Figura 1.4: Particionamiento de un grafo



Uno de los requisitos más usados es cumplir con restricciones de cardinalidad de los conjuntos V_c; notemos a la cardinalidad de los conjuntos V_c como $|V_c| = o_c$, donde:

$$0 \leq o_c \leq \left\lceil \frac{n}{k} \right\rceil, \quad \forall\, c \in [k]. \tag{1.8}$$

El problema de dividir los nodos de un grafo es interesante tanto por sus aplicaciones prácticas en diferentes industrias, así como por ser un campo en el cual se pueden diseñar nuevos algoritmos y heurísticas para su resolución, como sucede en [9]. Este tipo de problemas es ampliamente utilizado en la industria de las telecomunicaciones, donde se puede aplicar para subdividir una red de transmisión en grupos [5], en el diseño de calendarios en torneos deportivos [10], o en la resolución de problemas de flujo óptimo de potencia [2].

Para el estudio de nuestro problema de particionamiento de grafos, k-way balanceado con restricciones de peso en nodos y aristas, asumimos que todos los conjuntos V_c tienen la misma cardinalidad, es decir,

$$|V_c| = o =: \frac{n}{k}, \quad \forall\, c \in [k] \tag{1.9}$$

Así mismo asumimos que tanto los nodos $i \in V$ como las aristas $\{ij\} \in E$ tienen pesos, por lo que definimos las funciones de peso asociadas, como sigue:

$$w : V \longrightarrow \mathbb{R}^+ \tag{1.10}$$

$$d : E \longrightarrow \mathbb{R}^+ \tag{1.11}$$

es decir, d_{ij} es el peso de la arista $\{ij\} \in E$ y w_i representa el peso del nodo $i \in V$.

El problema planteado en el presente trabajo busca resolver este problema de particionamiento de grafos balanceado con pesos en los nodos de tal forma que se minimice el peso total de las aristas internas de cada conjunto de partición. Es destacable que este tipo de problema puede aplicarse para la resolución de instancias reales como la calendarización de campeonatos deportivos [10].

En [11] podemos encontrar un breve resumen de la literatura que encontramos referente al problema de particionamiento de grafos, donde se destaca que el estudio de estos problemas comenzó en los años sesenta con trabajos fundamentales como los de Carlson y Nemhauser, y de Kernighan y Lin.

El problema de particionamiento de grafos es NP-Difícil [10] y para su resolución se han estudiado distintos métodos; en la Programación Entera se destaca, entre otros, lo trabajado por Grötschel and Wakabayashi [11], quienes desarrollan un método exacto basado en la Programación Entera. Otra forma de resolución es emplear heurísticas, como la que desarrollan en [9]. Así mismo, se emplean métodos basados en la Programación Semidefinida para su resolución; en la literatura podemos destacar lo trabajado por Wolkowicz and Zhao [19] donde se emplea un método de punto interior primal-dual y en [5] donde se emplea una relajación lineal de una formulación semidefinida.

Capítulo 2

Metodología

Para la formulación del problema de particionamiento k-way balanceado con restricciones de peso tanto en los nodos como en las aristas, consideramos lo siguiente:

Sean S^n el conjunto de matrices simétricas de $n \times n$ y S^n_+ el conjunto de matrices simétricas semidefinidas positivas $n \times n$, de [3] se desprende que: $S^n_+ = \{X \in S^n | X \succeq 0\}$ es un Cono Convexo Cerrado en $\mathbb{R}^n$ de dimensión $n \times (n+1)/2$.

Se dice que un conjunto K es un Cono Convexo Cerrado si satisface que:

1. Si $x, w \in K$, entonces $\alpha x + \beta w \in K$ para todos los escalares no negativos α y β.

2. K es un conjunto cerrado.

Sea un grafo no dirigido $G = (V, E)$ con pesos en los nodos y en las aristas, w_i y d_{ij} respectivamente, donde $w_i \geq 0$ para todo $i \in V$ y $d_{ij} \geq 0$ para todo $\{ij\} \in E$, además asumimos que $d_{ij} = d_{ji}$, y sea A la matriz de adyacencia de G.

Queremos agrupar los nodos del grafo G en k conjuntos disjuntos V_k de tal forma que cumplan que $\cup_{c=1}^{k} V_c = V$, $V_i \cap V_j = \emptyset$ y $V_c \neq \emptyset$ para todo $c \in [k]$. Es decir, buscamos la *k-partición*, $\{V_1, \ldots, V_k\}$ del conjunto de nodos V.

Para modelar nuestro problema, al igual que en [5], definimos la variable binaria x_{ic} de la forma:

$$x_{ic} = \begin{cases} 1 \text{ si } i \in V_c \ , \\ 0 \text{ si no.} \end{cases} \tag{2.1}$$

es decir $x_{ic} = 1$ si el nodo i pertenece al conjunto V_c, $c \in [k]$.

Ahora, notemos que la expresión

$$x_{ic}(1 - x_{jc}) \tag{2.2}$$

es igual a 1 si el nodo i y el nodo j están contenidos en conjuntos de partición distintos, de lo contrario esta expresión toma el valor de 0.

Luego, a partir de esta expresión podemos calcular el peso total de las aristas que unen los diferentes conjuntos de la partición w_{cut}:

$$w_{cut} := \sum_{\{ij\} \in E} \sum_{c \in [k]} d_{ij} x_{ic}(1 - x_{jc}). \tag{2.3}$$

Nuestro problema consiste en maximizar el peso total de las aristas que unen los diferentes conjuntos de la partición w_{cut}; con esto minimizamos el peso total de las aristas internas de cada conjunto de partición.

$$\text{máx} \sum_{\{ij\} \in E} \sum_{c \in [k]} d_{ij} x_{ic}(1 - x_{jc}) \tag{2.4}$$

Para definir las restricciones de nuestro problema, al igual que en [11], definimos un vector τ-dimensional $(w_i^1, ..., w_i^\tau) \geq 0$ de pesos asociado con cada nodo i, establecemos los vectores de límite de peso inferior y superior $W_L, W_U \in \mathbb{R}^k$, con $W_L \leq W_U$, y definimos $\mathring{w}^t(S) = \sum_{i \in S} w_i^t$, para $S \subset V$.

