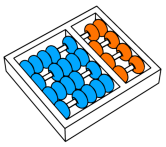
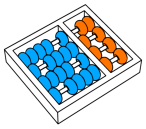


Universidade Estadual de Campinas
Instituto de Computação



Matheus Jun Ota

The Balanced Connected k-Partition Problem: Polyhedra and Algorithms

O Problema da k-Partição Conexa Balanceada: Poliedros e Algoritmos

CAMPINAS
2020

Matheus Jun Ota

The Balanced Connected k-Partition Problem: Polyhedra and Algorithms

O Problema da k-Partição Conexa Balanceada: Poliedros e Algoritmos

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Computer Science.

Supervisor/Orientador: Prof. Dr. Flávio Keidi Miyazawa
Co-supervisor/Coorientador: Prof. Dr. Phablo Fernando Soares Moura

Este exemplar corresponde à versão final da Dissertação defendida por Matheus Jun Ota e orientada pelo Prof. Dr. Flávio Keidi Miyazawa.

CAMPINAS
2020

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

Ot1b

Ota, Matheus Jun, 1994-
The balanced connected k-partition problem : polyhedra and algorithms / Matheus Jun Ota. – Campinas, SP : [s.n.], 2020.

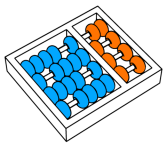
Orientador: Flávio Keidi Miyazawa.
Coorientador: Phablo Fernando Soares Moura.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Teoria dos grafos. 2. Programação linear inteira. 3. Otimização combinatória. 4. Combinatória poliédrica. 5. Algoritmos branch-and-cut. I. Miyazawa, Flávio Keidi, 1970-. II. Moura, Phablo Fernando Soares. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

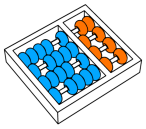
Informações para Biblioteca Digital

Título em outro idioma: O problema da k-partição conexa balanceada : poliedros e algoritmos
Palavras-chave em inglês:
Graph theory
Integer linear programming
Combinatorial optimization
Polyhedral combinatorics
Branch-and-cut algorithms
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Flávio Keidi Miyazawa [Orientador]
Ricardo Fukasawa
Eduardo Uchoa Barboza
Data de defesa: 13-10-2020
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-3730-6420
- Currículo Lattes do autor: http://lattes.cnpq.br/7738972154001613



Universidade Estadual de Campinas
Instituto de Computação



Matheus Jun Ota

The Balanced Connected k-Partition Problem: Polyhedra and Algorithms

O Problema da k-Partição Conexa Balanceada: Poliedros e Algoritmos

Banca Examinadora:

- Prof. Dr. Flávio Keidi Miyazawa
  Universidade Estadual de Campinas
- Prof. Dr. Ricardo Fukasawa
  Universidade de Waterloo
- Prof. Dr. Eduardo Uchoa Barboza
  Universidade Federal Fluminense

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 13 de outubro de 2020

You can often view glimpses of ingeniousness not as inexplicable miracles, but as the residue of experience. And when you do, the idea of genius goes from being mesmerizing to instead being actively inspirational.
(Grant Sanderson (3b1b))

Agradecimentos

Devido às limitações referentes ao tamanho médio desse tipo de seção, os seguintes agradecimentos são mais breves do que deveriam. Naturalmente, começo pela minha família, mais especificamente, meus pais: Márcio e Celina. Muito obrigado pelo amor e pelos ensinamentos. É um tanto quanto óbvio, mas acho importante ressaltar que tudo que sou, ou possa vir a ser, tem relação direta com a educação que vocês me deram. Todas minhas conquistas também são suas e sou eternamente grato por ser filho de vocês. Em seguida, agradeço aos meus irmãos Márcio e Marcelo e suas respectivas companheiras Karol e Carla. Cada um de nós temos nossas peculiaridades, e assim sendo, sempre aprendo muito com todos. Agradeço também a recém-chegada Mariana, minha sobrinha que trouxe muita alegria e felicidade para nossa família. Por fim, sou grato pelos nossos encontros. Ainda que estejamos muitas vezes distantes fisicamente, é sempre muito bom quando nos encontramos. Vocês são grandes influências para mim e, de certa forma, estão presentes em tudo que faço.

Em seguida, agradeço aos meus orientadores: o professor Flávio K. Miyazawa e o professor Phablo F. S. Moura. Aqui agradeço também à professora Yoshiko Wakabayashi, que participou de todas nossas reuniões e foi essencial para os bons resultados obtidos nesse projeto. Obrigado pela amizade, por todo o trabalho que conduzimos juntos e pelo conhecimento que vocês compartilharam comigo durante esses dois anos. É um privilégio muito grande poder trabalhar com vocês. Sinto que devo agradecer novamente ao professor Flávio, que me orienta desde a iniciação científica. Desde que comecei a conduzir pesquisas em otimização combinatória com o Flávio eu me sinto mais realizado e feliz. Naquela época, eu não tinha experiência com otimização e meu desempenho acadêmico não era dos melhores. Ainda assim, sempre com muita paciência, o Flávio me orientou. Acredito que hoje sou um cientista da computação melhor, e isso decorre do trabalho que realizei com o Flávio.

Esse é o meu oitavo ano na Unicamp, e durante esse tempo já morei em quatro repúblicas. Agradeço aos amigos que fiz durante esse tempo (as listas seguintes não são exaustivas): Millôr, Pepeu, Yan, Kurumin, Vitinho, Jack, Yago, Seiji, Marcelo, Hugo, Udi, Tuti, Taffarel, Diogo, Antônio, Borges, Roger, Perônio, Murro, Buxexa, Hermes, Ura, Shima, Alegria, Capitão, Jobs, Julia, Andreia, ... Esses foram bons anos, e é inegável que vocês são relevantes nisso. Agradeço também aos meus colegas de Londrina: Caiot, Renan, Carlos e Takeo; e aos membros e ex-membros do LOCo: Ulysses, Guilherme, Alan, Lucas, Marcelo, Natanael, Yulle, Deyvison, Vitor (os dois), Vinicius, Mateus, Mauro (os dois), Alonso, Paulo, Welverton, Celso, Mayara, Alessandra, Enoque, Pedro, Renato, Felipe, ...

Agradeço ao Davi que, em 2011, me incentivou a estudar para o vestibular da Unicamp. Essa universidade foi um grande presente na minha vida. Tenho certeza que a Unicamp fez de mim uma pessoa melhor. Digo isso não só pelo conhecimento de elite que a universidade produz, como também pelas vivências e amizades. O Instituto de Computação foi muito

importante para definir o que sou hoje. Dessa forma, agradeço de maneira especial aos docentes da área de teoria da computação do IC, que ministram ótimas aulas e sempre foram abertos a responderem as minhas dúvidas. Também agradeço aos funcionários da secretaria do IC, que pacientemente responderam, repetidas vezes, várias das minhas dúvidas sobre alguns processos burocráticos.

Devo agradecer também à Tayná, pelo carinho e companheirismo. Acho bonito e precioso o quanto podemos ser sinceros e aprender um com o outro. Além disso, deve ser possível contar em uma única mão o número de fisioterapeutas no mundo que escutariam, quase todas as noites, alguém falando sobre grafos, algoritmos e matemática. Fico feliz de compartilharmos nossas vidas.

O presente trabalho foi realizado com apoio do CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil. Dessa forma, agradeço aos contribuintes brasileiros que ajudaram a financiar os meus estudos. Agradeço também aos contribuintes da Bolsa Alumni. Creio que essa bolsa é um importante incentivo para a pesquisa desenvolvida no Instituto de Computação da Unicamp.

Em 1676, Isaac Newton escreveu em uma carta para Robert Hooke: "Se eu vi mais longe, foi por estar sobre ombros de gigantes". Deve ser extremamente clichê utilizar essa frase nessa ocasião, mas os poucos vislumbres que tive do que se encontrava atrás das montanhas foi devido aos muitos gigantes que me ajudaram.

Resumo

Dado um inteiro fixo $k \geq 2$, o *problema da k-partição conexa balanceada* (BCP_k) consiste em particionar um grafo em k subgrafos conexos mutuamente disjuntos e com pesos similares. Formalmente, dado um grafo conexo G com pesos não-negativos nos vértices, desejamos encontrar uma partição $\{V_i\}_{i=1}^{k}$ de $V(G)$ tal que cada classe V_i induz um subgrafo conexo em G, e o peso da classe com menor peso é o maior possível. Esse problema, conhecido por ser $\mathcal{NP}$-difícil, foi muito investigado por diversas abordagens e perspectivas: algoritmos exatos, algoritmos de aproximação para alguns valores de k ou classes de grafos, variantes próximas do problema e resultados de inaproximabilidade. Do ponto de vista prático, o BCP_k é utilizado para modelar problemas em processamento de imagens, análise de clusters, sistemas operacionais e robótica. Nesse trabalho, propomos duas formulações baseadas em Programação Linear Mista e uma formulação baseada em Programação Linear Inteira para o BCP_k. As primeiras duas formulações são baseadas em fluxos e possuem um número polinomial de variáveis e restrições. A última formulação contém somente variáveis binárias e um número potencialmente grande de desigualdades que podem ser separadas em tempo polinomial. Introduzimos novas desigualdades válidas para esse último modelo e projetamos algoritmos de separação polinomial correspondentes. Além disso, apresentamos resultados poliédricos associados a essa formulação. Pelo que sabemos, não existem resultados dessa natureza para o BCP_k na literatura. Utilizando a plataforma OpenStreetMap e os dados públicos sobre a criminalidade em certas regiões, geramos novas instâncias baseadas na aplicação de patrulhamento policial. Experimentos computacionais mostram que os algoritmos exatos baseados nas nossas formulações são superiores aos melhores métodos exatos presentes na literatura.

Abstract

Given a fixed integer $k \geq 2$, the *balanced connected k-partition problem* (BCP_k) consists of partitioning a graph into k mutually vertex-disjoint subgraphs of similar weight. More formally, given a connected graph G with non-negative weights on the vertices, we want to find a partition $\{V_i\}_{i=1}^{k}$ of $V(G)$ such that each class V_i induces a connected subgraph of G, and the weight of a class with the minimum weight is as large as possible. This problem, known to be $\mathcal{NP}$-hard, has been largely investigated under different approaches and perspectives: exact algorithms, approximation algorithms for some values of k or special classes of graphs, close variants of the problem, and inapproximability results. On the practical side, BCP_k is used to model many applications arising in image processing, cluster analysis, operating systems and robotics. We propose two MILP formulations and one ILP formulation for BCP_k. The first two are based on flows and have a polynomial number of constraints and variables. The last formulation contains only binary variables and a potentially large number of constraints that can be separated in polynomial time in the corresponding linear relaxation. We introduce new valid inequalities for this last model and design corresponding polynomial-time separation routines. Furthermore, we present polyhedral results based on this ILP formulation. To our knowledge, this is the first polyhedral study for BCP_k in the literature. Using the OpenStreetMap platform and public crime data, we generated new instances based on a police patrolling application. Our computational experiments show that the exact algorithms based on the proposed formulations outperform the other exact methods presented in the literature.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

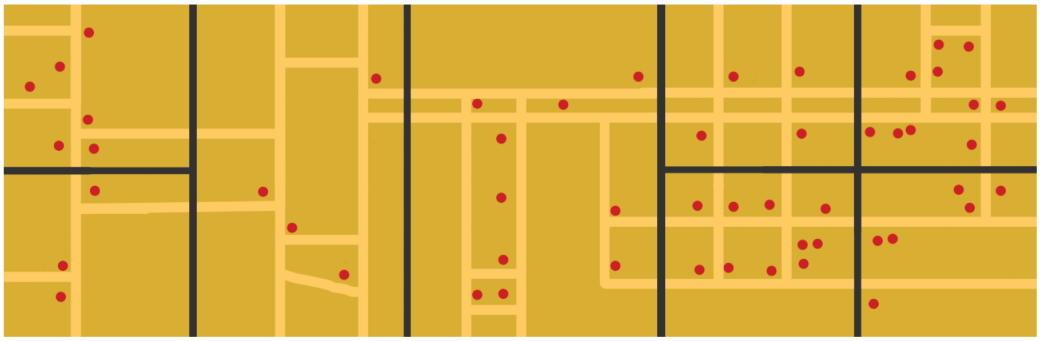
Let G be a connected undirected graph with vertex set $V(G)$ and edge set $E(G)$. When the graph G is clear from the context, we use the notation V and E, instead. For an integer t, the symbol $[t]$ denotes the set $\{1, 2, \ldots, t\}$ if $t \geq 1$ and $\emptyset$ otherwise. Let k be a fixed positive integer. A *k-partition* of G is a collection $\{V_i\}_{i\in[k]}$ of nonempty subsets of V such that $\bigcup_{i=1}^{k} V_i = V(G)$, and $V_i \cap V_j = \emptyset$, for all $i, j \in [k]$, $i \neq j$. We say that each set V_i is a *class* of the partition. Moreover, when the classes are indexed, we sometimes refer to V_i as the *i-th class*. From now on, we assume that $|V| \geq k$, otherwise G does not admit a k-partition. We say that a k-partition $\{V_i\}_{i\in[k]}$ of G is *connected* if, for every $i \in [k]$, the subgraph of G induced by V_i is connected.

For any set of numbers S, we denote by $S_{\geq}$ (resp. $S_{>}$) the set of non-negative (resp. positive) elements of S. Let $w\colon V \to \mathbb{Q}_{\geq}$ be a function that assigns weights to the vertices of G. For every subset $V' \subseteq V$, we define $w(V') = \sum_{v\in V'} w(v)$. For the sake of simplicity, for any subgraph H of G, we use $w(H)$ meaning $w(V(H))$. In the *balanced connected k-partition problem* (BCP_k), we are given a vertex-weighted connected graph, and we seek a connected k-partition that maximizes the weight of the lightest class.

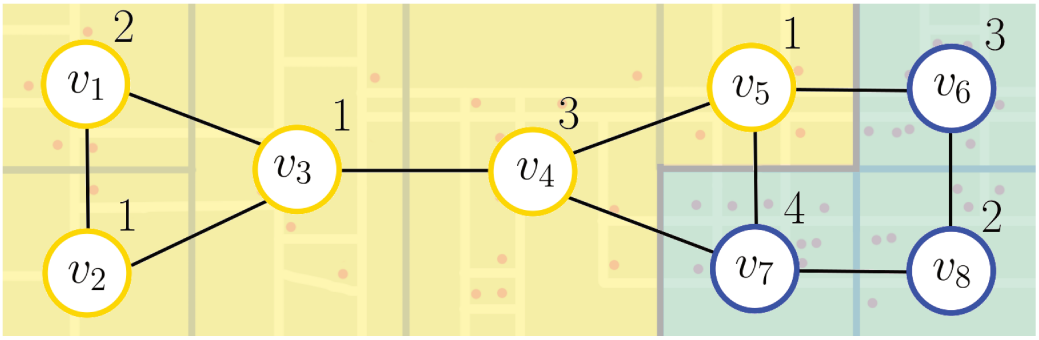
Problem 1 Balanced Connected k-Partition (BCP_k)
Instance: *a connected graph G and a vertex-weight function $w\colon V \to \mathbb{Q}_{\geq}$.*
Find: *a connected k-partition $\{V_i\}_{i\in[k]}$ of $V(G)$.*
Goal: *maximize* $\min_{i\in[k]}\{w(V_i)\}$.

Based on the work of Assunção and Furtado [4], we now illustrate an application of this problem in the context of police patrolling. Suppose we are given a map of a city that is divided into patrolling areas. Moreover, this map contains points representing crime occurrences. We can easily create a graph G_p for such a map. The set of vertices $V(G_p)$ corresponds to the patrolling areas, and an edge $\{p_1, p_2\}$ is in $E(G_p)$ if and only if patrolling area p_1 is adjacent to p_2. Next, we define a function $w_p\colon V(G_p) \to \mathbb{Q}_{\geq}$ that indicates the criminality in an area. Solving BCP_k in (G_p, w_p) is equivalent to attributing contiguous patrolling areas to k police teams, in a way that the criminality is balanced between the teams (see Figure 1.1).

Besides police patrolling, there are several other problems in image processing, data base, operating systems, cluster analysis, education, robotics and metabolic networks that can be modeled as a balanced connected partition problem [7, 30, 31, 34–36, 50]. These



(a) A toy example of a map. The black lines delimit the patrolling areas and the red dots represent crime occurrences.



(b) A corresponding instance (G_p, w_p) and a solution for BCP_2.

Figure 1.1: BCP_k application to police patrolling.

different real-world applications indicate the importance of designing algorithms for BCP_k and reporting on the computational experiments with their implementations. Not less important are the theoretical studies of the rich and diverse mathematical formulations and the polyhedral investigations BCP_k leads to.

1.1 Contributions and Outline

Chapter 2 presents briefly common concepts and notations from the areas of graph theory, integer programming and polyhedral combinatorics. This chapter establishes a common ground for the rest of the thesis. Chapter 3 discusses known results in the literature for BCP_k. We give special attention to the mixed integer linear programming formulations proposed by Matić [37] and Zhou et al. [50].

The subsequent chapters present novel results that advance the state of the art on exact algorithms for BCP_k. Chapter 4 introduces novel flow and multicommodity flow based formulations for the problem. Both formulations are compact, that is, they have a polynomial amount (on the size of the input graph) of variables and constraints.

Chapter 5 presents a novel cut-based ILP formulation. We also show two strong valid inequalities for this formulation. One of these inequalities is shown to be easily separable in polynomial-time. For the other inequality, we design an algorithm that separates it in polynomial time when the input graph is planar — a property that is common among graphs that arise from real world applications. We combine these separation routines in a

branch-and-cut algorithm for BCP_k. A further polyhedral study based on this formulation is presented in Section 5.4. To the best of our knowledge, these are the first polyhedral results for BCP_k described in the literature.

Lastly, we report on computational experiments in Chapter 6. In the same chapter, we also propose new benchmark instances, based on police patrolling applications. These instances were generated using OpenStreetMap [43] and real-world crime data available for the public. Our computational results show that the exact algorithms based on the proposed formulations outperform significantly the previous solving methods due to Matić and Zhou et al. Particularly, we solve grid instances with sizes over 400 times larger than the sizes of the largest instances solved by the previous state-of-the-art exact algorithms.

This work gave rise to two papers. One of them is a short version of this thesis and was published in the proceedings of the International Symposium on Combinatorial Optimization (ISCO 2020) [39]. The other paper is more complete and was published in the European Journal of Operational Research (EJOR) [40].

Chapter 2

Preliminaries

In this chapter, we give a brief overview of concepts and notations from the areas of Graph Theory, Linear Programming and Polyhedral Combinatorics. The reader familiar with these areas can safely skip it. The definitions here follows the terminology presented by Wakabayashi and Ferreira [17] and Nemhauser and Wolsey [42].

2.1 Graph Theory

A graph G is a tuple $(V(G), E(G))$ where $V(G)$ denotes its vertex set and $E(G)$ its edge set. When the graph G is clear from the context, we use the simplified notation V and E instead of $V(G)$ and $E(G)$. Unless otherwise stated, we assume that $n = |V|$ and $m = |E|$. Throughout this text, G is always a graph.

We denote each edge in our graph by $\{u, v\}$, where $u, v \in V$ are the *endpoints* of the edge. Two vertices u and v in G are said to be *adjacent* or *neighbors* if the graph contains an edge $e = \{u, v\}$; moreover, we say that e is *incident* to both u and v. Sometimes, we may simply write $e = uv$ to refer to an edge with endpoints u and v. For any set of vertices $W \subseteq V$, we define the *neighborhood* of W as being the set of vertices adjacent to W, formally $N(W) = \{v \in V \setminus W : \{u, v\} \in E, u \in W\}$. When W contains exactly one vertex v, we use $N(v)$ instead of $N(\{v\})$. In order to refer to the set of edges that connect vertices in W with vertices in $N(W)$, we use the notation $\delta(W)$. In other words, $\delta(W) = \{\{u, v\} \in E : u \in W, v \in V \setminus W\}$.

When H is a graph such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$, we say that H is a *subgraph* of G and denote this relationship by $H \subseteq G$. Let $W \subseteq V$, we use $G[W]$ to designate the subgraph of G with vertex set W and edge set $\{\{u, v\} \in E : u, v \in W\}$. The notation $G - W$ is used to refer to the subgraph $G[V \setminus W]$. Similarly, we use $G - H$, for any $H \subseteq G$, to refer to the subgraph with vertex set $V \setminus V(H)$ and edge set $\{\{u, v\} \in E \setminus E(H) : u, v \in V(G - H)\}$.

Let $p \in \mathbb{Z}_{\geq}$, a *path* P is a non-empty graph such that $p \leq n$, $V(P) = \{v_1, v_2, \ldots, v_p\}$, $|V(P)| = p$ and $E(P) = \{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{p-1}, v_p\}\}$. We say that P *connects* v_1 and v_p or that P is a (v_1, v_p)*-path*. The graph G is said to be *connected* if and only if for any pair of distinct vertices $u, v \in V$, there exists a path in G that connects u and v, that is, there exists $P \subseteq G$ such that P is a (u, v)-path. Let P be a path, the graph C with

vertex set $V(C) = V(P)$ and edge set $E(C) = E(P) \cup \{\{v_1, v_p\}\}$ is called a *cycle*. A *tree* is a connected graph without cycles; a *subtree* is a tree that is a subgraph of a tree. We say that a tree T *spans* the set of vertices $V(T)$. Moreover, a tree $T \subseteq G$ is a *spanning tree* of G if it spans $V(G)$.

In this dissertation we also work with digraphs. A *digraph* D is a tuple $(V(D), A(D))$, where $V(D)$ is its vertex set and $A(D)$ its arc set. An arc $a = (u, v) \in A(D)$, is similar to an edge, but it also has an orientation according to the pair order. Therefore, all the definitions presented for (undirected) graphs, that do not depend on the notion of an orientation, also applies for digraphs. For any arc $(u, v) \in A(D)$, we say that vertex u is the *source* while v is the *target* of the arc.

Let $W \subseteq V(D)$, we denote by $\delta^-(W) = \{(u, v) \in A(D) : u \notin W, v \in W\}$ the *incoming arcs* of W. Similarly, $\delta^+(W) = \{(u, v) \in A(D) : u \in W, v \notin W\}$ represents the *outcoming arcs* of W.

Let $p \leq n$, a *directed path* P is a non-empty digraph such that $V(P) = \{v_1, v_2, \ldots, v_p\}$, $|V(P)| = p$ and $A(P) = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{p-1}, v_p)\}$. An *arborescence* $\overrightarrow{T}$ can be seen as a directed tree; it is a digraph in which there exists a root $r \in V(\overrightarrow{T})$ such that, for any $v \in V(\overrightarrow{T}) \setminus \{r\}$, there is exactly one directed path in $\overrightarrow{T}$ that connects r and v.