Entonces nuestro problema debe satisfacer las restricciones de peso

$$W_L^{ct} \leq \mathring{w}^t(V_c) \leq W_U^{ct} \ , \quad \forall \, c \in [k] \quad \forall \, t \in [\tau] \tag{2.5}$$

Cada $t \in [\tau]$ corresponde a un atributo y para nuestro caso establece-

mos que $\tau = 2$ y que el primer atributo esta asociado a la cardinalidad y el segundo es un peso positivo definido para cada conjunto de la partición.

Respecto a la cardinalidad de los conjuntos, para limitar el tamaño de cada conjunto establecemos como se dijo anteriormente en (1.9) que todos los conjuntos V_C tienen la misma cardinalidad. Para esto, nuestro modelo debe satisfacer la restricción **R1**, que tiene el objeto de garantizar que la cardinalidad de todos los conjuntos de partición sea la misma:

$$\textbf{R1:} \quad \sum_i x_{ic} = o, \quad \forall\, c \in [k] \tag{2.6}$$

Por otro lado, establecemos la restricción **R2**, para asegurarnos que el nodo i esté solamente en un único conjunto de partición.

$$\textbf{R2:} \quad \sum_c x_{ic} = 1, \quad \forall\, i \in V \tag{2.7}$$

Ahora, incluimos la siguiente restricción **R3**, que impone el requisito de peso de los conjuntos de partición; para nuestro problema establecemos que en cada conjunto de partición V_c el valor de los límites de peso inferior y superior, W_L^c y W_U^c, respectivamente, serán los mismos. Luego:

$$\textbf{R3:} \quad W_L \leq \sum_i w_i\, x_{ic} \leq W_U\,, \quad \forall\, c \in [k] \tag{2.8}$$

En resumen nuestro problema consiste en resolver:

$$\text{máx} \sum_{\{ij\}\in E} \sum_{c\in[k]} d_{ij} x_{ic}(1 - x_{jc}) \tag{2.9}$$

$$\begin{aligned}
sujeto\, a:& \\
\sum_i x_{ic} &= o\,, && \forall\ c \in [k] \\
\sum_c x_{ic} &= 1\,, && \forall\, i \in V \\
W_L^c \leq \sum_i w_i\, x_{ic} &\leq W_U^c\,, && \forall\, c \in [k] \\
x_{ic} &\in \{0,1\}\,, && \forall\, i \in V\, \forall\, c \in [k]
\end{aligned}$$

Ahora, agrupamos nuestras variables binarias en la matriz $X = (x_{ic})$ con $X \in \mathbb{R}^{n\times k}$, con el objeto de escribir esta primera formulación (2.9) de forma matricial. De igual manera definimos la matriz $D \in \mathbb{R}^{n\times n}$ donde las entradas d_{ij} corresponden a la función de pesos de las aristas.

Notemos que la j-ésima columna de la matriz X es el vector indicador del conjunto V_j, $j \in [k]$.

Para la función objetivo dada por (2.4) notemos que w_{cut} puede expresarse como:

$$\sum_{\{ij\}\in E} \sum_{c\in[k]} d_{ij} x_{ic}(1 - x_{jc}) = \frac{1}{2}\, tr(X^T L X) \tag{2.10}$$

donde la función $tr()$ es la *traza de la matriz* y la matriz L es el *Laplaciano* asociado a la matriz D. Por definición $L \in \mathbb{R}^{n\times n}$ es la matriz tal que:

$$l_{ij} = \begin{cases} \sum_j d_{ij}, & \textbf{si}\ i = j, \\ -\, d_{ij} & \textbf{si}\ i \neq j, \\ 0 & \textbf{si}\ \{ij\} \notin E. \end{cases} \tag{2.11}$$

Por otro lado, notemos que $x_{ij}^T D x_{ij}$ es igual al doble del peso total de las aristas que pertenecen al j-ésimo conjunto de la partición V_j , $j \in [k]$; por

lo tanto $tr(X^T DX) = 2\, w_{uncut}$, donde w_{uncut} es el peso de todas las aristas no cortadas por la partición definida por X.

Notemos por e al vector de unos, es decir $e^T = (1, \ldots, 1)^T$; luego las restricciones **R1** y **R2** pueden escribirse como sigue:

$$\textbf{R1:} \quad (X^T e)_c = o, \quad \forall\, c \in [k] \tag{2.12}$$

$$\textbf{R2:} \quad (X\, e)_i = 1, \quad \forall\, i \in V \tag{2.13}$$

Mientras que para expresar de forma matricial a **R3** definimos el vector $W^T := (w_1, \ldots, w_n)^T$. Luego,

$$\textbf{R3:} \quad W_L^c \leq ((XW)^T e)_c \leq W_U^c, \quad \forall\, c \in [k] \tag{2.14}$$

De (2.10), (2.12), (2.13) y (2.14), la formulación del problema puede escribirse de forma matricial de la siguiente forma:

$$\begin{aligned} & \text{máx } \frac{1}{2} tr(X^T L X) \\ sujeto\, a: & \\ & X^T e = o\, e \\ & X e = e \\ W_L \leq & (XW)^T e \leq W_U \\ & X \quad \text{binaria de } n \times k. \end{aligned} \tag{2.15}$$

Esta formulación matricial (2.15) es un problema cuadrático de variables binarias y a partir de esta formulación construiremos una relajación semidefinida. Para esto, linealizamos la función objetivo, obteniendo una nueva variable $Y = XX^T$ y la sustituimos en (2.10).

Además, notemos que $tr(X^T L X) = tr(L(XX^T))$; luego, nuestra función objetivo puede escribirse como:

$$\frac{1}{2}\, tr(LY) \tag{2.16}$$

En este caso, la matriz $Y \in R^{n\times n}$ es una matriz semidefinida positiva ($Y \succeq 0$), es decir, $Y \in S^n_+$. Además, $y_{ij} = (X^T X)_{ij}$, es decir y_{ij} es la variable binaria tal que:

$$y_{ij} = \begin{cases} 1 \text{ si los nodos } i \text{ y } j \text{ están en el mismo conjunto,} \\ 0 \text{ en caso contrario.} \end{cases} \tag{2.17}$$

Este cambio de variable nos conduce a estudiar el conjunto:

$$F := Conv\{XX^T : X \text{ es matriz de partición}\} \tag{2.18}$$

Notemos que con esta linealización reemplazamos una suma de términos cuadráticos ($\sum_r x_{ir}x_{rj}$) por una variable binaria y_{ij}. Es así que aumentamos de $(n \times k)$ a $(n \times n)$ el número de variables utilizadas.

Así mismo, de [18] podemos ver que la matriz $Y \in F$ cumple con las siguientes propiedades:

(P1)

$$Y\, e = o\, e\,, \tag{2.19}$$

que se desprende de las restricciones **R1** ($X^T\, e_n = o\, e_k$) y **R2** ($Y\, e_k = e_n$), y

(P2)

$$diag(Y) = e\,, \tag{2.20}$$

pues cada fila de la matriz Y tiene exactamente una entrada igual a 1. La función $diag(\cdot)$ la definimos como:

$$\begin{aligned} diag(A) : \mathbb{R}^{n\times n} &\to \mathbb{R}^{n\times 1} \\ A &\to (a_{11}, \ldots, a_{ii}, \ldots, a_{nn})^T, \end{aligned}$$

es decir, $diag(Y)$ es el vector que contiene los elementos de la diagonal de la matriz Y.

(P3)

$$Y \in S_+^n, \tag{2.21}$$

se desprende de que $XX^T \in S_+^n$.

(P4)

$$o\,I - Y \succeq 0\,, \tag{2.22}$$

porque sabemos que $X^TX = o\,I$ y además se tiene que XX^TT posee los mismos valores propios (distintos de cero) que X^TX.

(P5)

$$Y \geq 0\,, \tag{2.23}$$

porque $X \geq 0$.

Ahora, tenemos de [5] que el paso complicado de esta relajación es el describir el casco convexo de Y, que se define como:

$$Conv\left\{ XX^T : \begin{array}{c} X \text{ es binaria}, \\ X^Te = o\,e, \\ Xe = e. \end{array} \right\} \tag{2.24}$$

donde X es matriz de partición, es decir $XX^T \in F$. Una descripción parcial de este conjunto se construye en [12], donde se establece como Lema que $Y \in F \iff Y \succeq 0, diag(Y) = e$ y $rank(Y) = 1$. Luego, la relajación semidefinida básica se obtiene eliminando la condición $rank(Y) = 1$ y optimizando los conjuntos restantes.

Por otro lado, emplearemos las desigualdades triangulares clásicas, en lugar de las desigualdades hipermétricas. Como se expresa en [12], utilizar las desigualdades hipermétricas presenta grandes desafíos y solamente una mejora marginal. Por lo tanto, definimos al poliedro MET como:

$$MET = \{Y = (y_{ij}) : \quad y_{ij} + y_{ik} \leq 1 + y_{jk} \quad \forall\, i, j, k\} \tag{2.25}$$

Las desigualdades triangulares agrupadas en MET establecen que si el nodo i y el nodo j están en el mismo conjunto V_c, y si el nodo i y el nodo k están en el mismo conjunto V_c, entonces también los nodos j y k deben estar en el mismo conjunto V_c.

Finalmente, definimos a la matriz J como $J = ee^T$, es decir J es la matriz cuadrada de unos:

$$J = \begin{pmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{pmatrix}$$

Así, las expresiones que describen parcialmente el casco convexo de Y, y que se desprenden de las propiedades son:

$$diag(Y) = e; \quad Ye = oe; \quad 0 \leq Y \leq J; \quad Y \succeq 0; \quad Y \in MET$$

Es prudente destacar que como se expresa en [18], dependiendo del subconjunto de restricciones seleccionadas se derivan la mayoría de las relajaciones comúnmente utilizadas en este tipo de problemas.

Por otro lado, **R3** podemos escribirla de forma similar a la formulación de [9]; así tenemos que:

$$W_L \leq w_i + \sum_{j:\{i,j\}\in E} w_j y_{ij} \leq W_U\,, \quad \forall i \in V. \tag{2.26}$$

La relajación que obtenemos es un programa semidefinido con un polinomio de igualdad lineal $(2n)$ y restricciones de desigualdad $O(n^3)$, es decir, en principio se lo puede resolver en tiempo polinomial [5].

$$\text{máx}\,\frac{1}{2}tr(LY) \tag{2.27}$$

$$\begin{aligned}
&sujeto\,a: \\
&\qquad diag(Y) = e\,, && (2.27.1)\\
&\qquad Ye = oe\,, && (2.27.2)\\
&\qquad 0 \leq Y \leq J\,, && (2.27.3)\\
&\qquad Y \succeq 0\,, && (2.27.4)\\
&\qquad Y \in MET\,, && (2.27.5)\\
&W_L \leq w_i + \sum_{j:\{i,j\}\in E} w_j y_{ij} \leq W_U \;\forall i \in V\,, && (2.27.6)\\
&\qquad y_{ij} \in \{0,1\}\,, \quad \forall\{i,j\} \in E. && (2.27.7)
\end{aligned}$$

Está no es la única formulación que se puede construir para este tipo de problemas, en [18] se nos presentan dos límites para este tipo de problemas w_{DH} y w_{RW}.

El límite propuesto por Donath y Hoffman, w_{DH}, parte de la caracterización de los valores propios de la matriz de adyacencia del grafo y empleando la dualidad de los problemas semidefinidos, se puede escribir este límite como un problema de la forma:

$$\text{mín}\,\frac{1}{2}\,L \bullet Y \quad s.a \quad oI \succeq Y \succeq 0, \quad diag(Y) = e$$

Mientras que Rendl y Wolkowicz con el objeto de mejorar a w_{DH} empleando nuevamente la dualidad proponen el límite w_{RW} que se puede escribir como un problema de la forma:

$$\text{mín}\,\frac{1}{2}\,L \bullet Y \quad s.a \quad diag(Y) = e, \quad Ye = oe \quad oI \succeq Y \succeq 0$$

Sin embargo, como se puede ver en [18] por el teorema de Perron-Frobenius, como $Y \geq 0$, $Ye = oe$, implica que Y no tiene un valor propio mayor que o, se concluye que está relajación domina a los límites w_{DH} y w_{RW}.

Notemos que a diferencia de los límites w_{DH} y w_{RW} nuestra formulación

tiene una sola restricción semidefinida (2.27.4). Por otro lado, en [18] se afirma que esta formulación es una buena aproximación para este tipo de problemas, incluso para valores grandes de k.