2.2 Integer Linear Programming

A *linear program* (LP) is a mathematical tool to formulate optimization problems with linear objective function and linear constraints. All linear programs can be written as

$$\begin{aligned} [\text{LP}] \qquad & \max c^T x \\ & \text{s.t. } Ax \leq b, \\ & \quad x \geq \mathbf{0}, \end{aligned}$$

where $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$ are vector of coefficients, $x \in \mathbb{R}^n$ is a vector of variables, and $A \in \mathbb{R}^{m \times n}$ is a matrix of coefficients. In the forthcoming discussion, the vectors mentioned are always column vectors. Moreover, we denote by x_i the value of the i-th coordinate of vector x. When a vector has all coordinates equals to 1, we denote it by $\mathbf{1}$, and when it has all coordinates as zeroes by $\mathbf{0}$. In this section, we use brackets [ ] to indicate that we are referring to formulations (as in [LP]).

Let $\mathcal{P} = \{x \in \mathbb{R}^n : Ax \leq b, x \geq 0\}$. For reasons that will be clear in Section 2.3.1, $\mathcal{P}$ is said to be the polyhedron associated with formulation [LP]. We sometimes refer to [LP] as maximizing $c^T x$ *over* $\mathcal{P}$. A linear program is *infeasible* if its associated polyhedron $\mathcal{P}$ is empty. In the context of linear programs, we might say that a vector $x \in \mathbb{R}^n$ is a *solution*. A solution is *feasible* for $\mathcal{P}$ if $x \in \mathcal{P}$. Likewise, x is *infeasible* for $\mathcal{P}$ if $x \notin \mathcal{P}$. An *optimal solution* x^* is a feasible solution such that $c^T x^* = \max\{c^T x : x \in \mathcal{P}\}$.

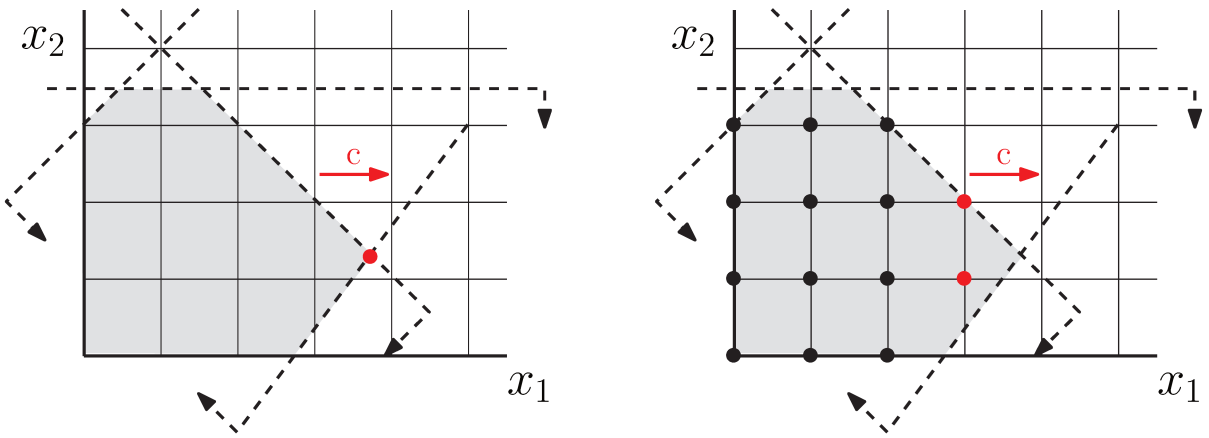
Many methods have been proposed to solve linear programs, like the interior point and the ellipsoid method, but in this thesis we assume we solve linear programs with the simplex method. Although its time complexity is exponential, the simplex method is still used on many linear programming applications, due to its efficiency in solving practical

problems. In addition, Spielman and Teng [45] derived a theoretical explanation for the excellent practical performance of the simplex method. Using probability theory, they showed that the smoothed time complexity of the method is polynomial on the size of the input.

Frequently, combinatorial optimization problems seek integer solutions. In this case, the formulation have additional *integrality constraints*; that is, constraints of the type $x_i \in \mathbb{Z}$, for some $i \in [n]$. When all of the variables in the formulation have to assume integer values, the formulation is said to be an *integer linear program* (ILP); when only some of the variables have to be integer, it is called a *mixed integer linear program* (MILP). Below we give a generic formulation for an ILP.

$$\begin{aligned} [\text{ILP}] \qquad & \max c^T x \\ & \text{s.t. } Ax \leq b, \\ & x \geq \mathbf{0}, \\ & x \in \mathbb{Z}^n. \end{aligned}$$

Usually solving ILP's or MILP's is $\mathcal{NP}$-hard. In order to handle this difficulty, we might consider the linear relaxation of the formulations. A *linear relaxation* of a problem can be obtained by dropping off the integrality constraints (notice that the previous formulation [LP] is a linear relaxation of [ILP]). Linear relaxations are important tools that appear frequently in solving methods for ILP's (or MILP's).



(a) Linear program.

(b) Integer linear program. The black dots and red dots are the integer points in $\mathcal{P}$.

Figure 2.1: Graphical representation of a linear program in $\mathbb{R}^2$. The shaded area corresponds to $\mathcal{P}$ and the dashed lines indicate the linear constraints. The red arrow indicates the direction of growth of vector c and the red dots denote optimal solutions.

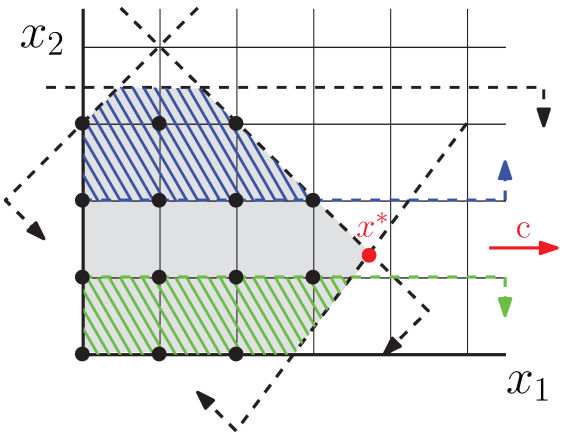
2.2.1 Branch-and-bound

The *branch-and-bound* method is a generic fundamental scheme, and it is based on applying a systematic enumeration of all the candidate solutions. To do so, it has a routine that partitions the solution set into smaller sets (branch), and a second routine, that prunes

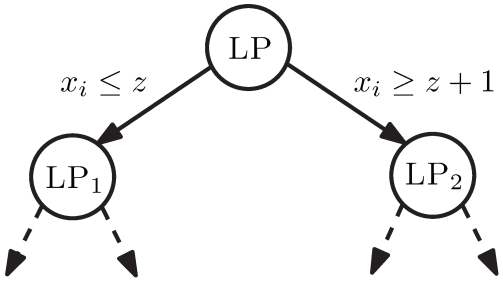
branches by verifying the upper and lower bounds (bound). In this section, we illustrate how the branch-and-bound method can be applied to solve integer linear programs.

Suppose we want to solve the previous [ILP] formulation. Let x^* be an optimal solution for the linear relaxation [LP] and assume that x^* is not integer. In other words, there exists $i \in [n]$ such that $x_i^* \in (z, z+1)$ for some $z \in \mathbb{Z}$. Aiming to avoid the fractional vector x^*, we create two subproblems ([LP_1] and [LP_2]) that partitions (branch) the set of solutions of the original linear relaxation (see Figure 2.2(a)).

$$
\begin{aligned}
[\text{LP}_1] \quad & \max\ c^T x \\
& \text{s.t.}\ \ Ax \leq b, \\
& \qquad x \geq \mathbf{0}, \\
& \qquad x_i \leq z.
\end{aligned}
\qquad\qquad
\begin{aligned}
[\text{LP}_2] \quad & \max\ c^T x \\
& \text{s.t.}\ \ Ax \leq b, \\
& \qquad x \geq \mathbf{0}, \\
& \qquad x_i \geq z + 1.
\end{aligned}
$$



(a) Branching procedure. The blue and green lines represent the branching inequalities. Note how the branching invalidates the fractional solution x^*.



(b) Branch-and-bound tree.

Figure 2.2: Illustration of the branch-and-bound method.

Applying this strategy recursively, we would end up obtaining a tree-like structure for the generated subproblems, where each node v of the tree corresponds to a linear program [LP_v] (Figure 2.2(b)). We refer to this tree as the *branch-and-bound tree*. Since it might have an exponential number of nodes, it is important to avoid exploring unnecessary nodes. For this reason, there are three conditions in which we do not branch and prune (bound) a node v. Let x_v^* be an optimal solution for [LP_v] (here the subscript simply indicates the corresponding branch-and-bound node and not the coordinate of the vector). The conditions are the following.

1 - if [LP_v] is unfeasible;

2 - if the optimal solution x_v^* for [LP_v] is integer;

3 - if the optimal solution for [LP_v] is worse (with respect to the objective function) than the best solution already found.

Conditions 1 and 2 are easy to justify. Let $\mathcal{P}_v$ be the set of feasible solutions for [LP$_v$] and $\mathcal{P}'_v$ be the set of feasible solutions for [LP$_v$] with some additional integrality constraints. Since $\mathcal{P}'_v \subseteq \mathcal{P}_v$, if $\mathcal{P}_v$ is empty, then so is $\mathcal{P}'_v$. Therefore, branching would give no benefits in this case, and the node can be pruned (condition 1). Similarly, the node can also be pruned if x^*_v is integer, since then x^*_v is also optimal for $\mathcal{P}'_v$ (condition 2).

Note that condition 2 follows from the straightforward statement that an optimal solution for $\mathcal{P}'_v$ cannot be better than an optimal solution for $\mathcal{P}_v$. This observation points us towards a third pruning condition. During the execution of the method, we store in a variable y the best integer solution found so far, and use $f(y) = c^T y$ as a lower bound on the cost of an optimal solution. If x^*_v is such that $f(x^*_v) \leq f(y)$, we can safely prune node v (condition 3).

Finally, in our discussion, we partitioned the solution space imposing the inequalities, or *branching rules*, $x_i \leq z$ and $x_i \geq z + 1$. Naturally, other inequalities that partition the solution space and invalidate the fractional solution x^*_v could also be used in the branch-and-bound method.

2.2.2 Branch-and-cut

Suppose we have the following ILP:

$$
\begin{aligned}
\text{[B\&C]} \qquad & \max c^T x \\
& \text{s.t. } Ax \leq b, \\
& \quad\; Cx \leq d, \\
& \quad\; x \geq \mathbf{0}, \\
& \quad\; x \in \mathbb{Z}^n,
\end{aligned}
$$

where A has a polynomial number of lines, but C has an exponential number of lines.

The number of constraints encoded in matrix C can be enormous, and therefore, it might not be feasible to solve the linear relaxation of this formulation. One way of treating this difficulty is with the *cutting plane method*. We remark here that this method can be used to solve LP's as well as ILP's. However, we address here the situation where the cutting plane method is being used to solve LP's with an exponential number of constraints. In Section 2.3.2 we elaborate on the fact that solving ILP's is equivalent to solving LP's with a very large number of (probably unknown) inequalities.

Geometrically, we can think of the cutting plane method in the following way. Let $\mathcal{P}$ be the polyhedron associated with the linear relaxation of [B&C]. Additionally, let $\mathcal{Q} \supseteq \mathcal{P}$ be a relaxation of $\mathcal{P}$, described by all the lines in matrix A and only some of the lines in matrix C. Let y be an optimal solution for $\mathcal{Q}$ obtained by our linear programming solver (which might use the simplex method), we have then two cases: (i) $y \in \mathcal{P}$ or (ii) $y \notin \mathcal{P}$.

If y is an optimal solution for $\mathcal{Q}$ and $y \in \mathcal{P}$ (i), then y must be an optimal solution for $\mathcal{P}$. On the other hand, if $y \notin \mathcal{P}$ (ii), there must exists an inequality $\pi^T x \leq \pi_0$ such that, if we add this inequality to $\mathcal{Q}$, obtaining $\mathcal{Q}'$, we would have that y does not belong to $\mathcal{Q}'$. In other words, $y \notin \mathcal{Q}'$ where $\mathcal{Q}' = \{x \in \mathbb{R}^n : x \in \mathcal{Q}, \pi^T x \leq \pi_0\}$. Thus, we can update $\mathcal{Q}$ to $\mathcal{Q}'$, and repeat the process until we arrive at case (i).

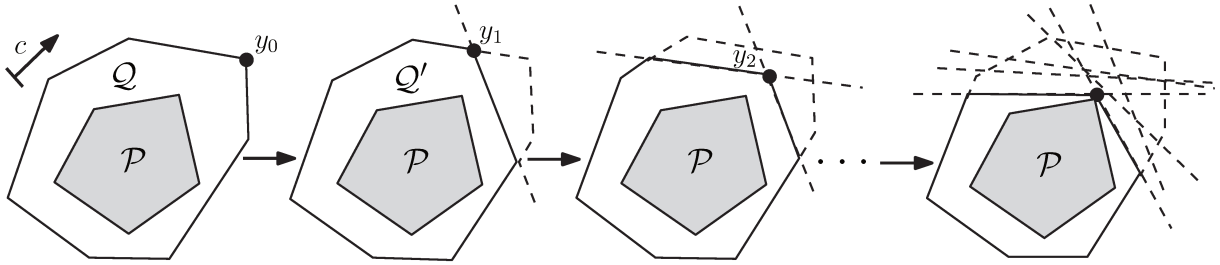


Figure 2.3: Iterations of the cutting plane method. Figure based on Miyazawa [38].

Since the inequality $\pi^T x \leq \pi_0$ *separates* y from the relaxed polyhedron $\mathcal{Q}$, we say that such inequality is a *cutting plane*. Moreover, we refer to the algorithm that finds such a cutting plane by the name of *separation routine*.

We should note here that finding a cutting plane that separates a vector that does not belong to $\mathcal{P}$ is polynomially equivalent to solving the separation problem (SEP); on the other hand, finding an optimal solution for a polyhedron $\mathcal{P}$, is polynomially equivalent to solving the optimization problem (OPT). An important result from Grötschel, Lovász and Schrijver [21] shows that, if all the considered numbers are rational, (SEP) and (OPT) have the same computational complexity. Hence, if we could solve (SEP) in polynomial time, we could also create a polynomial-time algorithm for solving (OPT) — although such an algorithm might not be of practical interest, due to its dependence on the ellipsoid method [17].

To conclude, when we combine the cutting plane method with the branch-and-bound procedure — executing the cutting plane method in each node of the branch-and-bound tree — we obtain an algorithm of type *branch-and-cut*.

2.3 Polyhedral Combinatorics

2.3.1 Linear Algebra and Polyhedral Theory

Let t be a positive integer. Given a subset $S \subseteq \mathbb{R}^n$, a vector $x \in \mathbb{R}^n$ is a *linear combination* of vectors $x^1, \ldots, x^t$ in S if $x = \sum_{i=1}^{t} \lambda_i x^i$ for some $(\lambda_1, \ldots, \lambda_t) \in \mathbb{R}^t$. Furthermore, if besides x being a linear combination, we also have that $\sum_{i=1}^{t} \lambda_i = 1$ and $(\lambda_1, \ldots, \lambda_t) \in \mathbb{R}^t_{\geq}$, x is said to be a *convex combination* of $x^1, \ldots, x^t$. The *convex hull* of S, denoted by $\text{conv}(S)$, is the set of all vectors that are a convex combination of finitely many vectors in S.

The vectors $x^1, \ldots, x^t$ are *linearly independent* (LI) if and only if $\sum_{i=1}^{t} \lambda_i x^i = 0$ implies that $\lambda_i = 0$, for all $i \in [t]$. In a similar way, the vectors $x^1, x^2, \ldots, x^t$ are called *affinely independent* (AI) if and only if $\sum_{i=1}^{t} \lambda_i x^i = 0$ and $\sum_{i=1}^{t} \lambda_i = 0$ implies that $\lambda_i = 0$, for all $i \in [t]$. Notice that linear independence implies affine independence, but the converse is not valid. The *rank* of S, denoted by $\text{rank}(S)$, is the maximum number of LI vectors in S. Similarly, the *affine-rank* of S or $\text{affine-rank}(S)$, is the maximum number of AI vectors in S. The *dimension* of S is given by $\dim(S) = \text{affine-rank}(S) - 1$, and we say that S has *full dimension* if $\dim(S) = n$.

A *halfspace* is a set of the form $\{x \in \mathbb{R}^n : a^T x \leq b\}$ for $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$. A *polyhedron* $\mathcal{P} \subseteq \mathbb{R}^n$ is the intersection of $m \in \mathbb{Z}_{\geq}$ halfspaces, i.e. $\mathcal{P} = \{x \in \mathbb{R}^n : Ax \leq b\}$, where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. If there exists a real value $\alpha \geq 0$ such that $|x_i| \leq \alpha$, for all $x \in \mathcal{P}$ and $i \in [n]$, we say that $\mathcal{P}$ is a *polytope*. A point $x \in \mathcal{P}$ is said to be a *vertex* of $\mathcal{P}$ if and only if it cannot be written as a convex combination of points in $\mathcal{P} \setminus \{x\}$.

An inequality $\pi^T x \leq \pi_0$, also denoted by (π, π_0), is said to be a *valid inequality* for $\mathcal{P}$ if it is satisfied by all vectors in $\mathcal{P}$. Consider a scalar $\alpha \in \mathbb{R}_{>}$, since the inequalities (π, π_0) and $(\alpha\pi, \alpha\pi_0)$ describe the same set, we say that they are *identical*. Let (π, π_0) and (μ, μ_0) be two valid inequalities (for $\mathcal{P}$) that are not identical, we say that (π, π_0) *dominates* (μ, μ_0) if $\{x \in R^n : \pi^T x \leq \pi_0\} \subseteq \{x \in R^n : \mu^T x \leq \mu_0\}$. A valid inequality (π, π_0) is *redundant* in the description of $\mathcal{P}$, if there exist $k \geq 2$ valid inequalities (π^i, π_0^i), for $i \in [k]$, such that $(\sum_{i \in [k]} \alpha_i \pi^i)^T x \leq (\sum_{i \in [k]} \alpha_i \pi_0^i)$ dominates $\pi^T x \leq \pi_0$, where $\alpha_i \in \mathbb{R}_{>}$, $i \in [k]$. Observe that, if (π, π_0) is redundant, the following inclusion holds

$$\{x \in R^n : (\pi^i)^T x \leq \pi_0^i, \forall i \in [k]\} \subseteq \{x \in R^n : \pi^T x \leq \pi_0\}.$$

In the context of combinatorial optimization, one of the objectives of studying polyhedral theory is on finding inequalities that are non-redundant to a polyhedron $\mathcal{P}$. If (π, π_0) is a valid inequality for $\mathcal{P}$ and $F = \{x \in \mathcal{P} : \pi^T x = \pi_0\}$, then F is said to be a *face* of $\mathcal{P}$. A face F of $\mathcal{P}$ is said to be a *facet* of $\mathcal{P}$, if $\dim(F) = \dim(\mathcal{P}) - 1$. An important result from polyhedral theory says that, if $\mathcal{P}$ is full dimensional, (π, π_0) defines a facet of $\mathcal{P}$ if and only if every set of inequalities that describe $\mathcal{P}$ contains an inequality that is identical to (π, π_0). Thus, if $\dim(\mathcal{P}) = n$, inequalities that define facets of $\mathcal{P}$ are guaranteed to be non-redundant.

Given the previous definitions, we can now show that an inequality $(\pi, \pi_0) \neq (\mathbf{0}, 0)$ defines a facet F of a full dimensional polyhedron $\mathcal{P}$ by finding n vectors $x^1, \ldots, x^n \in F$ that are AI, since then it follows that $\dim(F) = n - 1$. An equivalent way of establishing that (π, π_0) induces a facet of $\mathcal{P}$ is by selecting $t \geq n$ vectors $x^1, \ldots, x^t \in F$ and solving the system of linear equations

$$\sum_{j=1}^{n} \mu_j x_j^i = \mu_0 \qquad \forall\ i \in [t],$$

for the $n+1$ variables μ and μ_0. If the only solution for this system is of the type $\mu = \lambda\pi$ and $\mu_0 = \lambda\pi_0$ for a constant $\lambda \neq 0$, then the inequality (π, π_0) defines a facet.

2.3.2 A short example

Up to now, we only defined concepts from linear algebra and polyhedral theory. The branch of mathematics entitled *polyhedral combinatorics* applies these concepts to solve problems in combinatorics. In this section, we illustrate one of these applications.

Many combinatorial optimization problems (including $\mathcal{NP}$-hard ones) can be formulated as an integer linear programming problem $\max\{c^T x : x \in \mathcal{P} \cap \mathbb{Z}\}$, where $\mathcal{P} = \{x \in \mathbb{R}^n : Ax \leq b, x \geq \mathbf{0}\}$. Let $\mathcal{P}_I = \text{conv}(\mathcal{P} \cup \mathbb{Z})$, it is not hard to see that,

if $\mathcal{P}$ is a polytope, then

$$\max\{c^T x : x \in \mathcal{P} \cap \mathbb{Z}\} = \max\{c^T x : x \in \mathcal{P}_I\}.$$

Hence, given that we can solve a linear programming problem in polynomial time using the ellipsoid method [28], if we had the defining inequalities for $\mathcal{P}_I$, we would be able to solve efficiently many combinatorial optimization problems, including $\mathcal{NP}$-hard problems. Therefore, we usually do not expect to be easy or even feasible to obtain a complete description of $\mathcal{P}_I$. On the other hand, we can investigate if the inequalities in our formulation (represented by the linear system $Ax \leq b$) induces facets of $\mathcal{P}_I$, since as we discussed previously, these inequalities are guaranteed to be non-redundant (see Figure 2.4).

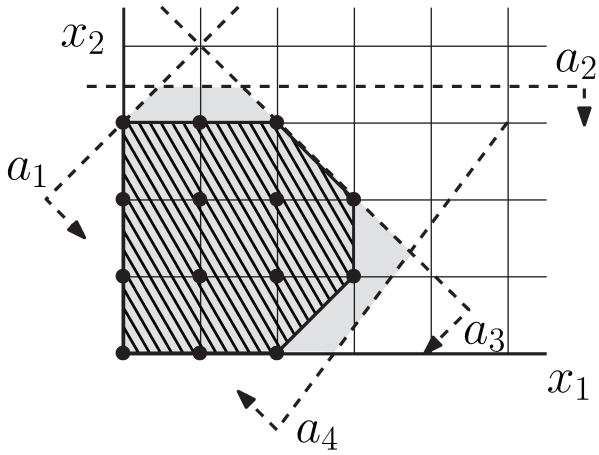


Figure 2.4: Example of $\mathcal{P}$ and $\mathcal{P}_I$ with $n = 2$. The shaded and dashed area correspond to $\mathcal{P}$ and $\mathcal{P}_I$, respectively. Note how a_3 corresponds to a facet of $\mathcal{P}_I$, but a_1 does not.

We now give a classical example (taken from Nemhauser and Wolsey [42]) that illustrates how to apply the previous ideas to a famous $\mathcal{NP}$-complete problem: the maximum independent set problem. Given an undirected graph $G = (V, E)$, an *independent set* is a set of vertices $S \subseteq V$, such that no two vertices in S are adjacent. The *maximum independent set problem* asks for an independent set of maximum cardinality.