Esta formulación es un problema de programación semidefinida y como tal tiene varias formas de resolución, como en [19] donde plantean la solución por medio del método de punto interior primal-dual, en cambio en [15] se plantean varias relajaciones.

2.1. Primera relajación

En [5] proponen dos relajaciones de (2.27): una basada en un enfoque lineal y otra en un enfoque semidefinido positivo; en este trabajo analizaremos la primera opción. Así, relajaremos la formulación dada en (2.27) bajo un enfoque basado en la Programación Lineal, a través de la eliminación de la restricción (2.27.4) obteniendo así la siguiente formulación.

Primera Relajación ($\mathcal{F}_1$):

$$\begin{aligned} & \text{máx}\, \frac{1}{2} tr(LY) \qquad (2.28) \\ sujeto\, a: . & \\ & diag(Y) = e\,, \\ & Ye = oe\,, \\ & 0 \leq Y \leq J\,, \\ & Y \in MET\,, \\ W_L \leq w_i + & \sum_{j:\{i,j\}\in E} w_j y_{ij} \leq W_U\,, \quad \forall i \in V\,, \\ & y_{ij} \in \{0,1\}\,, \quad \forall \{i,j\} \in E. \end{aligned}$$

Este enfoque lineal lo obtenemos de eliminar la restricción $Y \succeq 0$ mientras que el enfoque semidefinido consiste en la eliminación de las restricciones (2.27.3) y (2.27.5), con lo que se obtiene un programa semidefinido con $2n$ restricciones de igualdad.

Según [5] el mantener explícitamente las restricciones ($Y \in MET$) no

es práctico incluso para valores pequeños de n al representar en total $3\binom{n}{3}$ restricciones, es así que propone una relajación que permita una solución en forma iterativa.

2.2. Segunda relajación

La segunda relajación se consigue eliminando el conjunto de restricciones $Y \in MET$ de la Formulación 1, obteniendo así una formulación simplificada de la misma que viene dada por:

Segunda relajación ($\mathcal{F}_2$)

$$\begin{aligned} & \text{máx } \frac{1}{2} tr(LY) \\ sujeto.\, a. & \\ & diag(Y) = e\,, \\ & Ye = oe\,, \\ & 0 \leq Y \leq J\,, \\ W_L \leq w_i + & \sum_{j:\{i,j\}\in E} w_j y_{ij} \leq W_U\,, \quad \forall i \in V\,, \\ & y_{ij} \in \{0,1\}\,, \quad \forall\, \{i,j\} \in E. \end{aligned} \tag{2.29}$$

Es importante mencionar que esta relajación puramente lineal del problema (2.29), usando solo $diag(Y) = e$, $Ye = oe$, $Y \geq 0$ da resultados sorprendentemente buenos para problemas donde k es grande, digamos $k \approx \sqrt{n}$ o mayor [18].

Aún así, es fácil ver que una solución de (2.29) viola varias desigualdades triangulares del poliedro MET por lo tanto, al usar esta relajación, iterativamente agregaremos las desigualdades violadas de la siguiente manera.

1. Resolvemos la versión simplificada del problema dada por (2.29).
2. Agregamos las restricciones de MET violadas a (2.29).

3. Resolvemos y eliminamos las desigualdades triangulares inactivas.

4. Repetimos desde **1** hasta que no se encuentren, significativamente, desigualdades violadas o se alcance el número máximo de desigualdades incluidas en la formulación.

Basados en [5] definimos que el número máximo de desigualdades agregadas será de $5n$.

En [18] podemos observar que una relajación basada en la programación cuadrática es posible, sin embargo es demasiado débil. Esta relajación se basa en optimizar w_{cut} sobre el casco convexo de las matrices de partición X, es decir, sobre $\sum_j x_{ij} = 1, \sum_i x_{ij} = o, x \geq O$.

La relajación cuadrática es posible porque, si $A \geq 0$, entonces $L \succeq 0$, y se tiene que $tr(X^T LX)$ es convexa y su débilidad se desprende de que $\hat{x}_{ij} = \frac{1}{k}$ el promedio de todas las matrices de partición da $tr(\hat{X}^T L\hat{X}) = 0$

Capítulo 3

Experimentación Computacional

Para la experimentación computacional creamos varias instancias, para ello utilizamos el paquete *igraph* del lenguaje de programación Python, el código empleado se puede revisar en el Anexo 1.

En total creamos 15 instancias que son presentadas en el Cuadro 3.1, la organización de esta tabla es la siguiente: la primera columna muestra el valor del número de nodos n, la segunda columna la cantidad de conjuntos k y la tercera el número de aristas m.

n	**k**	**m**
4	2	5
6	2	15
6	6	15
10	2	33
10	5	33
15	3	76
15	5	76
20	4	135
20	5	135
25	5	193
30	6	282
40	8	532
50	5	859
50	25	859
100	20	3181

Cuadro 3.1: Instancias generadas

Notemos que en todos los casos, las instancias simuladas cumplen que $n \, mod \, k = 0$, en las aplicaciones en el mundo real la mayoría de veces esto no se cumple, en esos casos se puede introducir vértices artificiales para que n se convierta en un múltiplo de k y se introducen restricciones que garanticen que los vértices artificiales sean distribuidas en los distintos conjuntos.

Estas instancias se resolvieron utilizando el solucionador de programación de números enteros Gurobi Optimize 9.5 [7] y el lenguaje de programación Python como interfaz. Todos los experimentos se realizaron en un Intel Core i7 2,30 GHz con 8 GB de RAM con Windows 10 Home. No se limitó el tiempo de cálculo, pero luego de 12 horas (43200 segundos) se frenó el experimento en las instancias más grande.

3.1. Comportamiento de la primera relajación

En el Cuadro.3.2 presentamos el comportamiento de la primera relajación $\mathcal{F}_1$ al resolver las 15 instancias simuladas, en la primera columna se muestra la instancia simulada (n, k) donde n es el número de nodos y k la cantidad de conjuntos, la segunda columna Z el valor de la función objetivo, la tercera columna es el *GAP* y la cuarta es el tiempo t en segundos.