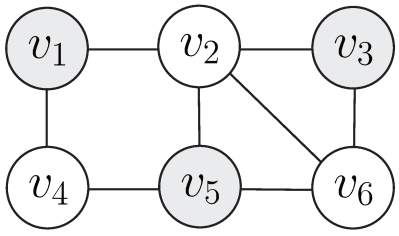


Figure 2.5: Example of a maximum independent set. The gray vertices denote vertices in the independent set.

For any vertex $v \in V$, let x_v be a binary variable that is equal to one if and only if v is in the independent set. Consider the following formulation for this problem.

$$\max \sum_{v \in V} x_v$$

$$\text{s.t. } x_u + x_v \leq 1 \qquad \forall \{u, v\} \in E, \tag{2.1}$$

$$x_v \in \{0, 1\} \qquad \forall v \in V. \tag{2.2}$$

Let $n = |V|$ and let $\mathcal{P}_S$ be the convex hull of the feasible solutions for the above formulation, i.e. $\mathcal{P}_S = \text{conv}(\{x \in \mathbb{B}^n : x \text{ satisfies } (2.1)\})$. We first show that $\mathcal{P}_S$ is full-dimensional ($\dim(\mathcal{P}_S) = n$). Next, we derive a valid inequality that has dimension $n-1$, and therefore, defines a facet of $\mathcal{P}_S$.

Proposition 2.3.1 $\dim(\mathcal{P}_S) = n$.

Proof. Let $V = \{v_1, \ldots, v_n\}$ and consider the set $S_i = \{v_i\}$, for $i \in [n]$. Clearly S_i is an independent set, since it contains exactly one vertex. Furthermore, it is trivial that the empty set is also an independent set. For any $i \in [n]$, define e_i as being a vector where only its i-th coordinate is one and all the other coordinates are zeroes. Because of our interpretation for variables x, e_i corresponds to the solution S_i. In this sense, we say that e_i is the *incidence vector* of S_i. Moreover, $\mathbf{0}$ is the incidence vector of the empty set. Thus, we found $n+1$ vectors in $\mathcal{P}_S$, it remains to show that these vectors are affinely independent.

Consider the matrix M whose columns are given by $e_1, \ldots, e_n$, in this order. We represent this matrix by $M = [e_1, \ldots, e_n]$. Note that M is an identity matrix with dimension n. Hence, the vectors $\{e_i\}_{i \in [n]}$ are linearly independent and together with the null vector we found $n+1$ affinely independent vectors that belong to $\mathcal{P}_S$. □

Let us break down the previous proof. In order to enumerate the desired number of affinely independent vectors, we first constructed feasible combinatorial solutions for the original problem. These were very simple solutions: an empty set and sets with exactly one vertex. Next, we transformed the solutions into incidence vectors and analyzed the geometrical properties of these vectors. In our case, each vector was parallel to a different axis of the standard coordinate system, so it was easy to argue that they were linearly independent. This type of reasoning, which explores the relationship between properties of combinatorial solutions and geometrical aspects of its incidence vectors, is commonly used in polyhedral combinatorics.

Before we proceed, let us introduce a new concept. A *clique* $C \subseteq V$ is a set of vertices such that each pair of nodes in C is connected by an edge. We say that a clique C is *maximal* if there is no clique C' in G such that $C \subset C'$. In the following proposition, we use the notion of a maximal clique to derive a new facet-defining inequality. Usually, when one proposes new inequalities, one first has to show that these inequalities are valid. However, in this case, the validity is trivial and we omit the details.

Proposition 2.3.2 *Let C be a maximal clique of G, then the inequality $\sum_{v \in C} x_v \leq 1$ defines a facet of $\mathcal{P}_S$.*

Proof. Let F be the face associated with the above inequality, hence

$$F = \left\{ x \in \mathcal{P}_S : \sum_{v \in C} x_v = 1 \right\}.$$

We need to show that the dimension of F is $n - 1$.

Without loss of generality we assume that $V = \{v_1, \ldots, v_n\}$ and $C = \{v_1, \ldots, v_k\}$. First we observe that, for any vertex v_p, with $p \in [n] \setminus [k]$, there exists a vertex $\ell(v_p) \in C$ such that $\{\ell(v_p), v_p\} \notin E$ (if there was no such vertex, the clique would not be maximal). Let α_p be the incidence vector of the independent set $\{\ell(v_p), v_p\}$, for all $p \in [n] \setminus [k]$. Notice that $(\{e_1, \ldots, e_k\} \cup \{\alpha_{k+1}, \ldots, \alpha_n\})$ is contained in F. Now we need to show that these vectors are affinely independent.

Consider matrix $M = [e_1, \ldots, e_k, \alpha_{k+1}, \ldots, \alpha_n]$ and observe that M is an upper triangular matrix. Therefore, all of its columns are linearly independent vectors. Since M has n columns, $\dim(F) = n - 1$. □

Polyhedral studies of combinatorial optimization problems led to major achievements in the area. The strong inequalities derived for the travelling salesman problem motivated a very efficient branch-and-cut algorithm. This algorithm solved instances with around dozens of thousands of vertices [3], while before using polyhedral techniques, the algorithms were able to solve only up to 120 vertices. Although more convoluted, the polyhedral study we conduct on Section 5.4 uses arguments that are similar to the ones presented in this section.

Chapter 3

Known Results

Problems regarding the partition of vertex-weighted graphs into connected subgraphs with similar weights have been largely investigated in the literature since the early eighties. Such partitions are generally called balanced, and several different functions have been considered to measure this feature; such as minimize the weight of the heaviest class, or minimize the maximum difference of weights between the classes. The balanced connected k-partition problem, which is the focus here, is one of those problems. It is closely related to another problem, referred to as MIN-MAX BCP$_k$, whose objective function is to minimize the weight of the heaviest class. When $k = 2$, for any instance, an optimal 2-partition for BCP$_k$ is also an optimal solution for MIN-MAX BCP$_k$. However, when $k > 2$ the corresponding optimal connected k-partitions may differ (Figure 3.1).

3.1 Hardness and Approximation

The *unweighted* BCP$_k$ (to be denoted by 1-BCP$_k$) is the restricted version of BCP$_k$ in which all the vertices have unit weight. This restricted problem is $\mathcal{NP}$-hard on bipartite graphs for every fixed $k \geq 2$, as proven by Dyer and Frieze [15]. Chlebíková [12] showed that 1-BCP$_2$ is $\mathcal{NP}$-hard to approximate within an absolute error guarantee of $n^{1-\varepsilon}$, for all $\varepsilon > 0$, where $n = |V|$. For the weighted case, Becker, Lari, Lucertini and Simeone [6] proved that BCP$_2$ is already $\mathcal{NP}$-hard on (nontrivial) grid graphs. Chataigner, Salgado, and Wakabayashi [10] showed that, for each $k \geq 2$, BCP$_k$ is $\mathcal{NP}$-hard in the strong sense, even on k-connected graphs, and therefore does not admit an FPTAS, unless $\mathcal{P} = \mathcal{NP}$. Wu [49] observed that BCP$_k$ is $\mathcal{NP}$-hard on interval graphs for any fixed $k \geq 2$.

Before we proceed with the approximation results for BCP$_k$ and its variants, we first need to establish the used notation. Consider an algorithm A for an optimization problem

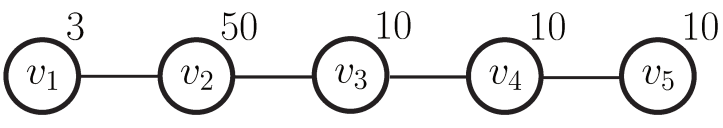


Figure 3.1: Example adapted from Lucertini, Perl, and Simeone [31]. For $k = 3$, note that $\{\{v_1\}, \{v_2\}, \{v_3, v_4, v_5\}\}$ is an optimal solution for MIN-MAX BCP$_k$ but it is not for BCP$_k$.

and let I be a feasible instance of such a problem. We denote by $A(I)$ the cost of the solution returned by A on the instance I. Moreover, we use $\text{opt}(I)$ to refer to the cost of an optimal solution for the instance I. Let $\alpha \geq 1$, the algorithm A is said to be an α-approximation for a minimization problem if, for any feasible instance I, it holds that $A(I) \leq \alpha\ \text{opt}(I)$. Similarly, A is an α-approximation for a maximization problem if $A(I) \geq \frac{1}{\alpha}\ \text{opt}(I)$, for every feasible instance I.

Chlebíková [12] designed a 4/3-approximation algorithm for BCP_2. For BCP_3 (resp. BCP_4) on 3-connected (resp. 4-connected) graphs, algorithms with approximation ratio 2 were proposed by Chataigner et al. [10]. The case $k = 3$ on general graphs was recently investigated by Chen, Chen, Chen, Lin, Liu, and Zhang [11]. They presented a 5/3-approximation for BCP_3 and a 3/2-approximation for MIN-MAX BCP_3. Wu [49] designed a fully polynomial-time approximation scheme (FPTAS) for BCP_2 on interval graphs. When k is part of the input, Borndörfer, Elijazyfer and Schwartz [9] designed Δ-approximation algorithms for both MAX-MIN and MIN-MAX versions of the balanced connected partition problem, where Δ is the maximum degree of an arbitrary spanning tree of the input graph G. Specifically for the MAX-MIN version, their Δ-approximation only holds for instances in which the largest weight is at most $w(G)/(\Delta\, k)$.

Both BCP_k and MIN-MAX BCP_k can be solved in linear time on trees as shown by Frederickson [18]. One may easily check that 1-BCP_2 on 2-connected graphs can be solved in polynomial time. This problem also admits polynomial-time algorithms on graphs such that each block has at most two articulation vertices [1, 12]. In special, when the input graph is k-connected, polynomial-time algorithms and other interesting structural results have been obtained for BCP_k by Ma and Ma [32], Györi [24], and Lovász [29]. Many other results on the mentioned problems and variants have appeared in the literature [2, 41, 44, 46].

Mixed integer linear programming formulations for BCP_2 were proposed by Matić [37] and for MIN-MAX BCP_k by Zhou, Wang, Ding, Hu, and Shang [50]. Matić also presented a VNS-based heuristic for BCP_2, and Zhou et al. devised a genetic algorithm for MIN-MAX BCP_k. Both works reported on the computational results obtained with the proposed formulations and heuristics, but presented no polyhedral study.

3.2 A note on Minimum Spanning Trees

Matić and Zhou et al. formulations use flows to guarantee that each class is connected. Such an idea is very common in the literature and can be easily described in the context of the minimum spanning tree (MST) problem. In a sense, BCP_k and MST's are loosely related, since checking if a subgraph is connected is equivalent to checking if the subgraph contains a spanning tree. For any $V' \subseteq V$, $G[V']$ is connected if and only if there is a tree $T \subseteq G[V']$ such that T spans the set of vertices V'. Therefore, for any solution $\{V_i\}_{i\in[k]}$ for BCP_k, there is a corresponding forest $\{T_i\}_{i\in[k]}$, such that T_i spans V_i.

The forthcoming formulations come from the excellent chapter written by Magnanti and Wolsey [33]. Recall that a spanning tree of G is a subgraph that is a tree and spans all the vertices in G. Given a connected graph $G = (V, E)$ with weights on the edges

given by a function $w\colon E \mapsto \mathbb{Q}_{\geq}$, the minimum spanning tree problem asks for a spanning tree T such that $w(T) = \sum_{e \in E(T)} w(e)$ is minimum.

When formulating problems that aim to find a tree, we usually have a binary variable x_e, for each edge $e \in E$, that indicates if the edge belongs to the tree. Thus, it is important to consider how to guarantee that the graph induced by the x variables is connected. In this section, we present two formulations that handle this requirement in different ways. The first model is more natural, but needs a large number of inequalities; while the second formulation is compact, but introduces new flow variables.

Below is an ILP formulation with a possibly exponential number of constraints. To simplify, for any set of edges $E' \subseteq E$, we use the notation $x(E') = \sum_{e \in E'} x_e$.

$$
\begin{aligned}
\min \quad & \sum_{e \in E} w(e)\, x_e \\
\text{s.t.} \quad & x(E) = n - 1, && && (3.1)\\
& x(\delta(S)) \geq 1 && \forall S \subset V, S \neq \emptyset, && (3.2)\\
& x_e \in \{0, 1\} && \forall e \in E. && (3.3)
\end{aligned}
$$

Constraint (3.1) implies that exactly $n - 1$ edges are going to be chosen. Inequalities (3.2) guarantee that the graph induced by the x variables is connected. Overall, the formulation above is very simple, it just states that a spanning tree is a connected subgraph with $n - 1$ edges.

Another way of ensuring a solution is connected is to choose a vertex $s \in V$ to be a source, and send flow from s to every other vertex in the graph. Since the flow has a direction — for any edge $\{u, v\}$, it either goes from u to v or from v to u — we create a digraph D from the original graph G. The set of vertices of D is the same as G, and for each edge in G, we add two anti-parallel arcs to D; that is, $A(D) = \{(u, v), (v, u) \colon \{u, v\} \in E\}$.

An arc $(u, v) \in A(D)$ will be allowed to have flow passing through it only if the correspondent edge $e = \{u, v\} \in E$ was chosen by the x_e variable. Moreover, each vertex in $V \setminus \{s\}$ will not produce any flow and consume exactly one unit of it. Thus, every vertex must be connected to s, and thereby, the subgraph induced by the x variables must be connected. Henceforth, we refer to an arc (u, v) simply as uv; and we denote by $f(A')$ the sum of variables $\sum_{a \in A'} f_a$, where $A' \subseteq A(D)$.

$$
\begin{aligned}
\min \quad & \sum_{e \in E} w(e)\, x_e \\
\text{s.t.} \quad & x(E) = n - 1, && && (3.1)\\
& f(\delta^+(s)) - f(\delta^-(s)) = n - 1, && && (3.4)\\
& f(\delta^-(v)) - f(\delta^+(v)) = 1 && \forall v \in V \setminus \{s\}, && (3.5)\\
& f_{uv} \leq (n-1) x_e && \forall e = \{u, v\} \in E, && (3.6)\\
& f_{vu} \leq (n-1) x_e && \forall e = \{u, v\} \in E, && (3.7)\\
& f_a \geq 0 && \forall a \in A(D), && (3.8)\\
& x_e \in \{0, 1\} && \forall e \in E. && (3.9)
\end{aligned}
$$

Constraint (3.1) is the same as in the previous model. Equality (3.4) guarantees that the source outputs $n-1$ units of flow. Constraints (3.5) ensure that each vertex consumes exactly one unit of flow. Finally, inequalities (3.6) and (3.7) imply that a positive flow will pass only through arcs that were selected by the correspondent x variables. This formulation has a polynomial number of constraints, but it also introduces new variables.

Although both of the formulations presented correctly find an MST; the polytope associated with their linear relaxations can be quite larger than the convex hull of incidence vectors of spanning trees. Magnanti and Wolsey [33] present further ideas to strengthen these formulations. To avoid deviating from the main topic of this thesis, we refer the reader to their work for more details.

3.3 MILP formulations

We now present the formulations of Matić [37] and Zhou et al. [50]. These models are all based on the idea of using additional flow variables to guarantee the connectedness of the classes. To the best of our knowledge, these are the only formulations in the literature for variants of the BCP_k problem. For presentation purposes, we define r_i as being the root (or representative) of class V_i.

Formulation of Matić

The formulation proposed by Matić was designed for MIN-MAX BCP_2, which, as we stated earlier, is equivalent to BCP_2. The model construct a directed graph D_M from the original graph G in the following (non-standard) way:

(i). $V(D_M) = V \cup \{s\}$, where s is a vertex that represents a source;

(ii). $A(D_M) = \{(u,v) : \{u,v\} \in E\} \cup \{(s,v) : v \in V\}$, note that we arbitrarily fixed an orientation for each edge in the original graph.

Due to how we are fixing the orientation of the arcs in the digraph D_M, when talking about Matić formulation, we speak of trees — ignoring the direction of the arcs — instead of arborescences. Let T_i be a tree that spans V_i, for $i \in \{1,2\}$. Let $T_i' = T_i \cup (s, r_i)$. The formulation variables are as follows:

- x_v: if $x_v = 1$, vertex v belongs to V_1, otherwise, $v \in V_2$,
- y_a: indicates if $a \in A(T_1')$,
- z_a: indicates if $a \in A(T_2')$,
- f_a: indicates the flow on arc a, if the flow is negative, it is going contrary to the arc direction.

For any vertex $v \in V(D_M)$, the flow variables should obey the following equations

$$\sum_{a \in \delta^+(v)} f_a = \begin{cases} n, & v = s, \\ \left(\sum_{a \in \delta^-(v)} f_a\right) - 1, & \text{otherwise}, \end{cases} \tag{3.10}$$

which is very similar to how we used the flow variables for obtaining spanning trees. A source vertex outputs enough flow for every other vertex, and each vertex that is not a source consumes one unit of it. The following is the formulation proposed by Matić. Again, for $A' \subseteq A(D_M)$, we use shorthand notations of the type $f(A') = \sum_{a \in A'} f_a$. This notation is also used for the summation of y and z variables.

$$\min \; -w(G) + 2\sum_{v \in V} w(v)\; x_v$$

$$\text{s.t. } 2\sum_{v \in V} w(v)\; x_v \geq w(G), \tag{3.11}$$

$$2y_{uv} \leq x_u + x_v \qquad \forall (u,v) \in A(D_M) \backslash \delta^+(s), \tag{3.12}$$

$$2z_{uv} \leq 2 - x_u - x_v \qquad \forall (u,v) \in A(D_M) \backslash \delta^+(s), \tag{3.13}$$

$$y_{sv} \leq x_v \qquad \forall (s,v) \in \delta^+(s), \tag{3.14}$$

$$z_{sv} \leq 1 - x_v \qquad \forall (s,v) \in \delta^+(s), \tag{3.15}$$

$$f_a \leq n y_a + n z_a \qquad \forall a \in A(D_M), \tag{3.16}$$

$$f_a \geq -n y_a - n z_a \qquad \forall a \in A(D_M), \tag{3.17}$$

$$f(\delta^-(v)) - f(\delta^+(v)) = 1 \qquad \forall v \in V, \tag{3.18}$$

$$f(\delta^+(s)) = n, \tag{3.19}$$

$$y(A(D_M) \setminus \delta^+(s)) + z(A(D_M) \setminus \delta^+(s)) = n - 2, \tag{3.20}$$

$$y(\delta^+(s)) + z(\delta^+(s)) = 2, \tag{3.21}$$

$$x_v, \in \{0,1\} \qquad \forall v \in V, \tag{3.22}$$

$$y_a, z_a \in \{0,1\} \qquad \forall a \in A(D_M), \tag{3.23}$$

$$f_a \in \mathbb{R} \qquad \forall a \in A(D_M). \tag{3.24}$$

Constraint (3.11) guarantees that V_1 is the heaviest class. Constraints (3.12) indicate that if $a = (u,v) \in A(T_1')$, then vertices u and v are in V_1; similarly, constraints (3.13) indicate that if $a = (u,v) \in A(T_2')$, then u and v are not in V_1, and thus, $u, v \in V_2$. Constraints (3.14) indicate that if an arc $(s,v) \in A(T_1')$, then $v \in V_1$; and constraints (3.15) similarly do the same when $v \in V_2$. Inequalities (3.16) and (3.17) attribute upper and lower bounds for the flow that passes through an arc $a \in A(D_M)$, in other words, if $a \in T_1'$ or $a \in T_2'$, then $f_a \in [-n, n]$; otherwise, $f_a = 0$. Constraints (3.18) and (3.19) attribute the flow according to the equations (3.10). Inequality (3.20) indicates that the total number of arcs in the graph $T_1 \cup T_2$ must be $n-2$. Finally, constraints (3.21) impose that exactly 2 arcs that outgoes s must be selected.

Let $m = |E|$ and $n = |V|$, the formulation proposed by Matić has $2m + 3n$ binary variables, $m + n$ real variables and $4m + 5n + 4$ constraints. In his paper [37], Matić reports that, with a time limit of 2 hours, this formulation could only find provably optimal solutions in graphs with up to 70 vertices.

Formulation of Zhou et al.

The formulation proposed by Zhou et al. [51] solves MIN-MAX BCP_k with $k \geq 2$. Although not explicitly stated in their paper, this formulation is also based on a digraph D_Z defined over the input graph $G = (V, E)$; with the difference that it creates two anti-parallel arcs for each edge in E (the standard way). Formally,

(i). $V(D_Z) = V \cup \{s\}$, where s is a vertex that represents a source,

(ii). $A(D_Z) = \{(u, v), (v, u) : \{u, v\} \in E\} \cup \{(s, v) : v \in V\}$.

The following variables are used by the model:

- $x_{v,i}$: indicates if vertex $v \in V_i$,
- $r_{v,i}$: indicates if $v = r_i$,
- $y_{uv,i}$: if $(u, v) \in A(D_Z) \setminus \delta^+(s)$, this variable indicates if $u, v \in V_i$, otherwise, it indicates if $v \in V_i$,
- f_a: indicates the flow on arc a, its value is not allowed to be negative.

Below is the formulation for MIN-MAX BCP_k proposed by Zhou et al.

$$\min \ t$$

$$\text{s.t. } t \geq \sum_{v \in V(G)} w(v)\ x_{v,i} \qquad \forall i \in [k], \tag{3.25}$$

$$\sum_{i=1}^{k} x_{v,i} = 1 \qquad \forall v \in V, \tag{3.26}$$

$$r_{v,i} \leq x_{v,i} \qquad \forall v \in V, i \in [k], \tag{3.27}$$

$$n\ r_{v,i} \leq (n + 1 - \sum_{u=1}^{v} x_{u,i}) \qquad \forall v \in V, i \in [k], \tag{3.28}$$

$$f(\delta^+(s)) = n, \tag{3.29}$$

$$f(\delta^-(v)) - f(\delta^+(v)) = 1 \qquad \forall v \in V(G), \tag{3.30}$$

$$f_{sv} \leq n \sum_{i=1}^{k} r_{v,i} \qquad \forall (s, v) \in \delta^+(s), \tag{3.31}$$

$$f_{uv} \leq n \sum_{i=1}^{k} y_{uv,i} \qquad \forall (u, v) \in A(D_Z) \setminus \delta^+(s), \tag{3.32}$$

$$y_{uv,i} \leq x_{u,i} \qquad \forall (u, v) \in A(D_Z) \setminus \delta^+(s), i \in [k], \tag{3.33}$$

$$y_{uv,i} \leq x_{v,i} \qquad \forall (u, v) \in A(D_Z) \setminus \delta^+(s), i \in [k], \tag{3.34}$$

$$y_{uv,i} \geq x_{u,i} + x_{v,i} - 1 \qquad \forall (u, v) \in A(D_Z) \setminus \delta^+(s), i \in [k], \tag{3.35}$$

$$x_{v,i}, r_{v,i} \in \{0, 1\} \qquad \forall v \in V, i \in [k], \tag{3.36}$$

$$y_{a,i} \in \{0, 1\} \qquad \forall a \in A(D_Z) \setminus \delta^+(s), i \in [k], \tag{3.37}$$

$$f_a \in \mathbb{R}_{\geq} \qquad \forall a \in A(D_Z), \tag{3.38}$$

$$t \in \mathbb{R}_{\geq}. \tag{3.39}$$

Just as in Matić formulation, the source s sends n units of flow, and each vertex in V consume one unit. Furthermore, we assume the set of vertices in V are labeled from 1 to n.