Las instancias con $n < 20$ así como la instancia $(50, 25)$ se resuelven en menos de un segundo, mientras que en contraste es importante remarcar que en las instancias $(50, 5)$ y $(100, 20)$ no se encontró una solución después de 12 horas de ejecución, en cambio la instancia $(40, 8)$ pese a demorarse más de 1 hora y 20 minutos se encontró una solución, así mismo las cuatro instancias con $20 \leq n \leq 30$ se resuelven en un tiempo razonable.

En [18] se expresa que este tipo de relajaciones dan resultados buenos en problemas donde k es grande, es decir $k \approx \sqrt{n}$ o mayor. Lo que es comprobable principalmente en la instancia $(50, 25)$.

(n,k)	Z	GAP	Tiempo
(4,2)	16	0,00 %	0,035
(6,2)	57	0,00 %	0,056
(6,3)	70	0,00 %	0,027
(10,2)	119	0,14 %	0,081
(10,5)	171	0,00 %	0,039
(15,3)	329	0,00 %	0,283
(15,5)	374	0,45 %	0,153
(20,4)	644	0,00 %	2,258
(20,5)	671	0,34 %	2,433
(25,5)	946	0,32 %	40,609
(30,6)	1423	0,22 %	84,342
(40,8)	2623	0,06 %	5013,147
(50,5)	3912	2,29 %	42942,000
(50,25)	4360	0,00 %	0,437
(100,20)	15906	0,56 %	43132,000

Cuadro 3.2: Resumen comportamiento de $\mathcal{F}_1$

3.2. Comportamiento de la segunda relajación

En el Cuadro.3.3 presentamos el comportamiento de la relajación iterativa $\mathcal{F}_2$ al resolver las instancias simuladas, la organización de esta tabla de la segunda formulación, es la siguiente: en la primera columna se muestra la instancia simulada (n, k) donde n es el número de nodos y k la cantidad de conjuntos, en la segunda están la cantidad de iteraciones, en la tercera columna tenemos el valor de la función objetivo Z, la cuarta columna es el *GAP* y en la quinta columna está el tiempo t en segundos.

Todas las instancias se resuelven en menos de $1,288$ segundos, en un promedio de $2,6$ iteraciones, donde la instancia $(10, 5)$ resalta con 7 iteraciones y la instancia $(10, 2)$ tiene un valor de GAP de $0,53\,\%$ que es el mayor en las 15 instancias.

Es importante notar que esta relajación nos arroja una aproximación a la respuesta real del problema, la cual puede ser mejorada modificando los criterios de parada de las iteraciones o aumentando la cantidad de restricciones violadas que se agregan en cada iteración, sin embargo esto aumentaría el tiempo de computo.

(n,k)	ite	Z	GAP	Tiempo
(4,2)	3	16	0,00 %	0,057
(6,2)	3	57	0,00 %	0,055
(6,3)	2	70	0,00 %	0,033
(10,2)	2	133	0,53 %	0,079
(10,5)	7	171	0,00 %	0,143
(15,3)	2	354	0,25 %	0,206
(15,5)	3	379	0,11 %	0,115
(20,4)	2	686	0,07 %	0,126
(20,5)	2	695	0,10 %	0,074
(25,5)	2	1000	0,06 %	0,079
(30,6)	2	1475	0,03 %	0,094
(40,8)	2	2683	0,00 %	0,123
(50,5)	2	4290	0,00 %	1,288
(50,25)	4	4363	0,00 %	0,246
(100,20)	2	16039	0,00 %	0,298

Cuadro 3.3: Resumen comportamiento de $\mathcal{F}_2$

3.3. Comparación

Al comparar el comportamiento de $\mathcal{F}_1$ y $\mathcal{F}_2$ podemos ver que la segunda relajación tiene un mejor rendimiento en el tiempo de computo empleado principalmente en instancias con $n \geq 20$, esto a costa de sacrificar la solución exacta del problema por una aproximación.

En las dos relajaciones, el desempeño es mejor en las instancias con k grande, ($k \approx \sqrt{n}$) o mayor, obteniendo en la mayoría de casos un tiempo de resolución adecuado. Es decir, tal como se expresa en [5] estas relajaciones basadas en un enfoque lineal son prometedoras en los casos de partición en muchos grupos y en contraste es débil si k es pequeño.

Capítulo 4

Conclusiones y recomendaciones

4.1. Conclusiones

El estudio de la Programación semidefinida para la solución del problema de particionamiento de grafos k-way balanceado con restricciones de peso nos permite formular varios modelos para resolver el problema propuesto, desde distintos enfoques y con varios métodos de resolución, en este trabajo se utilizó un enfoque lineal para la construcción de dos relajaciones a partir de una formulación semidefinida del problema.

Los modelos propuestos en este trabajo, recogidos en las relajaciones $\mathcal{F}_1$ y $\mathcal{F}_2$, nos permiten tener una resolución exacta o una aproximación, respectivamente, al problema de particionamiento de grafos k-way balanceado con restricciones de peso y fueron implementados en el lenguaje de programación Python tal como se puede ver en el Anexo.

Las instancias generadas para la experimentación fueron resueltas en su mayoría tanto por la relajación $\mathcal{F}_1$ como por $\mathcal{F}_2$, esta última, tal como se expresa en [5] y podemos ver por sus resultados computacionales, nos indica que el proceso iterativo propuesto en esta relajación conduce a resultados de alta calidad.

4.2. Recomendaciones

Se recomienda que en trabajos siguientes se implemente las relajaciones desarrolladas en el presente documento en instancias específicas basadas en problemas reales, instancias como las estudiadas por [10] y así evaluar el comportamiento de las relajaciones en problemas con datos no simulados.

En trabajos siguientes se podría estudiar la implementación de otras formulaciones basadas en la Programación Semidefinida, sea para resolver el problema propuesto en el presente documento, como para otros problemas de particionamiento de grafos, así como emplear relajación semidefinida presentada en [5] para la formulación desarrollada en el presente documento.

Por otro lado, se propone estudiar la solución del problema, por medio de métodos de resolución propios de la Programación Semidefinida como el método *Primal-Dual Interior-Point* presentado en [19].

Capítulo A

ANEXO I

Códigos de implementación

Código para la creación de instancias.

```
# Cargamos librerías
from igraph import *
import random

# Definición de la instancia (número de nodos)
n = 6

# creación del grafo
random.seed(0)
g = Graph.GRG(n,0.6)

# lista con el nombre de los nodos, i.e. A=[0,1,...,n]
A = [ ]
for i in range(n):
    A.append(i)
g.vs["label"] = A

# lista con los pesos (w_i) de los nodos, distribución ↘
  uniforme entre 0.1 y 0.9.
W = [ ]
random.seed(50)
```

```
for i in range(n):
    W.append(random.randint(1,9)/10)
g.vs["peso"] = W

# lista con los pesos (d_ij) de las aristas, distribución uniforme entre 1 y 9.
e = g.ecount()                          # número de aristas
random.seed(10)
B = [ ]
for i in range (e):
    B.append(random.randint(1,9))
g.es["flujo"] = B

#listas creadas para vizualización
ws = [(A[i],W[i]) for i in range(n)]    # lista con la dupla (i,w(i))
E = g.get_edgelist()                    # lista con la dupla (i,j)
ds = [(E[i],B[i]) for i in range(e)]    # lista con la dupla ((i,j),d(ij))

# Gráfico del grafo.
random.seed(3)
layout = g.layout_reingold_tilford(root=[2])
layout = g.layout("fr")
plot(g, layout=layout, bbox=(500, 500), edge_label = ds, edge_width = 0.3, edge_label_size = 7.5, vertex_size = 25, vertex_color = 'white', vertex_label_size = 7 , vertex_label = ws)
```

Implementación de la primera formulación $\mathcal{F}_1$.

```
# Cargamos librerías
from igraph import *
import random
from gurobipy import *
import numpy as np

# Definición instancia
n = 4          # número de aristas
k = 2          # número de clusters

# Cálculo de la cardinalidad de los clusters
o = int(n/k)

# Creoción del grafo
random.seed(0)
g = Graph.GRG(n, 0.6)

A = [ ]
for i in range(n):
    A.append(i)
g.vs["label"] = A


W = [ ]
random.seed(50)
for i in range(n):
    W.append(random.randint(1,9)/10)
g.vs["peso"] = W


e = g.ecount()
random.seed(10)
B = [ ]
for i in range (e):
    B.append(random.randint(1,9))
g.es["flujo"] = B

# listas creadas para vizualización
ws = [(A[i],W[i]) for i in range(n)]    # lista con la ↘
   dupla (i,w(i))
E = g.get_edgelist()                    # lista con la ↘
   dupla (i,j)
ds = [(E[i],B[i]) for i in range(e)]    # lista con la ↘
   dupla ((i,j),d(ij))
```

```
# Calculamos mínimo y máximo con la media y la varianza de la distribución
WL = np.mean(W)*o - np.var(W) - 1 # peso mínimo
WU = np.mean(W)*o + np.var(W) + 1 # peso máximo

# Definimos conjuntos de indexación
I = tuplelist(range(0, n))              # indexación para los vertices
P = tuplelist(range(1, k+1))            # indexación para los clusters

# Generamos la libreria de datos de arcos y pesos de los arcos
arcs = g.get_edgelist()                 # lista de arcos del grafo
J = list(range(0, len(B)))              # Conjunto de indexación de los arcos
datos = {arcs[j]:B[j] for j in J}
arcos, d = multidict(datos)

# construcción del vector de pesos.
MD = []
for i in I :
    for j in I :
        MD.append((i,j))                # matriz (nxn)
                                        # [(0,0),(0, 1),...,(0,n),(1,0),...(5,5)]
R = tuplelist(range(0, len(MD)))    # Conjunto de indexacion de elementos de la matriz anterior

MG = []
for r in R:
    if MD[r] in arcos:
        MG.append(d[MD[r]])
    else :
        MG.append(0)
D = np.array(MG).reshape(n,n) # matriz de pesos D, triangular superior.
for i in I :
    for j in I :
        D[j][i] = D[i][j]       # matriz D in R (nxn) matriz de pesos D

# Construcción del Laplaciano associado a la matriz D. L=(lij)(nxn)
    # Por definición, -- lii = sum j (dij), and
    #                 -- lij =     dij for i!=j
la = []
for i in I :
```

```
    for j in I :
        if i == j :
            la.append(int(sum(D[:, i:j+1])))
        else :
            la.append(-D[i][j])
L = np.array(la).reshape(n,n)  #L in R (nxn)

# Constucción de la libreria con los elementos de la matriz L
l = {(i,j):0 for i in I for j in I}  # clase dict de ceros
for i in I:
    for j in I:
        if i == j:
            l[(i,j)] = sum(D[j])
        else:
            if (i,j) in arcos:
                l[(i,j)] = -D[i][j]
                l[(j,i)] = -D[i][j]

# Constucción de la libreria con los pesos de los nodos
w = {i: W[i] for i in I}

# Resolución modelo
m = Model("m_lp_met")  # Modelo relajación con enfoque lineal

# Definimos la variable -- Y R(n x n)
# Y = X'X -> y_ij = 1 si i&j estan en el mismo cluster
#                   0 caso contrario

y = m.addVars(I, I, vtype = GRB.BINARY, name="y")

# Definimos la función objetivo
# max 0.5 tr(LY) = 0.5 *  sum_(i 1-n){ sum_(r 1-n){ l[i,r]*y[r,i] }}
m.setObjective( 0.5*quicksum(l[i,r] * y[r,i] for r in I for i in I), GRB.MAXIMIZE)

# RESTRICCIONES
# diag(Y) = e == yii = 1 para todo i
m.addConstrs((y[i,i] == 1 for i in I) , "diag")

# Ye = o*e donde o = int(n/k)
m.addConstrs((y.sum('*',i) == o for i in I), "eme")

# 0     Y     J
m.addConstrs((y[i,j] <= 1 for i in I for j in I ) , "men_uno")
m.addConstrs((y[i,j] >= 0 for i in I for j in I ) , "
```

```
    may_cero")

# Y MET donde MET = {Y - (yij) : yij + yik 1 + yjk, i , j , k}.
m.addConstrs((y[i,j] + y[i,k] <= 1 + y[j,k] for i in I for j in I for k in I ) , "MET")

# Peso de nodos
m.addConstrs((quicksum(w[j]*y[i,j] for j in I) - w[i]*y[i,i] >= WL - w[i] for i in I ) , "peso_inf")
m.addConstrs((quicksum(w[j]*y[i,j] for j in I) - w[i]*y[i,i] <= WU - w[i] for i in I ) , "peso_sup")

#optimizamos
m.optimize()

# Impresión de resultados
print('Instancia ({};{}) '.format(n,k))
print('resumen:\nn={}, k={}, m={}\nZ = {}, GAP = {}%, Tiempo = {}s'.format(n,k,e,m.objVal, m.MIPGap, m.Runtime))

print('Conjuntos')
for i in I:
    pr = []
    for j in I:
        if y[i,j].x > 0 and i < j:
            pr.append(j)
    pr.append(i)
    if len(pr) == 0:
        print(pr)
```