Constraints (3.25) force the objective function to minimize the weight of the heaviest class. Constraint (3.26) means that each vertex must belong to exactly one class. For each class $i \in [k]$ and vertex $v \in V$, inequalities (3.27) indicate that vertex v can be the root r_i for class i only if v belongs to V_i. Furthermore, constraints (3.28) imply that r_i is the vertex in V_i with minimum label. Constraints (3.29) force the source s to output n units of flow, while constraints (3.30) impose that each vertex $v \in V$ consumes one unit of flow. For every arc $(s, v) \in \delta^+(s)$, inequalities (3.31) say that the flow f_{sv} can only be non-zero if $v = r_i$. Similarly, for every arc $a = (u, v) \notin \delta^+(s)$, constraints (3.32) state that $f_a > 0$ implies that u and v belong to the same class. Finally, constraints (3.33), (3.34) and (3.35) tie together the x variables with the y variables.

This model has $2nk + 2mk$ binary variables, $1 + 2m + n$ real variables and $O(mk)$ constraints. Computational results by Zhou et al. show that this formulation is considerably faster than the one proposed by Matić, and it was able to solve instances for BCP_2 with up to 170 vertices within the time limit of 2 hours. Moreover, when $k > 2$, they conducted experiments only in a single graph with 70 vertices. In one hour, they were able to solve this single instance for $k = 2, 3, 4, 5$, but failed when $k = 6$.

Chapter 4

Flow Formulations

Building on the concept of using flow variables for obtaining k disjoint trees, we designed two mixed integer linear programming formulations for BCP_k. Let (G, w) be an input for BCP_k, we denote these formulations by $\mathcal{F}_k(G, w)$ and $\bar{\mathcal{F}}_k(G, w)$. Sometimes, we refer to them simply by $\mathcal{F}$ and $\bar{\mathcal{F}}$, respectively.

4.1 Flow Formulation

In order to define formulation $\mathcal{F}_k(G, w)$ we construct a digraph $D_{\mathcal{F}}$ from the input graph $G = (V, E)$ as follows. First, we add to G a set of k new vertices $S = \{s_1, \ldots, s_k\}$. Each vertex in S represents a source of flow. Second, we replace every edge of G with two anti-parallel arcs. Finally, we add an arc from each source to each vertex in V (see Figure 4.1(a)). More formally, the vertex set of $D_{\mathcal{F}}$ is $V(D_{\mathcal{F}}) = V \cup S$ and its arc set is

$$A(D_{\mathcal{F}}) = \{(u, v), (v, u) : \{u, v\} \in E\} \cup \{(s_i, v) : i \in [k], v \in V\}.$$

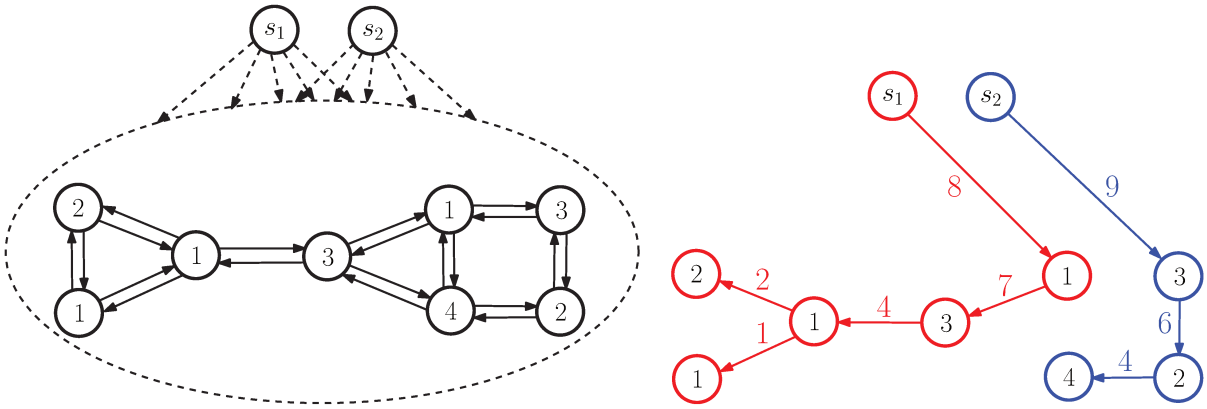
Let $\{\overrightarrow{T_i}\}_{i\in[k]}$ be a set of arborescences such that $\{V(\overrightarrow{T_i})\}_{i\in[k]}$ is a connected k-partition of G. We are now ready to present the formulation variables:

- y_a: indicates if an arc $a \in A(D)$ belongs to $A(\overrightarrow{T_i}) \cup \{(s_i, r_i)\}$, for some $i \in [k]$;
- f_a: represents the flow on arc a, each vertex v will consume $w(v)$ units of flow.

The corresponding formulation is shown below.

$$\begin{aligned}
\mathcal{F}_k(G, w) \qquad & \max f(\delta^+(s_1)) \\
& \text{s.t. } f(\delta^+(s_i)) \le f(\delta^+(s_{i+1})) && \forall i \in [k-1], && (4.1)\\
& f(\delta^-(v)) - f(\delta^+(v)) = w(v) && \forall v \in V, && (4.2)\\
& f_a \le w(G) y_a && \forall a \in A(D_{\mathcal{F}}), && (4.3)\\
& y(\delta^+(s_i)) \le 1 && \forall i \in [k], && (4.4)\\
& y(\delta^-(v)) \le 1 && \forall v \in V, && (4.5)\\
& y_a \in \{0, 1\} && \forall a \in A(D_{\mathcal{F}}), && (4.6)\\
& f_a \in \mathbb{R}_{\ge} && \forall a \in A(D_{\mathcal{F}}). && (4.7)
\end{aligned}$$

Constraints (4.2) imply that each vertex $v \in V$ consumes $w(v)$ flow units. Inequalities (4.3) impose that a strictly positive flow can only pass through an arc that was selected by the y variables. By constraints (4.4), at most one arc leaving a source transports positive flow. Inequalities (4.5) require that every non-source vertex receives a positive flow from at most one vertex of $D_{\mathcal{F}}$. Lastly, inequalities (4.1) impose that the flows sent by the sources are in a non-decreasing order. This explains the objective function.



(a) The digraph $D_{\mathcal{F}}$ obtained from the graph G shown in Figure 1.1. Vertices s_1 and s_2 dominates all vertices in the dashed circle. The numbers inside the vertices are the weights.

(b) A feasible solution: the arcs represent non-zero y-variables, and the flow in each of them is indicated on their side.

Figure 4.1: Digraph $D_{\mathcal{F}}$ and a feasile solution for formulation $\mathcal{F}_2$.

Since the flow sent by a source needs to be totally consumed, it follows that the flow sent through an arc (s_i, r_i) corresponds exactly to the weight of the i-th class (see Figure 4.1(b)).

In a feasible solution, vertices with weight zero may not receive any flow, and thus they may not belong to any of the k arborescences. If this happens, each such a vertex can be added to one of the classes found by the formulation (including first those at distance 1 to one of the classes, then the remaining ones with the same procedure w.r.t. the connected classes that are obtained). This inclusion leads to a solution that defines a connected k-partition as desired, without increasing the weight of each class. It follows from these remarks that a solution obtained with formulation $\mathcal{F}_k(G, w)$ leads to a solution of BCP$_k$.

The proposed formulation $\mathcal{F}_k(G, w)$ has a total of $2nk + 4m$ variables (half of them binary), and only $\mathcal{O}(n + m + k)$ constraints, where $n = |V|$ and $m = |E|$. The possible drawbacks of this formulation are the large number of symmetric solutions and the dependency of inequalities (4.3) on the weights assigned to the vertices. To overcome such disadvantages, we propose in the next section another model based on multicommodity flows; it considers a total order of the vertices to avoid symmetries and uncouple the weights assigned to the vertices from the flow circulating in the digraph.

4.2 Asymmetric Flow Formulation

Our second formulation for BCP_k, denoted by $\bar{\mathcal{F}}_k(G,w)$, is also based on a digraph, which we denote by $D_{\bar{\mathcal{F}}}$. This digraph has vertex set $V(D_{\bar{\mathcal{F}}}) = V \cup \{s\}$ and arc set

$$A(D_{\bar{\mathcal{F}}}) = \{(u,v),(v,u)\colon \{u,v\} \in E\} \cup \{(s,v)\colon v \in V\}.$$

As we mentioned, we assume there is a total ordering $\succ$ defined on the vertices of G.

In this formulation, we consider flows of *type* i corresponding to the classes $i \in [k]$ that they will define. For each $a \in A(D_{\bar{\mathcal{F}}})$ and $i \in [k]$, we create a real variable $f_{a,i}$ that corresponds to the amount of flow of type i that passes through arc a. Furthermore, we use binary variables $y_{a,i}$ to indicate if arc $a \in A(D_{\bar{\mathcal{F}}})$ belongs to $A(\overrightarrow{T_i}) \cup \{(s,r_i)\}$. The flow variable $f_{a,i}$ can only assume positive values if $y_{a,i} = 1$. The root r_i will receive $|V_i|$ units of flow and each vertex in V will consume a single unit of flow. The ordering of the vertices imposes that, among the vertices of each arborescence, the root is the smallest one (this helps breaking symmetries).

For simplicity, for any $A' \subseteq A(D_{\bar{\mathcal{F}}})$ and integer $i \in [k]$, we use the short notation $y(A',i)$ to replace the sum $\sum_{a\in A'} y_{a,i}$. We use similar notation for the summation of flow variables.

$$
\begin{aligned}
\bar{\mathcal{F}}_k(G,w) \qquad \max \quad & \sum_{v\in V(D_{\mathcal{F}'})} w(v)\, y(\delta^-(v),1) & & \\
\text{s.t.} \quad & \sum_{v\in V} w(v)\, y(\delta^-(v),i) \le \sum_{v\in V} w(v)\, y(\delta^-(v),i+1) & \forall i \in [k-1], \quad & (4.8)\\
& y(\delta^+(s),i) \le 1 & \forall i \in [k], \quad & (4.9)\\
& \sum_{i\in[k]} y(\delta^-(v),i) \le 1 & \forall v \in V, \quad & (4.10)\\
& y_{sv,i} + y(\delta^-(u),i) \le 1 & \forall u,v \in V,\, v \succ u,\, i \in [k], \quad & (4.11)\\
& f_{a,i} \le n\, y_{a,i} & \forall a \in A(D),\, i \in [k], \quad & (4.12)\\
& f(\delta^+(v),i) \le f(\delta^-(v),i) & \forall v \in V,\, i \in [k], \quad & (4.13)\\
& \sum_{i\in[k]} f(\delta^-(v),i) - \sum_{i\in[k]} f(\delta^+(v),i) = 1 & \forall v \in V, \quad & (4.14)\\
& y_{a,i} \in \{0,1\} & \forall a \in A(D),\, i \in [k], \quad & (4.15)\\
& f_{a,i} \in \mathbb{R}_{\ge} & \forall a \in A(D),\, i \in [k]. \quad & (4.16)
\end{aligned}
$$

To show that the above formulation indeed models BCP_k correctly, let us consider the following polytope.

$$\mathcal{Q}_k(G,w) = \mathrm{conv}(\{(y,f) \in \mathbb{B}^{(n+2m)k} \times \mathbb{R}^{(n+2m)k}\colon (y,f) \text{ satisfies ineq. } (4.8)-(4.16)\}).$$

Let $\mathcal{V} = \{V_i\}_{i\in[k]}$ be a connected k-subpartition of G such that $w(V_i) \le w(V_{i+1})$ for all $i \in [k-1]$. Then, for each integer $i \in [k]$, there exists in $D_{\bar{\mathcal{F}}}$ an arborescence $\overrightarrow{T_i}$

rooted at r_i such that $V(\overrightarrow{T_i}) = V_i$ and $v \succ r_i$ for all $v \in V_i \setminus \{r_i\}$. Now, let g_i be the function $g_i \colon A(\overrightarrow{T_i}) \cup \{(s, r_i)\} \to \mathbb{R}_{\geq}$ defined as follows:

$$g_i((u,v)) = \begin{cases} 1, & \text{if } v \text{ is a leaf of } \overrightarrow{T_i}, \\ 1 + \sum_{(v,z) \in A(\overrightarrow{T_i})} g_i((v,z)), & \text{otherwise.} \end{cases}$$

It follows from this definition that $g_i((s, r_i)) = |V_i|$.

We now define vectors $\rho(\mathcal{V}) \in \mathbb{B}^{(n+2m)k}$ and $\tau(\mathcal{V}) \in \mathbb{R}^{(n+2m)k}$ such that, for every arc $a \in A(D_{\bar{\mathcal{F}}})$ and $i \in [k]$, we have

$$\rho(\mathcal{V})_{a,i} = \begin{cases} 1, & \text{if } a \in A(\overrightarrow{T_i}) \cup \{(s, r_i)\}, \\ 0, & \text{otherwise,} \end{cases} \quad \tau(\mathcal{V})_{a,i} = \begin{cases} g_i(a), & \text{if } a \in A(\overrightarrow{T_i}) \cup \{(s, r_i)\}, \\ 0, & \text{otherwise.} \end{cases}$$

We are now ready to prove the claimed statement on $\mathcal{Q}_k(G, w)$.

Proposition 4.2.1 *The polytope $\mathcal{Q}_k(G, w)$ is precisely the polytope*

$$\operatorname{conv}(\{(\rho(\mathcal{V}), \tau(\mathcal{V})) \in \mathbb{B}^{(n+2m)k} \times \mathbb{R}^{(n+2m)k} \colon \mathcal{V} \text{ is a connected } k\text{-partition of } G\}).$$

Proof. Let (y, f) be an extreme point of $\mathcal{Q}_k(G, w)$. For every $i \in [k]$, define the set $U_i = \{v \in V \colon y(\delta^-(v), i) = 1\}$. It follows from inequalities (4.10) that, for every vertex $v \in V$, at most one of the arcs entering it is chosen. Observe that inequalities (4.12) force that a flow of type i can only pass through an arc of type i if this arc is chosen. Hence, every vertex receives at most one type of flow from its in-neighbors. Furthermore, inequalities (4.13) and (4.14) guarantee that the flow that enters a vertex and leaves it are of the same type, and that each vertex consumes exactly one unit of such flow. Inequalities (4.9) imply that all flow of a given type passes through at most one arc that has a tail at the source s. Therefore, we have that $\{U_i\}_{i \in [k]}$ is a connected k-partition of G.

To prove the converse, let $\mathcal{V} = \{V_i\}_{i \in [k]}$ be a connected k-partition of G. We assume without loss of generality that $w(V_i) \leq w(V_{i+1})$ for all $i \in [k-1]$. Let (y, f) be a vector such that $y = \rho(\mathcal{V})$ and $f = \tau(\mathcal{V})$. For each $i \in [k]$, every vertex in $\overrightarrow{T_i}$ has in-degree at most one, and r_i is the smallest vertex in $V(\overrightarrow{T_i})$ with respect to the order $\succ$. Thus, inequalities (4.10) and (4.11) hold for (y, f). From the definition of $\rho(\mathcal{V})$, the entry of y indexed by (s, r_i) and i equals one, for all $i \in [k]$. Consequently, (y, f) also satisfies inequalities (4.9). Recall that $g_i((s, r_i)) = |V_i|$ for every $i \in [k]$. This clearly implies that inequalities (4.12) are satisfied by (y, f).

Note that, for every $i \in [k]$, the function g_i assigns to each arc $(u, v) \in A(\overrightarrow{T_i}) \cup \{(s, r_i)\}$ the value one plus the sum of the sizes of the sub-arborescences of $\overrightarrow{T_i}$ rooted at the out-neighbors of v in $\overrightarrow{T_i}$. Hence, inequalities (4.13) and (4.14) hold for (y, f). Finally, inequalities (4.8) are satisfied, as we assumed that the elements of partition $\mathcal{V}$ are in a non-decreasing order of weights. Therefore, we conclude that (y, f) belongs to $\mathcal{Q}_k(G, w)$. □

4.3 Relaxed formulations

Formulations $\mathcal{F}$ and $\bar{\mathcal{F}}$ order the classes of a solution by their weights and then the objective function maximizes the weight of the first class. Quite trivially, we can create similar formulations $\mathcal{F}'$ and $\bar{\mathcal{F}}'$ in a way that they do not impose such an ordering. In these new formulations, we change the objective function to maximize a variable $z \geq 0$. For $\mathcal{F}'$ we remove constraints (4.1) and add constraints $z \leq f(\delta^+(s_i))$, for every $i \in [k]$. Similarly, we construct formulation $\bar{\mathcal{F}}'$ by removing inequalities (4.8) and adding the inequalities $z \leq \sum_{v \in V} w(v)\, y(\delta^-(v), i)$, for $i \in [k]$.

These modifications are clearly valid; however these new z-*versions* of the formulations may have a significant amount of symmetric solutions. For example, take any connected k-partition $\{V_i\}_{i \in [k]}$ induced by a solution for $\mathcal{F}$, we could easily create an "equivalent" solution $\{V'_i\}_{i \in [k]}$ simply by relabeling the classes, that is, by assigning $V'_i = V_j$, $i \neq j$, such that $\{V_i\}_{i \in [k]} = \{V'_i\}_{i \in [k]}$. Thus, each feasible solution for $\mathcal{F}$ have $k!$ symmetric feasible solutions in $\mathcal{F}'$.

Avoiding symmetries reduces the space of feasible solutions without compromising the correctness of the formulation. Therefore, it is a very common practice among the integer linear programming community. However, a formulation with fewer symmetries will not always have a better practical performance. In fact, on some classes of instances, such a phenomenon was observed in our computational experiments (see Chapter 6).

Chapter 5

Cut Formulation

In this Chapter, we introduce a cut-based ILP formulation for BCP$_k$. This Cut formulation — denoted by $\mathcal{C}_k(G, w)$, or simply $\mathcal{C}$ — is defined directly on the input graph and it has a large number of inequalities. We derive valid inequalities, design polynomial-time separation routines and implement a branch-and-cut algorithm. Furthermore, we conduct a polyhedral study based on this Cut formulation.

5.1 Definitions and valid inequalities

Let (G, w) be an input for BCP$_k$, the ILP formulation we propose for BCP$_k$, called $\mathcal{C}_k(G, w)$, is based on the following central concept. Let u and v be two non-adjacent vertices in a graph G. We say that a set $S \subseteq V \setminus \{u, v\}$ is a (u, v)-*separator* if u and v belong to different components of $G - S$. We denote by $\Gamma(u, v)$ the collection of all minimal (u, v)-separators in G.

In the formulation, we use a binary variable $x_{v,i}$, for every $v \in V$ and $i \in [k]$, that is set to one if and only if v belongs to the i-th class.

$$
\begin{aligned}
\mathcal{C}_k(G, w) \qquad & \max \sum_{v \in V} w(v)\, x_{v,1} & & \\
& \text{s.t.} \sum_{v \in V} w(v)\, x_{v,i} \leq \sum_{v \in V} w(v)\, x_{v,i+1} & \forall i \in [k-1], & \quad (5.1)\\
& \sum_{i \in [k]} x_{v,i} \leq 1 & \forall v \in V, & \quad (5.2)\\
& x_{u,i} + x_{v,i} - \sum_{z \in S} x_{z,i} \leq 1 & \forall uv \notin E, S \in \Gamma(u, v), i \in [k], & \quad (5.3)\\
& x_{v,i} \in \{0, 1\} & \forall v \in V \text{ and } i \in [k]. & \quad (5.4)
\end{aligned}
$$

Inequalities (5.1) impose a non-decreasing weight ordering of the classes. Inequalities (5.2) require that every vertex is assigned to at most one class. Inequalities (5.3) guarantee that every class induces a connected subgraph (see Figure 5.1). The objective function maximizes the weight of the first class. Observe that, just like we did for the flow based formulations, we could easily create a formulation $\mathcal{C}'$ that do not require ordering the classes by their weights. More details on this formulation in Section 5.4.2.

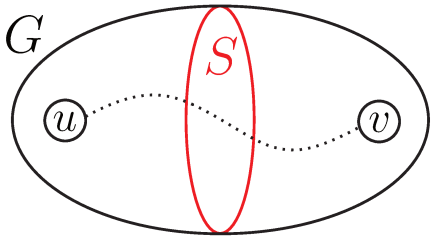


Figure 5.1: Illustration for a (u, v)-separator S. The dotted line represents a (u, v)-path.

In Section 5.2 we show that the separation problem for inequalities (5.3) can be solved in polynomial time. Thus, in view of the equivalence of separation and optimization problems [22], the linear relaxation of $\mathcal{C}$ can be solved in polynomial time.

Since feasible solutions of formulation $\mathcal{C}_k(G, w)$ may have empty classes and nodes not assigned to any class, to refer to these solutions we introduce the following concept. We say that $\mathcal{V} = \{V_i\}_{i=1}^k$ is a *connected k-subpartition* of G, if it is a connected k-partition of a subgraph (not necessarily proper) of G, and additionally, $w(V_i) \leq w(V_{i+1})$ for all $i \in [k-1]$. For such a connected k-subpartition $\mathcal{V}$, we denote by $\xi(\mathcal{V}) \in \mathbb{B}^{nk}$ the binary vector such that its non-null entries are precisely $\xi(\mathcal{V})_{v,i} = 1$ for all $i \in [k]$ and $v \in V_i$ (that is, $\xi(\mathcal{V})$ denotes the incidence vector of $\mathcal{V}$). To show our next results, let us define the polytope

$$\mathcal{P}_k(G, w) = \text{conv}(\{x \in \mathbb{B}^{nk} \colon x \text{ satisfies inequalities (5.1) } - \text{ (5.3) of } \mathcal{C}_k(G, w)\}).$$

We first prove that formulation $\mathcal{C}_k(G, w)$ correctly models BCP_k. Then, we present classes of valid inequalities that strengthen the formulation.

Proposition 5.1.1

$$\mathcal{P}_k(G, w) = \text{conv}(\{\xi(\mathcal{V}) \in \mathbb{B}^{nk} \colon \mathcal{V} \textit{ is a connected } k\textit{-subpartition of } G\}).$$

Proof. Consider first an extreme point $x \in \mathcal{P}_k(G, w)$. For each $i \in [k]$, we define the set of vertices $U_i = \{v \in V \colon x_{v,i} = 1\}$. It follows from inequalities (5.1) and (5.2) that $\mathcal{U} := \{U_i\}_{i=1}^k$ is a k-partition of a subgraph of G such that $w(U_i) \leq w(U_{i+1})$ for all $i \in [k-1]$. To prove that $\mathcal{U}$ is a connected k-subpartition, we suppose to the contrary that there exists $i \in [k]$ such that $G[U_i]$ is not connected. Hence, there exist vertices u and v belonging to two different components of $G[U_i]$. Moreover, there is a minimal set of vertices S that separates u and v and such that $S \cap U_i = \emptyset$. Thus, vector x violates inequalities (5.3), a contradiction.

To show the converse, consider now a connected k-subpartition $\mathcal{V} = \{V_i\}_{i=1}^k$ of G. Clearly $\xi(\mathcal{V})$, satisfies inequalities (5.1) and (5.2). Take a fixed $i \in [k]$. For every pair u,v of non-adjacent vertices in V_i, and every (u, v)-separator S in G, it holds that $S \cap V_i \neq \emptyset$, because $G[V_i]$ is connected. Therefore, $\xi(\mathcal{V})$ satisfies inequalities (5.3). ∎

Before showing the next results, we state the claim below.

Claim 5.1.2 *The inequalities*

$$\sum_{v \in V} w(v)\, x_{v,j} \leq \frac{w(G)}{(k-j+1)} \qquad \forall j \in [k], \tag{5.5}$$

are valid for $\mathcal{P}_k(G, w)$.

Proof. Because of the weight ordering imposed by inequality (5.1), the following inequalities are valid for any $j \in [k]$ (Figure 5.2):

$$(k-j+1) \sum_{v \in V} w(v)\, x_{v,j} \leq \sum_{i \in [k] \setminus [j-1]} \sum_{v \in V} w(v)\, x_{v,i} \leq w(G).$$

□

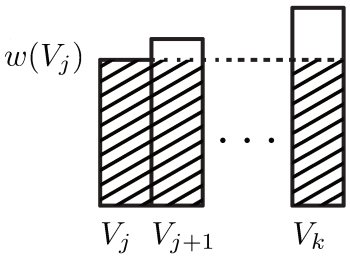


Figure 5.2: Illustration for claim 5.1.2. Each bar has unit width and a height that corresponds to the weight of a class. The claim follows from computing the dashed area.

Because of the structure of inequalities (5.5), we can take advantage of the extensive work regarding strong inequalities for the 0-1 knapsack polytope. We elaborate more on this matter in Section 5.3.1. In addition, this derived upper bound can also be used to perform a lifting of inequalities (5.3).

Proposition 5.1.3 *Let* u *and* v *be two non-adjacent vertices of* G*, and let* S *be a minimal* (u, v)*-separator. Let* $i \in [k]$*, and let* $L = \{z \in S\colon\ w(P_z) > w(G)/(k-i+1)\}$*, where* P_z *is a minimum-weight path between* u *and* v *in* G *that contains* z*. The following inequality is valid for* $\mathcal{P}_k(G, w)$*:*

$$x_{u,i} + x_{v,i} - \sum_{z \in S \setminus L} x_{z,i} \leq 1. \tag{5.6}$$

Proof. Consider an extreme point x of $\mathcal{P}_k(G, w)$, and define $V_i = \{v \in V\colon x_{v,i} = 1\}$ for each $i \in [k]$. Since inequalities (5.5) are valid, if u and v belong to V_i, then there exists a vertex $z \in S \setminus L$ such that z also belongs to V_i. Therefore, x satisfies inequality (5.6). □

While the result above shows a class of inequalities that dominates the inequalities (5.3), the next class is of a different nature. It was inspired by a result proposed by de Aragão and Uchoa [13] for a connected assignment problem.

Proposition 5.1.4 *Let* $q \geq 2$ *be a fixed integer, and let* S *be a subset of vertices of* G *containing* q *distinct pairs of vertices* $\{s_i, t_i\}$*,* $i \in [q]$*, all mutually disjoint. Let* $N(S)$ *be the set of neighbors of* S *in* $V \setminus S$*. Moreover, let* $\sigma\colon [q] \to [k]$ *be an injective function, and*

let $I = \{\sigma(i) \colon i \in [q]\}$. *If there is no collection of* q *vertex-disjoint* (s_i, t_i)*-paths in* $G[S]$, *then the following inequality is valid for* $\mathcal{P}_k(G, w)$*:*

$$\sum_{i \in [q]} \left(x_{s_i,\sigma(i)} + x_{t_i,\sigma(i)} \right) + \sum_{v \in N(S)} \sum_{i \in [k] \setminus I} x_{v,i} \leq 2q + |N(S)| - 1. \tag{5.7}$$

Proof. Suppose to the contrary that there exists an extreme point x of $\mathcal{P}_k(G, w)$ that violates inequality (5.7). Define $A = \sum_{i \in [q]} \left(x_{s_i,\sigma(i)} + x_{t_i,\sigma(i)} \right)$ and $B = \sum_{v \in N(S)} \sum_{i \in [k] \setminus I} x_{v,i}$. From inequalities (5.2), we have that $A \leq 2q$. Since x violates (5.7), it follows that $B > |N(S)| - 1$. Thus, since x satisfies inequalities (5.2), it follows that $B = |N(S)|$. Hence, every vertex in $N(S)$ belongs to a class that is different from those indexed by I. This implies that every class indexed by I contains precisely one of the q distinct pairs $\{s_i, t_i\}$. Therefore, there exists a collection of q vertex-disjoint (s_i, t_i)-paths in $G[S]$, a contradiction (see Figure 5.3). ∎

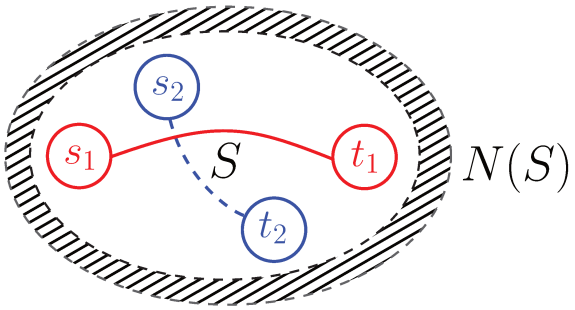


Figure 5.3: Illustration for Proposition 5.1.4 when $q = 2$. Assume s_1 and t_1 belong to class V_1. Since there is no (s_2, t_2)-path disjoint from a (s_1, t_1)-path in $G[S]$, s_2 and t_2 belong to V_2 only if $N(S) \cap V_2 \neq \emptyset$.

Kawarabayashi et al. [27] proved that, given an n-vertex graph G and a set of q pairs of terminals in G, the problem of deciding whether G contains q vertex-disjoint paths linking the given pairs of terminals can be solved in time $\mathcal{O}(n^2)$, for a fixed value of q. Hence, inequalities (5.7) can be separated in polynomial time when $S = V$.

5.2 Separation routines

We implemented a branch-and-cut approach based on the Cut formulation $\mathcal{C}_k(G, w)$. In this section, we describe the separation routines for inequalities (5.3) and (5.7) that are embedded in this algorithm. We report on the computational results obtained with this solving method in Chapter 6.

5.2.1 Connectivity inequalities

Let us focus first on the class of inequalities (5.3), henceforth called *connectivity inequalities*. We address here its corresponding separation problem: given a vector $\widetilde{x} \in \mathbb{R}^{nk}$, find connectivity inequalities that are violated by $\widetilde{x}$ or prove that this vector satisfies all such inequalities.

We address this problem with a construction that is similar to the one used for proving the directed version of Menger's theorem. Given the input graph $G = (V, E)$, for each $i \in [k]$, we define a digraph D_i with capacities $c_i \colon A(D_i) \to \mathbb{Q}_{\geq} \cup \{\infty\}$ assigned to its arcs, in the following manner. We set $V(D_i) = \{v_1, v_2 \colon v \in V\}$ and $A(D_i) = A_1 \cup A_2$, where $A_1 = \{(u_2, v_1), (v_2, u_1) \colon \{u, v\} \in E\}$ and $A_2 = \{(v_1, v_2) \colon v \in V\}$. We define $c_i(a) = \widetilde{x}_{v,i}$ if $a = (v_1, v_2) \in A_2$; and $c_i(a) = \infty$, if $a \in A_1$. Note that each arc in D_i with a finite capacity (i.e. each arc in A_2) is associated with a vertex of G. Now, for every pair of non-adjacent vertices $u, v \in V$ such that $\widetilde{x}_{u,i} + \widetilde{x}_{v,i} > 1$, we find in D_i a minimum (u_1, v_2)-separating cut. If the weight of such a cut is smaller than $\widetilde{x}_{u,i} + \widetilde{x}_{v,i} - 1$, then it is finite and the vertices of G associated with the arcs in this cut give an (u, v)-separator S in G that violates the connectivity inequality $\widetilde{x}_{u,i} + \widetilde{x}_{v,i} - \sum_{z \in S} \widetilde{x}_{z,i} \leq 1$ (see Figure 5.4).

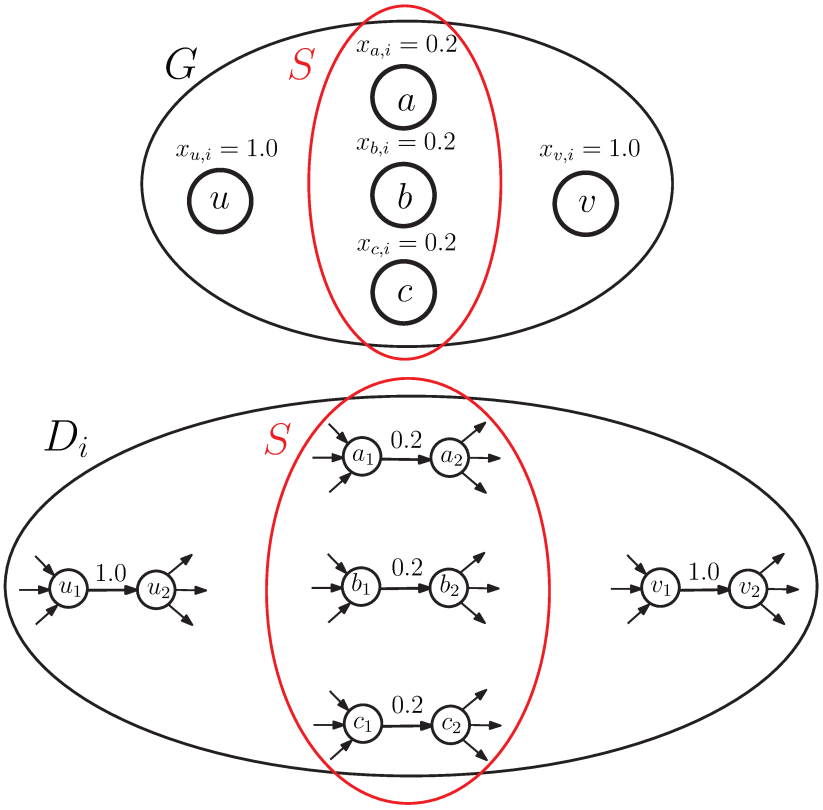


Figure 5.4: Separation of connectivity inequalities. Observe that the connectivity inequality $x_{u,i} + x_{v,i} - (x_{a,i} + x_{b,i} + x_{c,i}) \leq 1$ is violated in G.

Given an (u, v)-separator S, let H_u (resp. H_v) be the connected component of $G - S$ containing u (resp. v). We now describe a procedure for performing the lifting of the connectivity inequalities by removing iteratively unnecessary vertices from S. First, we remove every vertex z from S such that the neighborhood of z does not intersect with H_u and H_v (lines 2-4). Since removing a vertex from S changes the components of $G - S$, we use a Union-Find data structure to update the components. Next, we use Dijkstra's algorithm to remove from S the set L, as defined in Proposition 5.1.3 (lines 5-9).

Algorithm 1 Reduce Separator Algorithm

```
1: procedure ReduceSeparator(G, w, i, u, v, S)
2:     for z ∈ S do
3:         if N(z) ∩ H_v = ∅ or N(z) ∩ H_u = ∅ then
4:             S ← S \ {z}
5:     for z ∈ S do
6:         ℓ_uz ← ShortestPath(G, u, z)
7:         ℓ_vz ← ShortestPath(G, v, z)
8:         if ℓ_uz + ℓ_vz − w(z) > w(G)/(k−i+1) then
9:             S ← S \ {z}
       return S
```

The time complexity to separate the connectivity inequalities depends on the algorithm used to find a minimum cut. We use Goldberg's preflow algorithm [20] for maximum flow, whose time complexity is $\mathcal{O}(\tilde{n}^2\sqrt{\tilde{m}})$, for a digraph with $\tilde{n}$ vertices and $\tilde{m}$ arcs. Thus, in the worst-case, checking for every $i \in [k]$, and candidate pairs u,v in D_i, the time complexity of this separation algorithm is $\mathcal{O}(kn^4\sqrt{n+m})$. Despite the high time complexity, we note that, in the computational experiments, only a very small portion of the vertices are fractional. Hence, we can perform arc contractions on all arcs of D_i such that both of its endpoints correspond to vertices associated with variables of integer value. In other words, an arc $(u_2, v_1) \in A_1$ is contracted if $\widetilde{x}_{u,i} = \widetilde{x}_{v,i} = 1$, and an arc $(u_1, u_2) \in A_2$ is contracted if $\widetilde{x}_{u,i} = 1$. After such arc contractions, the graphs usually have a small number of vertices and arcs, and so the proposed separation algorithm runs quickly in practice.

5.2.2 Cross inequalities

Now we turn to the separation of inequalities (5.7) on planar graphs $G = (V, E)$, restricted to the case $S = V$. Consider a plane embedding of G, and let F be the boundary of a face with at least 4 distinct vertices and with no repeated vertices. Take four different vertices, say s_1, s_2, t_1, t_2, that appear in a clockwise order in F. Since G is planar, it does not contain vertex-disjoint paths P_1 and P_2, with endpoints s_1, t_1 and s_2, t_2, respectively. For $S = V$, inequalities (5.7) simplifies to

$$x_{s_1,\sigma(1)} + x_{s_2,\sigma(2)} + x_{t_1,\sigma(1)} + x_{t_2,\sigma(2)} \leq 3. \tag{5.8}$$

We refer to these inequalities as *cross inequalities* (Figure 5.5 explains the name).

For the separation problem of the cross inequalities induced by the vertices in F, where $|F| = f$, we implemented an $\mathcal{O}(fk^2)$ time complexity algorithm (the same complexity mentioned by Barboza [5], without much detail; the algorithms are possibly different). Next, we give more details on this separation algorithm.

Let $\widetilde{x} \in \mathbb{R}^{nk}$ be a fractional solution of formulation $\mathcal{C}$. Consider a linear ordering of the vertices in F which is obtained by traversing its vertices in clockwise order from an arbitrary fixed vertex. For every $j \in [f]$, we denote by $F(j)$ the jth vertex of F in such an ordering. Furthermore, define matrices L and R such that, for each $j \in [f]$ and

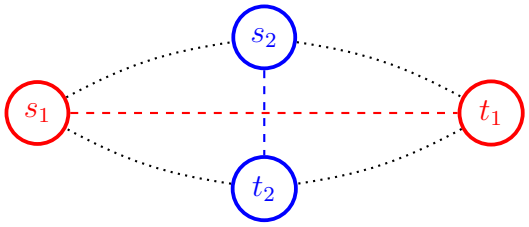


Figure 5.5: Drawing of the cross inequalities. The dotted lines represent the boundary F and we drew the graph in a way that the face considered is external. If s_1 and t_1 belong to V_1, then s_2 and t_2 cannot belong to V_2 (otherwise, there would be crossing edges).

each $i \in [k]$,

$$L(j,i) = \max_{j' \in [j]} \left\{ \widetilde{x}_{F(j'),i} \right\} \qquad \text{and} \qquad R(j,i) = \max_{j' \in [f] \setminus [j-1]} \left\{ \widetilde{x}_{F(j'),i} \right\}.$$

In other words, $L(j,i)$ (resp. $R(j,i)$) corresponds to the maximum value in an entry of $\widetilde{x}$ indexed by i and by a vertex that appears before (resp. after) $F(j)$ in the ordering. Clearly, L and R can be created in $\mathcal{O}(fk)$.

For every $j \in [f] \setminus \{1\}$, and every $i_1, i_2 \in [k]$ with $i_1 \neq i_2$, we define:

$$L_2(j,i_1,i_2) = \begin{cases} \widetilde{x}_{F(1),i_1} + \widetilde{x}_{F(2),i_2}, & \text{if } j = 2, \\ \max\left\{ L_2(j-1,i_1,i_2); \quad L(j-1,i_1) + \widetilde{x}_{F(j),i_2} \right\}, & \text{otherwise.} \end{cases}$$

Note that, given $j \geq 2$ and $i_1, i_2 \in [k]$, $L_2(j,i_1,i_2)$ is the maximum value of $\widetilde{x}_{F[j'],i_1} + \widetilde{x}_{F[j''],i_2}$ for all $j', j'' \in [j]$ with $j' < j''$. Our algorithm works as follows: for every $j \in \{3, \ldots, f-1\}$ and every $i_1, i_2 \in [k]$ with $i_1 \neq i_2$, it checks whether $L_2(j-1,i_1,i_2) + \widetilde{x}_{F(j),i_1} + R(j+1,i_2) > 3$, that is, whether there is a violated cross inequality (w.r.t. F) such that $\sigma(1) = i_1$, $\sigma(2) = i_2$ and $t_1 = F(j)$. Clearly, one may also keep track of the violated inequalities (if any).

Algorithm 2 Cross inequalities separation algorithm.

```
1: procedure CrossSeparation(F, L_2, R, x̃)
2:     for j ∈ [f − 1] \ [2] do
3:         for i_1 ∈ [k] do
4:             for i_2 ∈ [k] \ {i_1} do
5:                 if L_2(j − 1, i_1, i_2) + x̃_{F(j),i_1} + R(j + 1, i_2) > 3 then
6:                     return True (found violated cross inequality)
    return False
```

Proposition 5.2.1 *Let F, L_2, R and $\widetilde{x}$ be as defined. Then, Algorithm 2 returns true if and only if there exists a cross inequality with respect to F that is violated by $\widetilde{x}$.*

Proof. First we show that if there exists a cross inequality that is violated by $\widetilde{x}$, Algorithm 2 returns true. Let $x_{s_1,\sigma(1)} + x_{s_2,\sigma(2)} + x_{t_1,\sigma(1)} + x_{t_2,\sigma(2)} > 3$. Assume, without loss of generality, that $s_1, s_2, t_1, t_2 \in F$ are such that $(s_1, s_2, t_1, t_2) = (F(j_1), F(j_2), F(j_3), F(j_4))$,

with $1 \leq j_1 < j_2 < j_3 < j_4 \leq f$. As we stated previously, for any $j \geq 2$ and $i_1, i_2 \in [k]$,

$$L_2(j, i_1, i_2) = \max\{\widetilde{x}_{F[j'],i_1} + \widetilde{x}_{F[j''],i_2} \colon \forall j', j'' \in [j], j' < j''\}.$$

Hence, the inequality $x_{s_1,\sigma(1)} + x_{s_2,\sigma(2)} \leq L_2(j_3 - 1, \sigma(1), \sigma(2))$ holds. Moreover, because of how we defined matrix R, it is also true that $x_{t_2,\sigma(2)} \leq R(j_3 + 1, \sigma(2))$. Consider the iteration of the algorithm where $j = j_3$, $i_1 = \sigma(1)$ and $i_2 = \sigma(2)$ — if the algorithm returns true before this iteration, we are done. It follows from our remarks, that the condition on line 5 is satisfied, and the algorithm returns true.

Suppose now that there is no violated cross inequality but the algorithm returns true. Assume this happens at an iteration where $j = j_3$, $i_1 = a$ and $i_2 = b$. Let $j_1, j_2, j_4 \in [f]$ be such that $L_2(j_3 - 1, a, b) = \widetilde{x}_{F[j_1],a} + \widetilde{x}_{F(j_2),b}$ and $R(j_3 + 1, b) = \widetilde{x}_{F[j_4],b}$. Then, the algorithm returning true contradicts the assumption that $x_{F(j_1),a} + x_{F(j_2),b} + x_{F(j_3),a} + x_{F(j_4),b} \leq 3$. □

5.3 Enhancing the branch-and-cut algorithm

In this section, we elaborate on further techniques that we used to improve the performance of the branch-and-cut algorithm based on the Cut formulation.

5.3.1 Cover inequalities

Consider an input instance (G, w) of BCP_k. As we have previously mentioned, the following inequalities are valid for $\mathcal{P}_k(G, w)$.

$$\sum_{v \in V} w(v)\, x_{v,i} \leq \frac{w(G)}{k - i + 1} \qquad \forall i \in [k-1]. \tag{5.5}$$

Note that, for each $i \in [k-1]$, the corresponding inequality (5.5) defines a knapsack problem with capacity $w(G)/(k - i + 1)$. Hence, we can take advantage of the extensive work regarding strong inequalities for the 0/1 knapsack polytope as follows.

For each inequality of class (5.5), we use the heuristics implemented by Wolter [48] to separate lifted minimal cover inequalities and extended weight inequalities. For completeness, we now shortly discuss some fundamental ideas surrounding lifted cover inequalities.

We begin by defining the knapsack problem. Given a capacity $b \in \mathbb{R}_{\geq}$ and t items, where each item $i \in [t]$ has weight $a_i \in \mathbb{R}_{\geq}$ and profit $c_i \in \mathbb{R}_{\geq}$; the knapsack problem asks for a set of items that maximize the profit subject to the knapsack capacity. Formally, it can be written simply as $\max\{cx \colon ax \leq b, x \in \mathbb{B}^t\}$, where x_i, for $i \in [t]$, indicates if item i should be selected or not.

A set $C \subseteq [t]$ is called a *cover* if $\sum_{i \in C} a_i > b$, in lay terms, it is simply a set of items that do not fit in the knapsack. Furthermore, a *minimal cover* $C \subseteq [t]$, is a cover such that, for any $i \in C$, the set $C \setminus \{i\}$ is not a cover. Any (minimal) cover C, defines a corresponding *(minimal) cover inequality*

$$\sum_{i \in C} x_i \leq |C| - 1. \tag{5.9}$$

To obtain *lifted cover inequalities* we start from a minimal cover inequality (5.9), and then find coefficients $\alpha_i \geq 0$, for $i \in [t] \setminus C$, such that the resulting inequality

$$\sum_{i \in C} x_i + \sum_{i \in [t] \setminus C} \alpha_i x_i \leq |C| - 1 \tag{5.10}$$

is valid. Clearly, if $\alpha_i > 0$, for some $i \in [t] \setminus C$, inequality (5.10) dominates (5.9).

Now, let p be the weight of a heaviest item in the cover, that is, $p = \max_{i \in C}\{a_i\}$. We define the *extension* of C as being the set $E(C) = C \cup \{i \in [t] \setminus C \colon a_i \geq p\}$. A very simple lifted cover inequality could be obtained simply by noticing that if (5.9) is valid, the *extended cover inequality* $\sum_{i \in E(C)} x_i \leq |C| - 1$ is also valid. Naturally, more elaborate ideas are available in the literature for performing the lifting of cover inequalities. As we said in the beginning, this was a very brief exposition to some of the concepts surrounding valid inequalities for the 0/1 knapsack polytope. For more details, the reader is referred to Wolter's thesis [48], and the comprehensive survey written by Hojny, Gally, Habeck, Lüthen, Matter, Pfetsch and Schmitt [25].