Implementación de la segunda formulación $\mathcal{F}_2$.

```
# Cargamos librerías
from igraph import *
import random
from gurobipy import *
import numpy as np

# Definición instancia
n = 100                 # número de aristas
K = 20                  # número de clusters

# Cálculo de la cardinalidad de los clusters
o = int(n/K)

# Creación del grafo
random.seed(0)
g = Graph.GRG(n, 0.6)

A = [ ]
for i in range(n):
    A.append(i)
g.vs["label"] = A

W = [ ]
random.seed(50)
for i in range(n):
    W.append(random.randint(1,9)/10)
g.vs["peso"] = W


e = g.ecount()
random.seed(10)
B = [ ]
for i in range (e):
    B.append(random.randint(1,9))
g.es["flujo"] = B

# listas creadas para vizualización
ws = [(A[i],W[i]) for i in range(n)]    # lista con la ↘
   dupla (i,w(i))
E = g.get_edgelist()                    # lista con la ↘
   dupla (i,j)
ds = [(E[i],B[i]) for i in range(e)]    # lista con la ↘
   dupla ((i,j),d(ij))

# Calculamos mínimo y máximo con la media y la varianza de↘
```

```
    la distribución
WL = np.mean(W)*o - np.var(W) - 1 # peso mínimo
WU = np.mean(W)*o + np.var(W) + 1 # peso máximo


# Definimos conjuntos de indexación
I = tuplelist(range(0, n))    # indexación para los ↘
   vertices
P = tuplelist(range(1, K+1)) # indexación para los ↘
   clusters

# Generamos la libreria de datos de arcos y pesos de los ↘
   arcos
arcs = g.get_edgelist()              # lista de arcos del ↘
   grafo
J = list(range(0, len(B)))           # Conjunto de indexaci↘
   ón de los arcos
datos = {arcs[j]:B[j] for j in J}
arcos, d = multidict(datos)

# construcción del vector de pesos.
MD = []
for i in I :
    for j in I :
        MD.append((i,j))          # matriz (nxn)
                                  # [(0,0),(0, 1),...,(0,n)↘
                                     ,(1,0),...(5,5)]
R = tuplelist(range(0, len(MD))) # Conjunto de indexacion ↘
   de elementos de la matriz anterior

MG = []
for r in R:
    if MD[r] in arcos:
        MG.append(d[MD[r]])
    else :
        MG.append(0)
D = np.array(MG).reshape(n,n)  # matriz de pesos D, ↘
   triangular superior.
for i in I :
    for j in I :
        D[j][i] = D[i][j]      # matriz de pesos D, full.

# Construcción del Laplaciano associado a la matriz D. L=(↘
   lij)(nxn)
    # Por definición, -- lii = sum j (dij), and
    #                 -- lij =     dij for i!=j
la = []
for i in I :
    for j in I :
```

```
            if i == j :
                la.append(int(sum(D[:, i:j+1])))
            else :
                la.append(-D[i][j])
    L = np.array(la).reshape(n,n)  #L in R (nxn)

    # Constucción de la libreria con los elementos de la matriz L
    l = {(i,j):0 for i in I for j in I}  # clase dict de ceros

    for i in I:
        for j in I:
            if i == j:
                l[(i,j)] = sum(D[j])
            else:
                if (i,j) in arcos:
                    l[(i,j)] = -D[i][j]
                    l[(j,i)] = -D[i][j]

    # Constucción de la libreria con los pesos de los nodos
    w = {i: W[i] for i in I}

    # Resolución del modelo
    # Primera Iteración

    m = Model("m_lp_met_ite")

    # Definimos la variable -- Y R(n x n)
    # Y = X'X -> y_ij = 1 si i&j estan en el mismo cluster
    #                   0 caso contrario

    y = m.addVars(I, I, vtype = GRB.BINARY, name="y")

    # Definimos la función objetivo
    # max 0.5 tr(LY) = 0.5 *  sum_(i->1-n){ sum_(r 1-n){ l[i,r]*y[r,i] }}
    m.setObjective( 0.5*quicksum(l[i,r] * y[r,i] for r in I for i in I), GRB.MAXIMIZE)

    # RESTRICCIONES

    # diag(Y) = e == yii = 1 para todo i
    m.addConstrs((y[i,i] == 1 for i in I) , "diag")

    # Ye = o*e donde o = int(n/k)
    m.addConstrs((y.sum('*',i) == o for i in I), "eme")

    # 0     Y     J
    m.addConstrs((y[i,j] <= 1 for i in I for j in I ) , "
```

```
    men_uno")
m.addConstrs((y[i,j] >= 0 for i in I for j in I ) , "↘
    may_cero")

#  Peso de nodos
m.addConstrs((quicksum(w[j]*y[i,j] for j in I) - w[i]*y[i,↘
    i] >= WL - w[i] for i in I ) , "peso_inf")
m.addConstrs((quicksum(w[j]*y[i,j] for j in I) - w[i]*y[i,↘
    i] <= WU - w[i] for i in I ) , "peso_sup")

#optimizamos
m.optimize()

# Conjuntos de control de iteraciones
ite = 1              # Contador de iteraciones
MCONTROL = []        # Conjunto de tripletas (ijk) incluidas↘
    como restricciones en el modelo.
indices = []         # Conjunto de tripletas (ijk) tq violan↘
    las restricciones en cada iteración
T = []               # lista para reportar el tiempo de cada↘
    iteración
R = []               # lista para reportar la cantidad de ↘
   restricciones violadas
Z = []               # lista para reportar el valor de la ↘
   función objetivo en cada iteración

# Busqueda de iteraciones violadas en la primera iteración
for i in I:
    for j in I:
        for k in I:
            if y[i,j].x + y[i,k].x - y[j,k].x > 1:
                indices.append((i,j,k))

indices = list(set(indices))    # se actualizan las ↘
   tripletas violadas en esta iteración
T.append(m.Runtime)             # se actualiza el valor del↘
    tiempo de cada iteración
Z.append(m.objVal)              # se actualiza el valor de ↘
   la función objetivo de cada iteración
R.append(len(indices))          # se actualiza la cantidad ↘
   de violaciones violadas en cada iteración

# cantidad de iteraciones a tomar.
c_res = 0

# Iteraciones

while len(MCONTROL) < 5*n and len(indices) > 0:
    c_res = ite*n*5             # definimos cantidad de ↘
```

```
    restricciones a tomar
met = []                    # lista de restricciones a ↘
    agregarse en esta iteración

# Agregamos las restricciones más violadas. de los ↘
    indices i, j, k tal que yij + yik - yjk > 1
if len(indices) > c_res :  # Tomamos aleatoramente '↘
    c_res' restricciones violadas del arreglo de ↘
    indices.
     indicesaux = indices
     while len(met) < c_res:
          a = random.randint(0,len(indicesaux)-1)
          met.append(indicesaux[a])
          indicesaux.pop(a)
else :
     met = indices

# Agregamos a la lista de control las restricciones ↘
    que se tomaron en esta iteración
for i in range(len(met)):
     MCONTROL.append(met[i])

# actualizamos el modelo y agregamos las restricciones↘
     Y     MET = {Y - (yij) : yij + yik     1 + yjk, ↘
      i  , j, k}.
m.update()
III = list(range(0, len(met)))     # Conjunto de ↘
    indexación
m.addConstrs((y[met[i][0],met[i][1]] + y[met[i][0],met↘
    [i][2]] <= 1 + y[met[i][1],met[i][2]] for i in III ↘
    ) , "MET")
m.update()

#resolvemos el modelo
m.optimize()

# Actualizamos restricciones violadas.
indices = []
for i in I:
     for j in I:
               for k in I:
                    if y[i,j].x + y[i,k].x - y[j,k].x > 1:
                         indices.append((i,j,k))
indices = list(set(indices))

# Actualizamos valores a reportar
T.append(m.Runtime)
R.append(len(indices))
Z.append(m.objVal)
```

```
    ite = ite + 1

#visualización de resultados
print('Resumen: Instancia({};{}) iterada'.format(n,K))
print('n={}, k={}, m={}\nZ = {}, GAP = {}%, Tiempo = {} s \niteraciones {}'.format(n,K,e,m.objVal, m.MIPGap, sum(T),ite))
for i in range(len(T)):
    print('iteración {}, Tiempo: {} s, R.V. {}, Z={}'.format(i+1,T[i],R[i],Z[i]))
print('Resultado:')
for i in I:
    pr = []
    for j in I:
        if y[i,j].x > 0 and i < j:
            pr.append(j)
    pr.append(i)
    if len(pr) == o:
        print(pr)
```