5.3.2 Domain Propagation

Suppose that our branch-and-cut algorithm is currently exploring a node in the branch-and-bound tree. *Domain propagation* refers to techniques that aim to tighten the variable bounds based on the domain of the other variables in the current node. When a complete description of the problem formulation is available to the solver, the solver itself can reduce the domain of the variables. However, when implementing a branch-and-cut algorithm, for example, the solver may not be able to effectively reduce the domain of the variables, since only a small part of the inequalities might be available to it. Thus, based on the work of Hojny et al. [26], we implemented an algorithm to effectively reduce the domains of the variables in our formulation $\mathcal{C}$.

For any $j \in [k]$, let $F_j \subseteq V$ be the set of vertices fixed on class j in the current node of the branch-and-bound tree; in other words, F_j contains all vertices $v \in V$ such that variable $x_{v,j}$ was fixed to one. Let us fix $i \in [k]$ and assume $F_i \neq \emptyset$. Consider the graph $G_i = G[V \setminus \bigcup_{i' \in [k] : i' \neq i} F_{i'}]$, that is, G_i is the graph obtained from G by removing the vertices that were fixed in classes distinct from i. Let $C_1, \ldots, C_t$ be the connected components of G_i, and suppose, without loss of generality, that $F_i \cap V(C_t) \neq \emptyset$ (see Figure 5.6). Then, for any vertex $u \in V(C_j)$, with $j \in [t-1]$, we can fix the variable $x_{u,i}$ at zero, or if variable $x_{u,i}$ was already fixed to one, we can declare the current node as infeasible and prune it (see Figure 5.6).

5.4 Polyhedral Study

We now present polyhedral results based on the Cut formulation. First, we consider the 1-BCP$_k$ case, a version of the problem where the weight of a class is the same as its cardinality. Next, we present formulation $\mathcal{C}'$, the z-version of $\mathcal{C}$, and we derive polyhedral results for it.

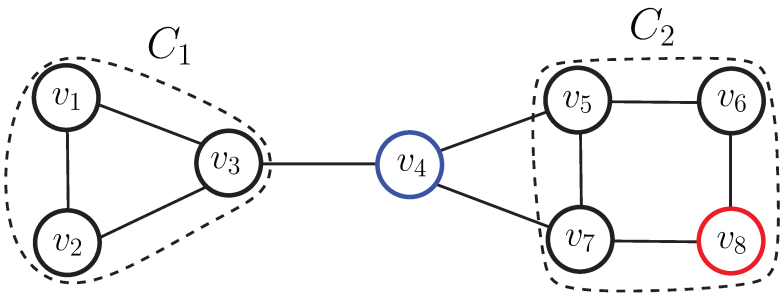


Figure 5.6: Domain propagation. Suppose in a given node of the branch-and-bound tree we have the above illustrated configuration, in which $F_1 = \{v_8\}$ and $F_2 = \{v_4\}$. Clearly, for this node, we can fix $x_{v,i} = 0$, for all $v \in V(C_1)$.

5.4.1 Polyhedral Study for 1-BCP$_k$

In this section, we focus on 1-BCP$_k$, the special case of BCP$_k$ in which all the vertices have unit weight. In this case, instead of $\mathcal{P}_k(G, w)$, we simply write $\mathcal{P}_k(G)$, the polytope defined as the convex hull of $\{x \in \mathbb{B}^{nk} : x \text{ satisfies (5.1) - (5.3)}\}$.

We denote by $\text{opt}(G)$ the optimal cost of a solution for 1-BCP$_k$ in the input graph G. Since we assume that G has a feasible solution, it follows that $\text{opt}(G) > 0$. Note that, if G has no matching of size k, then $\text{opt}(G) = 1$ (contrapositive in Figure 5.7); and thus, it is easy to find an optimal solution — it is sufficient to do a depth-first search until exactly $k-1$ vertices were still not visited. Moreover, using Edmonds blossom algorithm [16], we can efficiently check if G has a matching with cardinality k or not. Hence, we assume from now on that G has a matching of size k, since otherwise, the problem would be solvable in polynomial time.

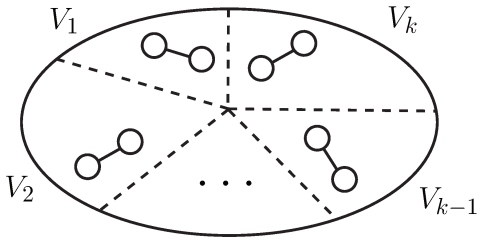


Figure 5.7: If $\text{opt}(G) > 1$, then G has a matching of size k.

For each $v \in V$ and $i \in [k]$ we shall construct a binary vector $\chi(v, i)$ that belongs to $\mathcal{P}_k(G)$. Let us denote by $e(v, i) \in \mathbb{B}^{nk}$ the unit vector such that its single non-null entry is indexed by v and i. Now consider any set $S \subseteq V \setminus \{v\}$, $|S| = k - i$, and a bijective function $\nu \colon S \cup \{v\} \to [k] \setminus [i-1]$, such that $\nu(v) = i$. This function indicates to which class the elements of $S \cup \{v\}$ are assigned to. Since G has a matching of size k, it follows that $n \geq 2k$ and such a set S exists. Fix a pair (S, ν), where S and ν are as previously defined. Let $\chi(v, i) \in \mathbb{B}^{nk}$ be the vector $e(v, \nu(v)) + \sum_{u \in S} e(u, \nu(u))$. Note that $\chi(v, i)$ belongs to $\mathcal{P}_k(G)$, it is the incidence vector of a connected k-subpartition, say $\mathcal{S}_i = \{S_i, \ldots, S_k\}$ of G, in which v belongs to the class S_i, and each vertex of $S \subseteq V \setminus \{v\}$ belongs to one of the classes $S_{i+1}, \ldots, S_k$, all of which are singletons (Figure 5.8).

To be rigorous, we should write $\chi^{s,\nu}(v, i)$, as different choices of S and ν give rise

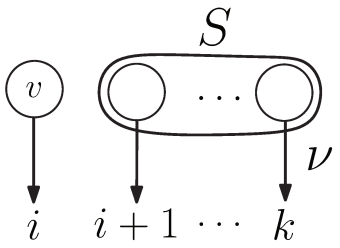


Figure 5.8: Graphical representation of the bijection $\nu\colon S \cup \{v\} \to [k] \setminus [i-1]$, used to define the incidence vector $\chi(v,i)$.

to different vectors, but we simply write $\chi(v,i)$ with the understanding that it refers to some S and bijection ν.

For all the forthcoming proofs we assume that $V = \{v_1, \ldots, v_n\}$ and that the coordinates of a vector $x \in \mathbb{R}^{nk}$ are ordered as $x = (x_{v_1,1}, \ldots, x_{v_n,1}, \ldots, x_{v_1,k}, \ldots, x_{v_n,k})^T$.

Proposition 5.4.1 *$\mathcal{P}_k(G)$ is full-dimensional, that is,* $\dim(\mathcal{P}_k(G)) = nk$.

Proof. Let $X = \{\chi(v,i) \in \mathbb{B}^{nk}\colon v \in V \text{ and } i \in [k]\}$ be the set of the nk vectors previously defined. Let M be the matrix whose columns are precisely the vectors in X:

$$M = [\chi(v_1,1), \ldots, \chi(v_n,1), \ldots, \chi(v_1,k), \ldots, \chi(v_n,k)].$$

Note that for any vector $\chi(v,i)$, with $v = v_t \in V$ and $i \in [k]$, all of the coordinates of $\chi(v,i)$ that come "before" coordinate (v_t,i) are set to zero, in other words,

$$\chi(v,i) = (0, \ldots, 0, \chi(v,i)_{v_t,i}, \chi(v,i)_{v_{t+1},i}, \ldots, \chi(v,i)_{v_n,k})^T.$$

Thus, matrix M has dimension nk, since it is a lower triangular square matrix with nonzero diagonal entries. Hence, the vectors in X are linearly independent. Considering that X and the null vector belongs to $\mathcal{P}_k(G)$, we conclude that $\dim(\mathcal{P}_k(G)) = nk$. ∎

In the forthcoming proofs, we have to refer to some connected k-subpartitions of G, defined (not uniquely) in terms of distinct vertices u, v of G, and specific classes i, j, where $i < j$. For that, we define a short notation to represent the incidence vectors of these connected k-subpartitions. Given such u, v, and i, j, choose two sets of vertices S and $\hat{S}$ in G, both of cardinality $k-i+1$, and bijections $\pi\colon S \to \{i, \ldots, k\}$ and $\hat{\pi}\colon \hat{S} \to \{i, \ldots, k\}$ such that $u \in S \cap \hat{S}$, $v \in S \setminus \hat{S}$, $\pi(u) = \hat{\pi}(u) = i$ and $\pi(v) = j$ (see Figure 5.9).

Let $\phi(u,i,v,j)$ and $\psi(u,i,v)$ be vectors in $\{0,1\}^{nk}$ such that their non-null entries are precisely: $\phi(u,i,v,j)_{z,\pi(z)} = 1$ for $z \in S$, and $\psi(u,i,v)_{z,\hat{\pi}(z)} = 1$ for $z \in \hat{S}$. Note that $\phi(u,i,v,j)_{u,i} = \phi(u,i,v,j)_{v,j} = \psi(u,i,v)_{u,i} = 1$ and $\psi(u,i,v)_{v,\ell} = 0$ for all $\ell \in [k]$. Moreover, the vectors $\phi(u,i,v,j)$ and $\psi(u,i,v)$ clearly belong to $\mathcal{P}_k(G)$.

Proposition 5.4.2 *For every $v \in V$ and $i \in [k]$, $x_{v,i} \geq 0$ induces a facet of $\mathcal{P}_k(G)$.*

Proof. Similarly to the proof of Proposition 5.4.1, let $X_1 = \{\chi(v,j) \in \mathbb{B}^{nk}\colon j \in [k] \setminus \{i\}\}$. Additionally, we define $X_2 = \{\psi(u,j,v) \in \mathbb{B}^{nk}\colon u \in V \setminus \{v\} \text{ and } j \in [k]\}$.

Let $v = v_t$. For every $j \in [k]$, let $M_j \in \mathbb{R}^{nk \times n}$ be the following matrix

$$M_j = [\psi(v_1,j,v), \ldots, \psi(v_{t-1},j,v), \chi(v,j), \psi(v_{t+1},j,v), \ldots, \psi(v_n,j,v)].$$

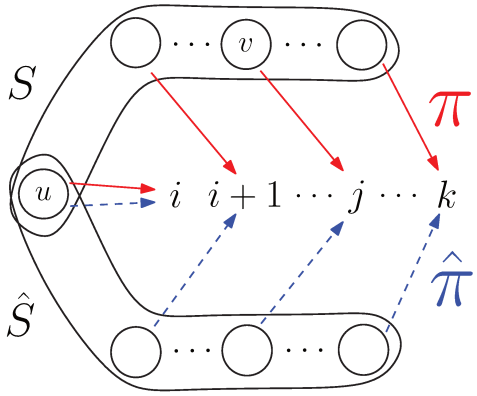


Figure 5.9: Bijections from sets S and $\hat{S}$ to $\{1, \ldots, k\}$. These functions are used to define the incidence vectors $\phi(u, i, v, j)$ and $\psi(u, i, v)$.

Consider the matrix $M \in \mathbb{R}^{nk \times nk}$ obtained by concatenating the previously defined matrices. In other words, $M = [M_1|M_2|\ldots|M_k]$, where the symbol $|$ denotes matrix concatenation. One can easily check that M is a lower triangular square matrix with nonzero elements in the main diagonal. Furthermore, besides $\chi(v, i)$, all other columns in M are vectors that belong to $X_1 \cup X_2$. Thus, $X_1 \cup X_2$ contains $nk-1$ linearly independent vectors.

Since the vectors in $\{\mathbf{0}\} \cup X_1 \cup X_2$ belongs to the face $\{x \in \mathcal{P}_k(G) \colon x_{v,i} = 0\}$ and they are all affinely independent, we conclude that the inequality $x_{v,i} \geq 0$ induces a facet of $\mathcal{P}_k(G)$. □

In what follows, considering that the polytope $\mathcal{P}_k(G)$ is full-dimensional, to prove that a face $\hat{F} = \{x \in \mathcal{P}_k(G) \colon \hat{\lambda}x = \hat{\lambda}_0\}$ is a facet of $\mathcal{P}_k(G)$, we show that if a nontrivial face $F = \{x \in \mathcal{P}_k(G) \colon \lambda x = \lambda_0\}$ of $\mathcal{P}_k(G)$ contains $\hat{F}$, then there exists a real positive constant c such that $\lambda = c\hat{\lambda}$ and $\lambda_0 = c\hat{\lambda}_0$.

Proposition 5.4.3 *For every $v \in V$, $\sum_{i \in [k]} x_{v,i} \leq 1$ induces a facet of $\mathcal{P}_k(G)$.*

Proof. Fix a vertex $v \in V$. Let $\hat{F} = \{x \in \mathcal{P}_k(G) \colon \hat{\lambda}x = 1\}$, where $\hat{\lambda}x \leq 1$ corresponds to $\sum_{i \in [k]} x_{v,i} \leq 1$. Let $F = \{x \in \mathcal{P}_k(G) \colon \lambda x = \lambda_0\}$ be a nontrivial face of $\mathcal{P}_k(G)$ such that $\hat{F} \subseteq F$. Using induction we shall prove that $\lambda_{v,i} = \lambda_0$ and $\lambda_{u,i} = 0$ for every $u \in V \setminus \{v\}$ and $i \in [k]$. As a base case, we start by showing that $\lambda_{v,k} = \lambda_0$ and $\lambda_{u,k} = 0$. Next, let $\ell \in [k-1]$, assuming the induction hypothesis that $\lambda_{v,i} = \lambda_0$ and $\lambda_{u,i} = 0$, for $i \in \{\ell + 1, \ldots, k\}$, we show that $\lambda_{v,\ell} = \lambda_0$ and $\lambda_{u,\ell} = 0$.

Base case: since G is nontrivial and connected, it is easy to see that G has a set of n nested connected subgraphs $T_1, T_2, \ldots, T_n$ such that T_1 consists solely of the vertex v, $T_j \subset T_{j+1}$ for each $j \in [n-1]$, and $V(T_n) = V$. (Take a spanning tree in G, and starting from v, define the subsequent subgraphs by adding at each step an appropriate edge and vertex from this spanning tree. See Figure 5.10.)

Consider the set of vectors $A = \{e(G_j, k)\}_{j \in [n]}$, where $e(G_j, k) = \sum_{u \in V(G_j)} e(u, k)$ for every $j \in [n]$. Since $v \in V(G_j)$ for all $j \in [n]$, it follows that $A \subseteq \hat{F}$. Therefore, $\lambda(e(G_1, k)) = \lambda_0$ and thus $\lambda_{v,k} = \lambda_0$. Additionally, since

$$\lambda(e(G_2, k)) = \lambda(e(G_3, k)) = \ldots = \lambda(e(G_n, k)) = \lambda_0,$$

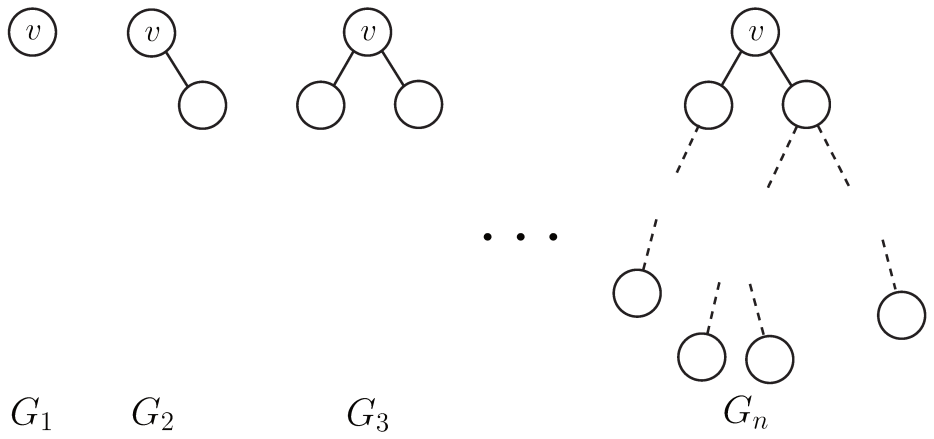


Figure 5.10: Nested trees used to define the set $A = \{e(G_j, k)\}_{j\in[n]}$.

it follows that $\lambda_{u,k} = 0$ for all $u \in V \setminus \{v\}$.

Induction step: let $\ell \in [k-1]$ and suppose that $\lambda_{v,i} = \lambda_0$ and $\lambda_{u,i} = 0$, for $u \in V \setminus \{v\}$ and $i \in \{\ell+1, \ldots, k\}$. Now define the set of vectors $B = \{\phi(u, \ell, v, k) \colon u \in V \setminus \{v\}\}$. Note that $B \subseteq \hat{F}$, since v belongs to exactly one class of the partition corresponding to each vector in B. Recall that $\phi(u, \ell, v, k) = \sum_{v'\in S} e(v', \pi(v'))$ and $\pi(v') > \ell$, for $v' \in S \setminus \{u\}$. Using the induction hypothesis, for each $u \in V \setminus \{v\}$, we obtain the following

$$\lambda(\phi(u, \ell, v, k)) = \lambda \left(e(u, \ell) + \sum_{v'\in S\setminus\{u\}} e(v', \pi(v')) \right) = \lambda_{u,\ell} + \lambda_0 = \lambda_0,$$

and thus, $\lambda_{u,\ell} = 0$. Moreover, observe that $\chi(v, \ell)$ belongs to $\hat{F}$. Using similar reasoning, it follows that $\lambda(\chi(v, \ell)) = \lambda_{v,\ell} = \lambda_0$.

We conclude that $\lambda_{v,i} = \lambda_0$ and $\lambda_{u,i} = 0$ for every $u \in V \setminus \{v\}$ and $i \in [k]$. In other words, $\lambda = \lambda_0\hat{\lambda}$. Since $\lambda_0 \neq 0$ (otherwise F would be a trivial face), it follows that $\lambda x \leq \lambda_0$ is a multiple scalar of $\hat{\lambda}x \leq 1$, and therefore $\hat{F}$ is a facet of $\mathcal{P}_k(G)$. □

Before showing our results regarding the connectivity inequalities, we need to introduce some concepts and notations. Let u and v be two non-adjacent vertices of G and let S be a minimal (u, v)-separator in G. We denote by $[u, v, S]$ such a triple, and denote by H_u and H_v the components of $G - S$ that contain u and v, respectively. Since S is minimal, it follows that every vertex in S has at least one neighbor in H_u and one in H_v.

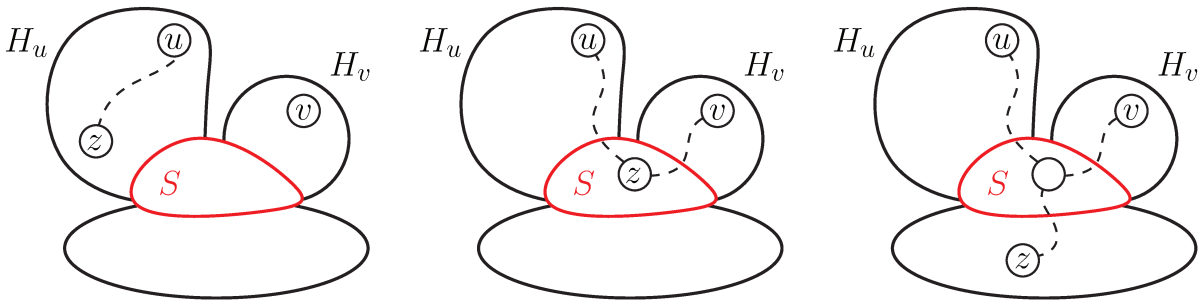
For any vertex z in G, we denote by G_z (when $[u, v, S]$ is clear from the context) a minimum size connected subgraph of G containing z, with the following properties:

(i) G_z contains v and is contained in H_v, if $z \in V(H_v)$;

(ii) G_z contains u and is contained in H_u, if $z \in V(H_u)$;

(iii) G_z contains u and v, otherwise.

Note that, in the latter case, G_z contains exactly one vertex of S. Clearly, such a subgraph always exists (and may not be unique). Moreover, if $z \in \{u, v\}$, the degree of z is zero. Likewise, if $z \in S$, the degree of z is two. In all other cases, the degree of z in G_z is exactly one and the subgraph $G_z - z$ is connected (see Figure 5.11).



(a) $z \in V(H_u)$. (b) $z \in S$. (c) $z \notin (V(H_u) \cup V(H_v) \cup S)$.

Figure 5.11: Subgraph G_z.

Let $i \in [k]$ and $z \in V$. We say that a connected $(k-i)$-subpartition $\mathcal{V}_z$ is a *robust connected $(k-i)$-partition* of $G - G_z$ if $\mathcal{V}_z = \{V_j\}_{j \in [k] \setminus [i]}$, and $|V(G_z)| \leq |V_j|$ for all $j \in [k] \setminus [i]$. We say that G is (u, v, S, i)-*robust* if, for every $z \in V$, there is a graph G_z such that $G - G_z$ has a robust connected $(k-i)$-partition.

Theorem 5.4.4 *Let u and v be non-adjacent vertices in G, let S be a minimal (u, v)-separator, and let $i \in [k]$. Then the inequality*

$$x_{u,i} + x_{v,i} - \sum_{s \in S} x_{s,i} \leq 1$$

induces a facet of $\mathcal{P}_k(G)$ if and only if G is (u, v, S, i)-robust.

Proof. Fix u, v, S and i as in the statement of the theorem. We first prove that, if the graph G is (u, v, S, i)-robust, then the inequality $x_{u,i} + x_{v,i} - \sum_{z \in S} x_{z,i} \leq 1$ defines a facet of $\mathcal{P}_k(G)$. Let $\hat{\lambda} x \leq 1$ denote this inequality and consider the face defined by it: $\hat{F} = \{x \in \mathcal{P}_k(G) \colon \hat{\lambda} x = 1\}$. Now let $F = \{x \in \mathcal{P}_k(G) \colon \lambda x = \lambda_0\}$ be a nontrivial face of $\mathcal{P}_k(G)$ such that $\hat{F} \subseteq F$.

We next prove that $\lambda = \lambda_0 \hat{\lambda}$. For that, we shall refer to entries of λ of the form $\lambda_{y,j}$, where $y \in V$ and $j \in [k]$, and analyze separately the cases: $j \in [i-1]$ (Case 1), $j \in [k] \setminus [i]$ (Case 2) and $j = i$ (Case 3).

For the Case 1 and Case 2, fix a vertex $z \in S$. Since G is (u, v, S, i)-robust, there is a graph G_z such that $G - G_z$ has a robust connected $(k-i)$-partition $\mathcal{V}_z = \{V_j\}_{j \in [k] \setminus [i]}$. By definition, G_z contains the vertices u, v, and z; thus, each class in $\mathcal{V}_z$ has size at least 3. Let $V_i = V(G_z)$, and let $W = \{v_{i+1}, \ldots, v_k\}$, where $v_j \in V_j$, for $j \in [k] \setminus [i]$.

For simplicity, set $v = v_i$. Let γ be the vector in $\{0,1\}^{nk}$ defined as $\gamma = \sum_{j=i}^{k} e(v_j, j)$ (the incidence vector of a $(k-i+1)$-subpartition whose classes are the singletons $\{v_j\}$, for $j \in \{i, \ldots, k\}$). Clearly, $\gamma \in \hat{F}$. This vector will be used to construct new vectors in $\hat{F}$.

Case 1. Let $j \in [i-1]$.

Consider a vertex $y \in V$. Since $|W| = k - i$ and $n \geq 2k$, there exists a set of vertices $\{v_{j+1}, \ldots, v_{i-1}\}$ that is disjoint from $W \cup \{u, v, y\}$.

(1a) If $y \notin W \cup \{v\}$, then the following vectors belong to $\hat{F}$:

$$\gamma' = \gamma + \sum_{\ell=j+1}^{i-1} e(v_\ell, \ell) \qquad \text{and} \qquad \gamma'' = \gamma' + e(y, j).$$

Therefore, $\lambda\gamma' = \lambda_0$ and $\lambda\gamma'' = \lambda(\gamma' + e(y,j)) = \lambda_0$. Hence, $\lambda(e(y,j)) = \lambda_{y,j} = 0$ for $y \notin W \cup \{v\}$.

(1b) If $y \in W \cup \{v\}$, then $y = v_\ell$ for some $\ell \in \{i, i+1, \ldots, k\}$. Take the subpartition γ' and construct a new subpartition γ'_a by replacing y with u. Then, from γ'_a, construct the subpartition γ'_b by adding y to class j. These subpartitions are described by the vectors

$$\gamma'_a = \gamma' - e(y, \ell) + e(u, \ell) \qquad \text{and} \qquad \gamma'_b = \gamma'_a + e(y, j).$$

Note that both vectors γ'_a and γ'_b are in $\hat{F} \subseteq F$. Thus, $\lambda_{y,j} = 0$ for $y \in W \cup \{v\}$.

Case 2. Let $j \in [k] \setminus [i]$.

(2a) If $y = v_j$, let w_ℓ be a vertex that is a neighbor of v_ℓ in $G[V_\ell]$ for $\ell \in [k] \setminus [j-1]$. Then, it is immediate that the following vectors belong to $\hat{F}$:

$$\theta = \gamma + \sum_{\ell=j}^{k} e(w_\ell, \ell) \qquad \text{and} \qquad \theta' = \theta - e(y, j).$$

Observe that $\theta, \theta' \in \hat{F} \subseteq F$ and thus $\lambda_{y,j} = 0$.

(2b) If $y \neq v_j$, consider the vector $\gamma_u = \gamma - e(v, i) + e(u, i)$. If $y = v$, take $\tilde{\gamma}_u = \gamma_u - e(v_j, j) + e(y, j)$; and if $y \neq v$, take $\tilde{\gamma}_y = \gamma - e(v_j, j) + e(y, j)$. Recall that $\lambda_{v_j,j} = 0$. Since γ, γ_u, $\tilde{\gamma}_u, \tilde{\gamma}_y \in \hat{F} \subseteq F$, it follows that $\lambda_{y,j} = 0$.

Case 3. We now focus on the entries of λ indexed by i.

For each $y \in V$, let G_y be such that $G - G_y$ admits a robust connected $(k-i)$-subpartition $\mathcal{V}_y$ (where G_y is a subgraph of G as previously defined). Let ϕ denote the incidence vector of $\mathcal{V}_y$, and let $\phi_y = \phi + \sum_{w \in V(G_y)} e(w, i)$.

(3a) Suppose $y \notin S \cup \{u, v\}$. In this case, $G_y - y$ is connected. Consider $\phi' = \phi_y - e(y, i)$. Clearly, $\phi_y, \phi' \in \hat{F} \subseteq F$. Thus, $\lambda_{y,i} = 0$ for every $y \in V \setminus (S \cup \{u, v\})$.

(3b) Let $y \in S \cup \{u, v\}$. If $y \in \{u, v\}$ then G_y consists of precisely the vertex y. Since ϕ_y belong to $\hat{F} \subseteq F$, we conclude that

$$\lambda(\phi + e(y, i)) = \lambda_{y,i} = \lambda_0.$$

Note that because Case 2 is valid, $\lambda\phi = 0$.

If $y \in S$, then $\{u, v, y\} \subseteq V(G_y)$ and $V(G_y) \cap S = \{y\}$. Since $\phi_y \in \hat{F} \subseteq F$ for every $y \in S$, recalling that $\lambda_{w,i} = 0$ for every $w \in V \setminus (S \cup \{u, v\})$, we have that $\lambda_{u,i} + \lambda_{v,i} + \lambda_{y,i} = \lambda_0$ for every $y \in S$. Combining with the previous equalities, we conclude that $\lambda_{y,i} = -\lambda_0$ for every $y \in S$.

Putting together Cases 1, 2 and 3, we conclude that $\lambda = \lambda_0 \hat{\lambda}$, as we wanted to show.

Let us prove now that if $x_{u,i} + x_{v,i} - \sum_{s \in S} x_{s,i} \leq 1$ induces a facet of $\mathcal{P}_k(G)$, then G is (u, v, S, i)-robust. Suppose G is not (u, v, S, i)-robust. Then, there is a vertex $z \in V$ such that for no minimal connected graph G_z there is a robust connected $(k-i)$-partition $\{V_j\}_{j \in [k] \setminus [i]}$ of $G - G_z$. This means that there is no connected $(k-i+1)$-subpartition $\{V_j\}_{j \in [k] \setminus [i-1]}$ of G such that

(i) $z \in V_i$,

(ii) $|V_i \cap \{u\}| + |V_i \cap \{v\}| - |V_i \cap S| = 1$, and

(iii) $|V_{j-1}| \leq |V_j|$ for all $j \in [k] \setminus [i]$.

Consider now a vertex $\bar{x}$ of $\mathcal{P}_k(G)$ that belongs to $\hat{F}$, and suppose that $\bar{x}_{z,i} = 1$. Let $W_j = \{w \in V \colon \bar{x}_{w,j} = 1\}$ for each $j \in [k] \setminus [i-1]$. Since $\bar{x} \in \hat{F}$, it clearly holds that $|W_i \cap \{u\}| + |W_i \cap \{v\}| - |W_i \cap S| = 1$ and $|W_{j-1}| \leq |W_j|$ for every $j \in [k] \setminus [i]$. Thus, $\{W_j\}_{j \in [k] \setminus [i-1]}$ is a connected $(k-i+1)$-subpartition of G that satisfies properties (i)–(iii), a contradiction. Hence, we have $\bar{x}_{z,i} = 0$ for every vertex $\bar{x} \in \hat{F}$. But then, $\hat{F}$ is contained in the face $\{x \in \mathcal{P}_k(G) : x_{z,i} = 0\}$, and so $\hat{F}$ is not a facet of $\mathcal{P}_k(G)$. This concludes the proof of the theorem. □

5.4.2 Polyhedral Study of a relaxed formulation

Consider the following formulation $\mathcal{C}'_k(G, w)$, obtained from formulation $\mathcal{C}_k(G, w)$ by dropping the ordering of the classes (5.1) and adding a new non-negative variable z which is smaller than the weight of each class.

$$
\begin{aligned}
\mathcal{C}'_k(G, w) \qquad & \max z \\
& \text{s.t. } z \leq \sum_{v \in V} w(v)\, x_{v,i} && \forall i \in [k], && (5.11)\\
& \sum_{i \in [k]} x_{v,i} \leq 1 && \forall v \in V, && (5.2)\\
& x_{u,i} + x_{v,i} - \sum_{s \in S} x_{s,i} \leq 1 \quad \forall uv \notin E, S \in \Gamma(u, v), i \in [k], && (5.3)\\
& x_{v,i} \in \{0, 1\} && \forall v \in V \text{ and } i \in [k], && (5.4)\\
& z \in \mathbb{R}_{\geq}\ . && && (5.12)
\end{aligned}
$$

Let $\bar{\mathcal{P}}_k(G, w) = \text{conv}(\{(x, z) \in \mathbb{B}^{nk} \times \mathbb{R}_{\geq} : (x, z) \text{ satisfies } (5.11), (5.2) \text{ and } (5.3)\})$. For ease of presentation, we refer to this polytope simply as $\bar{\mathcal{P}}$. The polyhedral study we

derive for $\bar{\mathcal{P}}$ is based on the previous results we obtained for $\mathcal{P}_k(G)$. However, $\bar{\mathcal{P}}$ has one extra dimension when compared to $\mathcal{P}_k(G)$, due to the introduction of the real variable z. To handle this difficulty, we define $\bar{\mathcal{P}}_x$ as being the projection of $\bar{\mathcal{P}}$ into the space of the x variables, i.e., $\bar{\mathcal{P}}_x = \text{conv}(\{x \in \mathbb{B}^{nk} : x \text{ satisfies (5.2) and (5.3)}\})$. Clearly, $\mathcal{P}_k(G) \subseteq \bar{\mathcal{P}}_x$.

The next results are based on the following strategy. First, we show in Proposition 5.4.5 that, on some cases, it is possible to lift a face $\bar{F}_x$ of $\bar{\mathcal{P}}_x$ to obtain a face $\bar{F}$ of $\bar{\mathcal{P}}$, such that $\dim(\bar{F}) \geq \dim(\bar{F}_x) + 1$. In particular, this proposition allows us to say that some inequalities that induce facets of $\bar{\mathcal{P}}_x$ are also facet-defining for $\bar{\mathcal{P}}$. Next, we show in Lemma 5.4.6 that facet-defining inequalities of $\mathcal{P}_k(G)$ may also define facets of $\bar{\mathcal{P}}_x$. Using both results, we conclude that some inequalities that induce facets of $\mathcal{P}_k(G)$ also define facets of $\bar{\mathcal{P}}$.

Before we proceed, let us define p as the number of vertices in G with positive weights. We assume $p \geq k$, since otherwise an optimal solution has cost zero and the problem is solvable in polynomial time.

Proposition 5.4.5 *Let $\pi x \leq \pi_0$ be a valid inequality for the polytope $\bar{\mathcal{P}}$. Consider the faces $\bar{F} = \{(x,z) \in \bar{\mathcal{P}} : \pi x = \pi_0\}$ and $\bar{F}_x = \{x \in \bar{\mathcal{P}}_x : \pi x = \pi_0\}$. If $\dim(\bar{F}_x) \geq nk - p + 1$, then $\dim(\bar{F}) \geq \dim(\bar{F}_x) + 1$.*

Proof. Let d be the dimension of $\bar{F}_x$. Thus, there exists a set $\{x^0, x^1, \ldots, x^d\}$ of $d+1$ affinely independent vectors in $\bar{F}_x$. Let $Q = \{(x^0,0), (x^1,0), \ldots, (x^d,0)\}$, clearly $Q \subseteq \bar{F}$. It remains to construct one extra vector that is affinely independent with respect to all the vectors in Q.

Consider the vector

$$\bar{x} = \frac{1}{d+1}(x^0 + x^1 + \ldots + x^d),$$

since $\bar{x}$ is a convex combination of vectors in $\bar{F}_x$, $\bar{x}$ belongs to $\bar{F}_x$. Furthermore, $\bar{x}$ has, at most, $nk-d$ coordinates with zero value. The reason is that, for $v \in V$ and $i \in [k]$, $\bar{x}_{v,i} = 0$ if and only if $x^0_{v,i} = x^1_{v,i} = \ldots = x^d_{v,i} = 0$. Hence, if $\bar{x}$ had at least $nk - d + 1$ coordinates with zero value, then $\bar{F}_x$ would be contained in a space with dimension at most $d-1$, a contradiction. Given that $d \geq nk-p+1$, it follows that at most $p-1$ coordinates of $\bar{x}$ are zero valued. Therefore, for any class $i \in [k]$, there exists $v \in V$, such that $w(v)$ and $\bar{x}_{v,i}$ are positive values.

For every $i \in [k]$, let $\bar{w}_i = \sum_{v \in V} w(v)\, \bar{x}_{v,i}$. Observe that $\bar{w}_i > 0$, for all $i \in [k]$. Define $\bar{w} = \min_{i \in [k]} \bar{w}_i$, then the vector $(\bar{x}, \bar{w})$ belongs to $\bar{F}$ and it is affinely independent with the vectors in Q (see Figure 5.12). ∎

Lemma 5.4.6 *Suppose $\pi x \leq \pi_0$ is an inequality that is valid for both $\mathcal{P}_k(G)$ and $\bar{\mathcal{P}}_x$. Let F and $\bar{F}_x$ be the faces induced by (π, π_0) in the polytopes $\mathcal{P}_k(G)$ and $\bar{\mathcal{P}}_x$, respectively. Then $\dim(F) \leq \dim(\bar{F}_x)$.*

Proof. Since $\mathcal{P}_k(G) \subseteq \bar{\mathcal{P}}_x$, all the $\dim(F) + 1$ affinely independent vectors that belong to F also belong to $\bar{F}_x$. ∎

Corollary 5.4.7 $\dim(\bar{\mathcal{P}}) = nk + 1$.

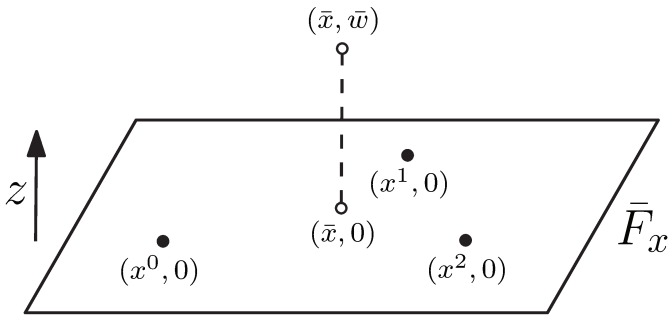


Figure 5.12: Constructing the vector $(\bar{x}, \bar{w})$ from vectors in $\bar{F}_x$ (or Q).

Proof. Consider applying Lemma 5.4.6 with the trivial inequality $\mathbf{0}x \leq 0$. The induced faces are $F = \mathcal{P}_k(G)$ and $\bar{F}_x = \bar{\mathcal{P}}_x$. Thus, $\dim(\bar{\mathcal{P}}_x) = nk$. Next, Proposition 5.4.5 gives us one extra affinely independent vector in $\bar{\mathcal{P}}$. □

Corollary 5.4.8 *For every $v \in V$ and $i \in [k]$, $x_{v,i} \geq 0$ induces a facet of $\bar{\mathcal{P}}$.*

Corollary 5.4.9 *For every $v \in V$, $\sum_{i\in[k]} x_{v,i} \leq 1$ induces a facet of $\bar{\mathcal{P}}$.*

Because formulation $\mathcal{C}'_k(G, w)$ is not ordering the classes by their weights, we prove that the connectivity inequalities induce facets of $\bar{\mathcal{P}}$, regardless if G is (u, v, S, i)-robust or not. Moreover, we analyze when the cross inequalities induce facets of $\bar{\mathcal{P}}$. The overall idea is similar: we first show that these inequalities induce facets of $\bar{\mathcal{P}}_x$, then we use Proposition 5.4.5 to claim that they also define facets of $\bar{\mathcal{P}}$.

Proposition 5.4.10 *Let u and v be non-adjacent vertices in G, let S be a minimal (u, v)-separator, and let $i \in [k]$. Then the inequality*

$$x_{u,i} + x_{v,i} - \sum_{s\in S} x_{s,i} \leq 1$$

induces a facet of $\bar{\mathcal{P}}$.

Proof. As usual, let $\hat{\lambda}x \leq 1$ denote the inequality $x_{u,i} + x_{v,i} - \sum_{z\in S} x_{z,i} \leq 1$. Consider the face $\hat{F} = \{x \in \bar{\mathcal{P}}_x \colon \hat{\lambda}x = 1\}$ and let $F = \{x \in \bar{\mathcal{P}}_x \colon \lambda x = \lambda_0\}$ be a nontrivial face such that $\hat{F} \subseteq F$. To show that each coefficient $\lambda_{z,j}$ has the appropriate value, we first consider $j = i$, and then $j \in [k] \setminus \{i\}$.

For any $z \in V$, recall the definition for G_z used in the last section. Let $e(G_z, i) = \sum_{v\in V(G_z)} e(v, i)$, in other words, $e(G_z, i)$ is the incidence vector of a solution in which all the vertices in G_z belong to the i-th class. As a consequence of how G_z was defined, $e(G_z, i)$ belongs to $\hat{F} \subseteq F$.

If $z \in V \setminus (S \cup \{u, v\})$, note that $(e(G_z, i) - e(z, i))$ also belongs to $\hat{F}$. Therefore, since $\lambda(e(G_z, i)) = \lambda_0$ and $\lambda(e(G_z, i) - e(z, i)) = \lambda_0$, it follows that $\lambda_{z,i} = 0$. Suppose now that $z \in S \cup \{u, v\}$. If $z \in \{u, v\}$, because both vectors $e(u, i)$ and $e(v, i)$ belong to $\hat{F} \subseteq F$, we get that $\lambda_{u,i} = \lambda_{v,i} = \lambda_0$. If $z \in S$, it follows from the previous cases that

$$\lambda(e(G_z, i)) = \lambda_{u,i} + \lambda_{v,i} + \lambda_{z,i} = \lambda_0.$$

Hence, $\lambda_{z,i} = -\lambda_0$.

Consider $j \in [k] \setminus \{i\}$. Since the vector $(e(u,i) + e(z,j))$ belongs to $\hat{F} \subseteq F$, it follows that $\lambda(e(u,i)+e(z,j)) = \lambda_0$, and thus $\lambda_{z,j} = 0$. Similarly, when $z = u$, $\lambda(e(v,i)+e(u,j)) = \lambda_0$ and $\lambda_{u,j} = 0$.

So far, we showed that $\hat{F}$ is a facet of $\bar{\mathcal{P}}_x$. Since $\bar{\mathcal{P}}_x$ is full-dimensional, $\dim(\hat{F}) = nk-1$. Using Proposition 5.4.5, we conclude that the inequality $x_{u,i}+x_{v,i}-\sum_{s\in S} x_{s,i} \leq 1$ induces a facet of $\bar{\mathcal{P}}$. □

Theorem 5.4.11 *Let $\{s_i, t_i\}_{i\in[2]} \subseteq V$ be a set of distinct pairs of vertices such that there is no collection of vertex-disjoint (s_i, t_i)-paths in G, for $i \in [2]$. Moreover, let $\sigma\colon [2] \to [k]$ be an injective function, and let I be its image. Then the cross inequality*

$$x_{s_1,\sigma(1)} + x_{t_1,\sigma(1)} + x_{s_2,\sigma(2)} + x_{t_2,\sigma(2)} \leq 3$$

induces a facet of $\bar{\mathcal{P}}$ if and only if the following conditions hold

1. *there is no vertex $z \in V$ such that, for every $i \in [2]$, z is a (s_i, t_i)-separator; and*
2. *$\{s_2, t_2\}$ is not a (s_1, t_1)-separator, and $\{s_1, t_1\}$ is not a (s_2, t_2)-separator.*

Proof. Let $\hat{F}$ be the face of $\bar{\mathcal{P}}_x$ induced by the inequality in the theorem statement. First, we demonstrate that if either condition (1) or (2) is not satisfied, $\hat{F}$ is not a facet of $\bar{\mathcal{P}}_x$.

Suppose there is a vertex z which is a (s_i, t_i)-separator, for every $i \in [2]$. Let $\tilde{x}$ be an integer vector that belongs to $\hat{F}$. Thus, there must exist $j \in [2]$, such that s_j and t_j are connected in the subgraph induced by the set of vertices $\{v \in V : \tilde{x}_{v,j} = 1\}$ and z belongs to this subgraph. Hence, $\hat{F} \subseteq \{x \in \bar{\mathcal{P}}_x\colon \sum_{i\in[k]} x_{z,i} = 1\}$ and $\hat{F}$ is not a facet.

Now suppose the set of vertices $\{s_2, t_2\}$ is a (s_1, t_1)-separator, then we shall demonstrate that $\hat{F} \subseteq \{x \in \bar{\mathcal{P}}_x\colon x_{s_1,\sigma(1)} + x_{t_1,\sigma(1)} - x_{s_2,\sigma(1)} - x_{t_2,\sigma(1)} = 1\}$. Again, let $\tilde{x}$ be an integer vector that belongs to $\hat{F}$. Let $i \in [2]$ be such that $\tilde{x}_{s_i,\sigma(i)} = \tilde{x}_{t_i,\sigma(i)} = 1$. If $i = 1$, assume without loss of generality that $\tilde{x}_{s_2,\sigma(1)} = 1$. Since $\tilde{x} \in \hat{F}$, t_2 belongs to class $\sigma(2)$ in $\tilde{x}$ and $\tilde{x}_{t_2,\sigma(1)} = 0$. On the other hand, if $i = 2$, $\tilde{x}_{s_2,\sigma(1)} = \tilde{x}_{t_2,\sigma(1)} = 0$ and exactly one vertex from $\{s_1, t_1\}$ belongs to class $\sigma(1)$ in the solution induced by $\tilde{x}$. Therefore, in both cases it holds that $x_{s_1,\sigma(1)} + x_{t_1,\sigma(1)} - x_{s_2,\sigma(1)} - x_{t_2,\sigma(1)} = 1$.

Next, we turn to proving that $\hat{F}$ is a facet if (1) and (2) are satisfied. In order to enumerate the appropriate vectors contained in $\hat{F}$, we start with a simple claim about paths that avoids specific vertices. All paths mentioned henceforth are simple.

Claim 5.4.12 *For any vertex $y \in V \setminus \{s_1, t_1, s_2, t_2\}$, there exists $a, b \in [2]$, $a \neq b$, such that there is a (s_a, t_a)-path $P_y^a \subset G$ that satisfy the properties: $y \notin V(P_y^a)$ and $|V(P_y^a) \cap \{s_b, t_b\}| \leq 1$.*

Proof. For some $i \in [2]$, there must exist a (s_i, t_i)-path P_y^i that does not contain y, otherwise, condition (2) would not hold. Let $j \in [2] \setminus \{i\}$, if $|P_y^i \cap \{s_j, t_j\}| \leq 1$ the claim holds with $a = i$ and $b = j$. Thus, assume that P_y^i contains both s_j and t_j. In this setting (see Figure 5.13), there is a (s_j, t_j)-path $P_y^j \subset P_y^i$ that does not intersect with $\{s_i, t_i, y\}$. □

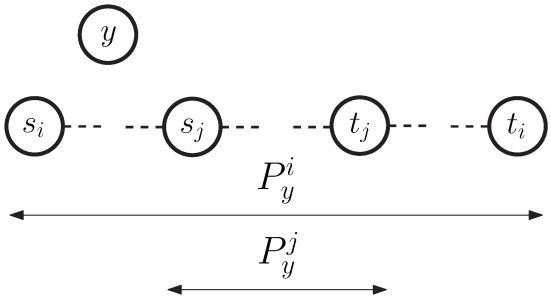


Figure 5.13: The path P_y^j if $|P_y^i \cap \{s_j, t_j\}| = 2$.