Referencias bibliográficas

[1] Marcelino Felipe Alvarez Nuñez. Teoría de grafos. 2013.

[2] Xiaoqing Bai and Hua Wei. A semidefinite programming method with graph partitioning technique for optimal power flow problems. *International Journal of Electrical Power & Energy Systems*, 33(7):1309–1314, 2011.

[3] Robert M Freund. Introduction to semidefinite programming (sdp). *Massachusetts Institute of Technology*, pages 8–11, 2004.

[4] Jean B Lasserre. Linear programming with positive semi definite matrices. In *Proceedings of 1995 34th IEEE Conference on Decision and Control*, volume 2, pages 1127–1132. IEEE, 1995.

[5] A. Lisser and F. Rendl. Graph partitioning using linear and semidefinite programming. *Mathematical Programming*, 95:91–101, 2003.

[6] Amador Menéndez Velázquez et al. Una breve introducción a la teoría de grafos. *Suma*, 1998.

[7] Inc. Gurobi Optimization. *Gurobi optimizer reference manual*. 2022.

[8] Gábor Pataki. Cone lps and semi-definite programs: Facial structure, basic solutions, and simplex method. Technical report, Univ. of Michigan, Ann Arbor, MI (United States), 1994.

[9] Diego Recalde, Daniel Severín, Ramiro Torres, and Polo Vaca. Balanced partition of a graph for football team realignment in ecuador. In *International Symposium on Combinatorial Optimization*, pages 357–368. Springer, 2016.

[10] Diego Recalde, Daniel Severín, Ramiro Torres, and Polo Vaca. 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(3):916–936, 2018.

[11] Diego Recalde, Ramiro Torres, and Polo Vaca. An exact approach for the multi-constraint graph partitioning problem. *EURO Journal on Computational Optimization*, 8:289–308, 2020.

[12] Franz Rendl. Semidefinite programming and combinatorial optimization. *Applied Numerical Mathematics*, 29(3):255–281, 1999.

[13] Adrián Ayastuy Rodriguez. Optimización convexa. 2014.

[14] Abbas Musleh Salman, Ahmed Alridha, and Ahmed Hadi Hussain. Some topics on convex optimization. In *Journal of Physics: Conference Series*, volume 1818, page 012171. IOP Publishing, 2021.

[15] Renata Sotirov. An efficient semidefinite programming relaxation for the graph partition problem. *INFORMS Journal on Computing*, 26(1):16–30, 2014.

[16] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *Society for Industrial and Applied Mathematics*, 38:44–95, 1996.

[17] Angelika Wiegele and Shudian Zhao. Tight sdp relaxations for cardinality-constrained problems. *arXiv preprint arXiv:2107.11338*, 2021.

[18] Henry Wolkowicz, Romesh Saigal, and Lieven Vandenberghe. *Handbook of Semidefinite Programming*, volume 27. 2000.

[19] Henry Wolkowicz and Qing Zhao. Semidefinite programming relaxations for the graph partitioning problem. *Discrete Applied Mathematics*, 96-97:461–479, 1999.

[20] Qing Zhao, Stefan Karisch, Franz Rendl, and Henry Wolkowicz. Semidefinite programming relaxations for the quadratic assignment problem. *Journal of Combinatorial Optimization*, 2(1):71–109, 1998.