As usual, assume $\hat{F} = \{x \in \bar{\mathcal{P}}_x \colon \hat{\lambda}x = \hat{\lambda}_0\}$ and let $F = \{x \in \bar{\mathcal{P}}_x \colon \lambda x = \lambda_0\}$ be a face such that $\hat{F} \subseteq F$. We show that there exists a constant $c \in \mathbb{R}_>$ such that $\lambda = c\hat{\lambda}$ and $\lambda_0 = c\hat{\lambda}_0$. In other words, $\lambda = (\lambda_0/\hat{\lambda}_0)\hat{\lambda} = (\lambda_0/3)\hat{\lambda}$.

Once more, we break the analysis into cases. First, we demonstrate that $\lambda_{y,j} = 0$, for any vertex $y \in V \setminus \{s_i, t_i\}_{i\in[2]}$ and $j \in [k]$. Afterwards, we prove that $\lambda_{s_i,\sigma(i)} = \lambda_{t_i,\sigma(i)} = \lambda_0/3$ and $\lambda_{s_i,j} = \lambda_{t_i,j} = 0$, for $i \in [2]$ and $j \in [k] \setminus \{\sigma(i)\}$. In the forthcoming discussion, we use the notation $e(G', i) = \sum_{v\in V(G')} e(v, i)$, for any subgraph $G' \subseteq G$ and $i \in [k]$.

Consider the entries $\lambda_{y,j}$, where $y \in V \setminus \{s_i, t_i\}_{i\in[2]}$ and $j \in [k] \setminus I$. By Claim 5.4.12, there is $a, b \in [k]$ such that P_y^a is a (s_a, t_a)-path, $|V(P_y^a) \cap \{s_b, t_b\}| \leq 1$ and $y \notin V(P_y^a)$. Let u be a vertex in $\{s_b, t_b\} \setminus V(P_y^a)$, we define the vectors

$$\mu = e(P_y^a, \sigma(a)) + e(u, \sigma(b)) \qquad \text{and} \qquad \mu' = \mu + e(y, j).$$

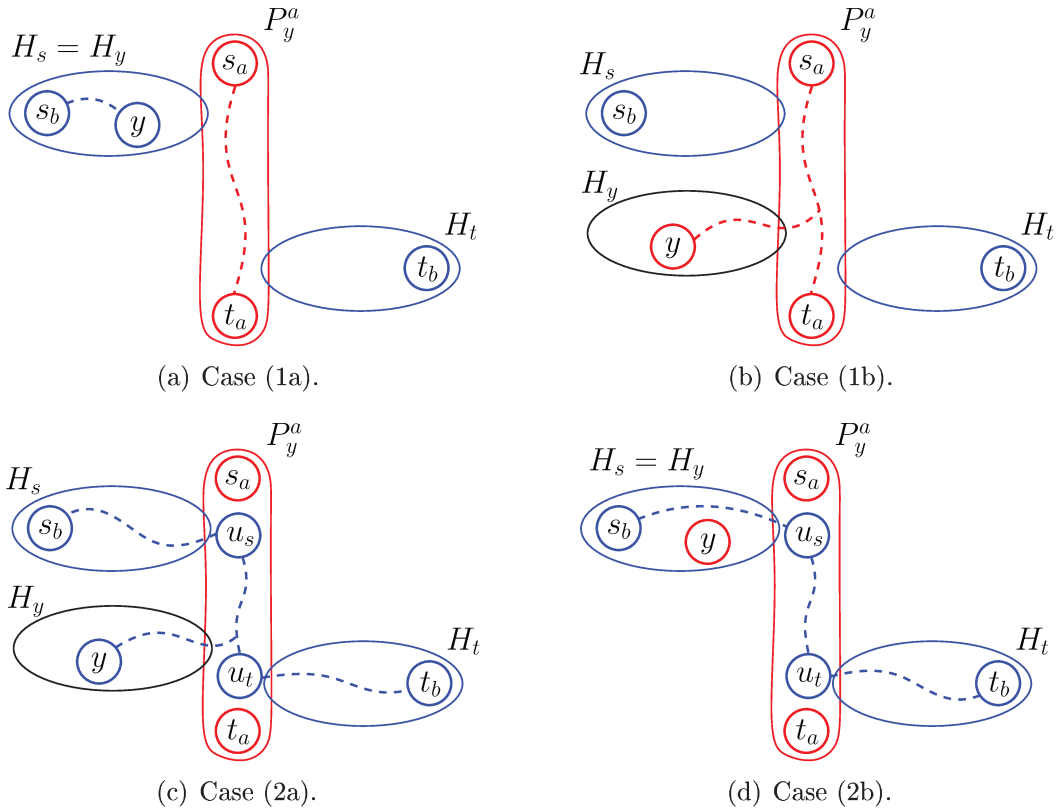
Clearly $\mu, \mu' \in \hat{F} \subseteq F$. Hence, solving the equations

$$\lambda\mu = \lambda_0 \qquad \text{and} \qquad \lambda\mu' = \lambda(\mu + e(y, j)) = \lambda_0,$$

we conclude $\lambda_{y,j} = 0$. This reasoning — where we obtain the value of a coordinate of λ by the construction of two vectors in F — will be repeated throughout the proof. Therefore, we present the next arguments in a more concise manner.

Choose a and b in a way that P_y^a satisfy the properties of Claim 5.4.12. Let H_y be the component of $G - P_y^a$ which contains y. Furthermore, let H_s and H_t be the components of $G - P_y^a$ that contains s_b and t_b, respectively. We define $H_s = \emptyset$ (resp. $H_t = \emptyset$) if $s_b \in V(P_y^a)$ (resp. $t_b \in V(P_y^a)$). In order to show that $\lambda_{y,j} = 0$ for $y \in V \setminus \{s_i, t_i\}_{i\in[2]}$ and $j \in I$, we separate the analysis into four cases:

- Cases of type 1.
  - (a) $H_y \in \{H_s, H_t\}$ and $j = \sigma(b)$,
  - (b) $H_y \notin \{H_s, H_t\}$ and $j = \sigma(a)$,
- Cases of type 2.
  - (a) $H_y \notin \{H_s, H_t\}$ and $j = \sigma(b)$,
  - (b) $H_y \in \{H_s, H_t\}$ and $j = \sigma(a)$.



(a) Case (1a). (b) Case (1b).

(c) Case (2a). (d) Case (2b).

Figure 5.14: The four cases in the proof. Red and blue represent class $\sigma(a)$ and $\sigma(b)$, respectively. The figures give us an idea of how we construct the desired vectors.

Note that, since condition (2) holds, if $V(H_s) \neq \emptyset$, $N(H_s)$ is not contained in $\{s_a, t_a\}$, that is, the neighborhood of H_s contains a vertex that is not one of the endpoints of P_y^a. Hence, we define u_s (resp. u_t) as a vertex in $N(H_s) \setminus \{s_a, t_a\}$ (resp. $N(H_t) \setminus \{s_a, t_a\}$). On the other hand, if $V(H_s) = \emptyset$, we define $u_s = s_b$ (resp. $u_t = t_b$).

Cases of type 1.

(1a) Suppose $H_y = H_s$ and $j = \sigma(b)$. Consider a path $P_s \subseteq H_s$ with endpoints at vertices y and s_b. Define vectors

$$\mu_s = e(P_y^a, \sigma(a)) + e(P_s, j) \qquad \text{and} \qquad \mu'_s = \mu_s - e(y, j).$$

One may easily verify that $\mu_s, \mu'_s \in \hat{F} \subseteq F$ (see Figure 5.14(a)). The case $H_y = H_t$ and $j = \sigma(b)$ can be addressed analogously.

(1b) Suppose $H_y \notin \{H_s, H_t\}$ and $j = \sigma(a)$. Let $P_y \subseteq H_y$ be a path that connects y to a vertex adjacent to P_y^a. Now the vectors $\mu + e(P_y, j)$ and $\mu + e(P_y, j) - e(y, j)$ belong to F (Figure 5.14(b)).

Cases of type 2.

(2a) If $H_y \notin \{H_s, H_t\}$ and $j = \sigma(b)$, we denote by $P_{s,t}$ the (u_s, u_t)-path contained in P_y^a. Let P_y^b be a (s_b, t_b)-path such that $V(P_y^b) \cap V(P_y^a) = V(P_{s,t})$. Note that $V(P_y^b) \cap \{s_a, t_a, y\} = \emptyset$. Let P'_y be a path that connects y to a vertex

adjacent to P_y^b, such that there exists a vertex $u' \in \{s_a, t_a\} \setminus V(P'_y)$. It follows from our construction that the vector $e(P_y^b, j) + e(P'_y, j) + e(u', \sigma(a))$ belongs to $F \subseteq F$ (see Figure 5.14), and as a result $\lambda_{y,j} = 0$.

(2b) Our last case is when $H_y \in \{H_s, H_t\}$ and $j = \sigma(a)$. Assume, with no loss of generality, that $H_y = H_s$. If there is a path P_y that connects y to the neighborhood of P_y^a and avoid s_b, we can apply the same reasoning as in Case (1b). Hence, we suppose s_b is a (y, u_s)-separator. Let P'_s be a path that connects s_b to a vertex in $N(u_s) \cap H_s$. Consider the path $P_{s,t}$ defined previously and let P_y^b be a (s_b, t_b)-path such that it contains $P_{s,t}$ and P'_s, and $V(P_y^b) \cap V(P_y^a) = V(P_{s,t})$. It follows from the construction that P_y^b does not intersect with $\{s_a, t_a, y\}$. Interchanging the roles of a and b and considering the path P_y^b instead of P_y^a, we reduce Case (2b) to Cases (1a) and (2a) (Figure 5.14(d)).

Finally, we now refer to entries $\lambda_{y,j}$, $y \in \{s_i, t_i\}_{i \in [2]}$. As a consequence of condition (2), for any $a, b \in [2]$, $a \neq b$, there must exist a (s_a, t_a)-path P^a such that $V(P^a) \cap \{s_b, t_b\} = \emptyset$. Hence, for any $j \in [k] \setminus \{\sigma(b)\}$, it holds that the vector $e(P^a, \sigma(a)) + e(t_b, \sigma(b)) + e(s_b, j)$ belongs to F. Therefore, $\lambda_{s_i,j} = 0$, for any $i \in [2]$ and $j \in [k] \setminus \{\sigma(i)\}$.

Consider now the vectors $e(P^1, \sigma(1)) + e(s_2, \sigma(2)), e(P^1, \sigma(1)) + e(t_2, \sigma(2))$ and their counterparts $e(P^2, \sigma(2)) + e(s_1, \sigma(1)), e(P^2, \sigma(2)) + e(t_1, \sigma(1))$. Because all these vectors belong to F and given our previous analysis for the zero valued entries of λ, we end up with the following system of linear equations.

$$\lambda_{s_1,\sigma(1)} + \lambda_{t_1,\sigma(1)} + \lambda_{s_2,\sigma(2)} = \lambda_0, \tag{5.13}$$

$$\lambda_{s_1,\sigma(1)} + \lambda_{t_1,\sigma(1)} + \lambda_{t_2,\sigma(2)} = \lambda_0, \tag{5.14}$$

$$\lambda_{s_2,\sigma(2)} + \lambda_{t_2,\sigma(2)} + \lambda_{s_1,\sigma(1)} = \lambda_0, \tag{5.15}$$

$$\lambda_{s_2,\sigma(2)} + \lambda_{t_2,\sigma(2)} + \lambda_{t_1,\sigma(1)} = \lambda_0. \tag{5.16}$$

Subtracting (5.16) from (5.15) we obtain that $\lambda_{s_1,\sigma(1)} = \lambda_{t_1,\sigma(1)}$. Equivalently, it also holds that $\lambda_{s_2,\sigma(2)} = \lambda_{t_2,\sigma(2)}$. Moreover, subtracting (5.13) from (5.15) gives us $\lambda_{t_2,\sigma(2)} = \lambda_{t_1,\sigma(1)}$. Therefore, we conclude that

$$\lambda_{s_1,\sigma(1)} = \lambda_{t_1,\sigma(1)} = \lambda_{s_2,\sigma(2)} = \lambda_{t_2,\sigma(2)} = \frac{\lambda_0}{3}.$$

□

Chapter 6

Computational Experiments

The computational experiments were carried out on a PC with Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz, 40 cores, 64 GB RAM and Ubuntu 18.04.2 LTS. The code was written in C++ using the graph library Lemon [14]. We implemented a branch-and-cut algorithm based on the Cut formulation $\mathcal{C}$ using SCIP 6.0 [19] and Gurobi 9.0 [23] as the LP solver. We also implemented branch-and-bound algorithms (using only Gurobi 9.0) based on the Flow formulations $\mathcal{F}$ and $\bar{\mathcal{F}}$ and on the models previously proposed by Matić [37] and Zhou et al. [50]. Each algorithm we implemented had 4 threads available to it. We used SCIP 6.0 in our branch-and-cut implementation for three reasons. First, unlike Gurobi 9.0, SCIP allows for multiple rounds of cut generation in non-root nodes of the branch-and-bound tree. Second, it has built-in routines for separating the lifted minimal cover inequalities mentioned in Chapter 5. Third, it allows the user to write a custom domain propagation routine, as the one mentioned in Section 5.3.

Due to small improvements in the preliminary experiments, we replace inequalities (4.5), (4.10) and (5.2) with equalities. Furthermore, to evaluate strictly the performance of the mentioned formulations, all standard cuts used by SCIP and Gurobi are deactivated, except for the lifted minimal cover inequalities.

Finally, for each of the formulations we proposed in this work, we also implemented their correspondent z-versions. In other words, we implemented a branch-and-cut algorithm for $\mathcal{C}'$ and branch-and-bound algorithms for $\mathcal{F}'$ and $\bar{\mathcal{F}}'$.

6.1 Benchmark instances

Computational experiments on instances consisting of grid graphs and random connected graphs are reported in [37,50]. In this work, besides evaluating our algorithms on instances previously proposed in the literature, we also considered larger graphs, different weight distributions and real world instances.

The grid instances are named in the format *gg_height_width_*[a|b|c] and the random instances have names in the format *rnd_n_m_*[a|b|c], where n is the number of vertices and m is the number of edges in the graph. The letters 'a', 'b' and 'c', indicate that the weights of that instance are integers obtained uniformly from the intervals $[1, 100]$, $[1, 500]$ and $[1000, 5000]$, respectively.

In all of these instances, the weights are integers uniformly distributed in three intervals, named a, b and c, indicated in the end of the instance names: a $= [1, 100]$, b $= [1, 500]$ and c $= [1000, 5000]$.

In order to generate a random connected graph with n vertices and m edges (with $m > n-1$), we first use Wilson's algorithm [47] to generate a uniformly random spanning tree T on n vertices, and then add $m - n + 1$ distinct new edges selected randomly from $E(K_n) \setminus E(T)$ with uniform probability. Wilson's algorithm returns a spanning tree T sampled from the set τ_n — of all possible spanning trees of K_n — with probability $1/|\tau_n|$.

In the experiments, for each format (considered a graph class) indicated in the tables and plots, we generated 10 random instances. The randomness of grid instances refer to their weights, and of random graphs refers to the graphs and the weights.

Finally, we also created instances based on a real-world application, namely demarcation of preventive police patrol areas [4]. This problem consists in subdividing a given map into k (contiguous) regions such that every region has roughly the same crime rate. Each of those regions is assigned to a police patrol team. Clearly, this problem can be modeled as BCP_k.

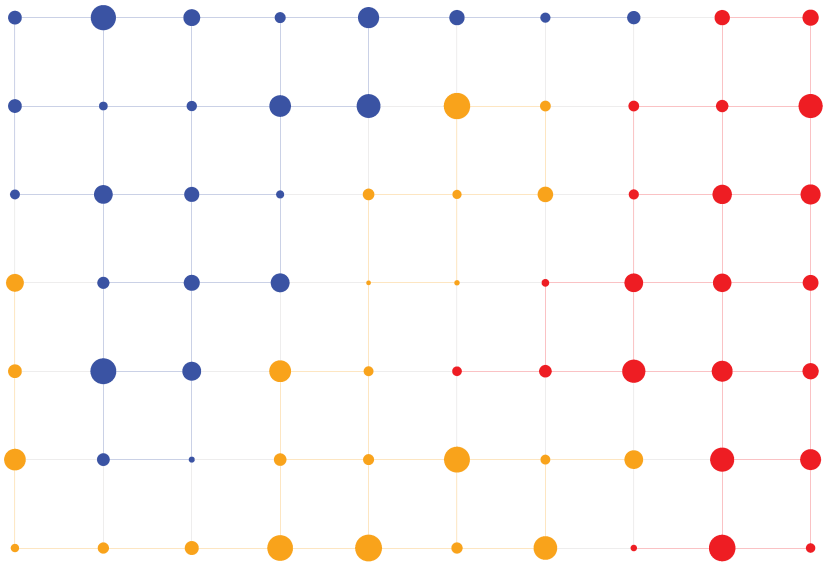
Using the OSMnx [8] library we transformed maps from OpenStreetMap [43] into undirected graphs. The edges in the graphs generated by OSMnx correspond to sections of the streets. As some of these edges may be too "long" (over 200 meters), we subdivide long edges into smaller edges so that the length of each edge is limited to 200 meters.

Working with the Socrata Open Data API, we downloaded the Public Safety data for the cities of Chicago, Los Angeles and New York; and using the transparency website of the Department of Public Safety of São Paulo, we downloaded data for the city of Campinas. The police patrolling instances that we generated have names in the format *name*_ n_ m, where *name* denotes the name of the corresponding geographic region, $n = |V|$ and $m = |E|$.

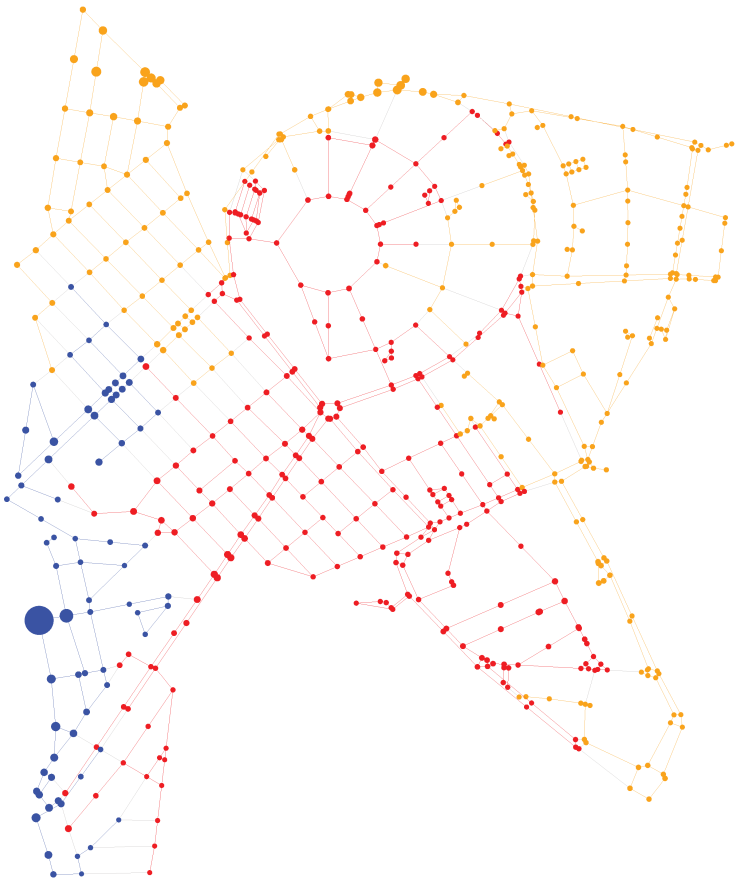
For each vertex v of a graph generated from a map, we assign a weight that is proportional to the crime rate geographically close to the point in the map associated with v. More precisely, let $G = (V, E)$ be a graph corresponding to a region of a city and C be a set of points of this region where crimes have occurred. Let $d\colon C \times V \to \mathbb{Q}_{\geq}$ be a function that computes the distance (in meters) of a crime to a vertex, and $f\colon \mathbb{R} \to \mathbb{R}$ be the normal probability density function with mean $\mu = 0$ and standard deviation $\sigma = 0.5$. We consider that a crime has influence on the weights of the vertices that are within a radius of 200 meters from it. So, for each point $c \in C$, we define $V_c = \{v \in V\colon d(c, v) \leq 200\}$, and $F_c = \sum_{v \in V_c} f(d(c, v)/200)$. Then, for each vertex $v \in V$, we set its weight as

$$w(v) = \left\lfloor 100 \sum_{c\colon v \in V_c} \frac{f(d(c, v)/200)}{F_c} \right\rfloor.$$

Note that the formula used to define the weight of a vertex agrees with the notion that the influence of a crime over a region is a Gaussian distribution on the distance to the crime.



(a) A 7 by 10 grid graph instance.



(b) A police patrolling instance, based on the University of Campinas campus.

Figure 6.1: Instances of BCP_3 and optimal solutions. The radius of each vertex is proportional to its weight. The vertices in V_1 are colored red.

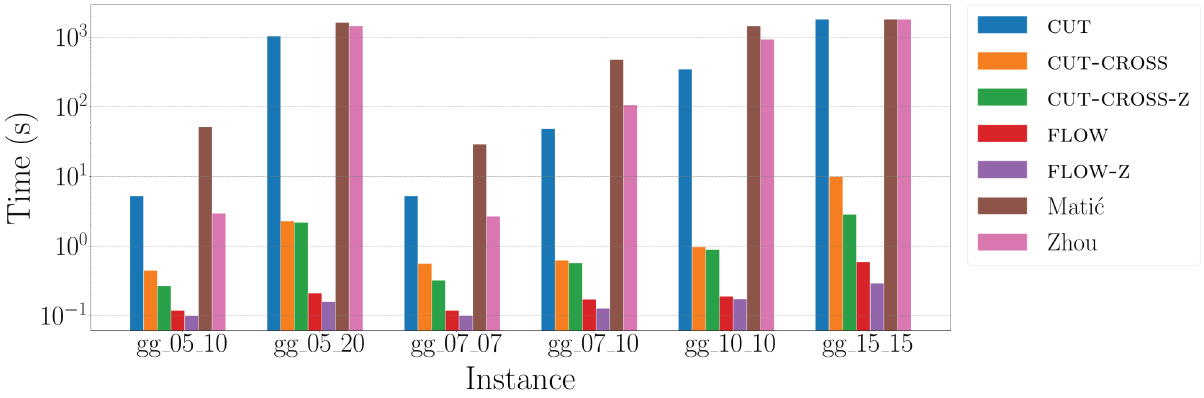


Figure 6.2: Computational results for BCP_2 on grid graphs. Time is in logarithmic scale.

6.2 Computational results

The execution time limit was set to 1800 seconds. In the following tables, we show the average number of explored nodes in the branch-and-bound tree (column Nodes) and the average time, in seconds, to solve the instances (column Time), ignoring the unsolved instances. When the time limit was exceeded for all 10 instances of a graph class, we set the corresponding table entries with a dash (-). In a row, when all 10 instances of a graph class were solved, the (average) time that is minimum is indicated in boldface. Similarly, when the number of nodes that were explored is minimum, we underline the corresponding value.

Henceforth, when we refer to any of the formulations it should be understood that we are referring to the corresponding exact algorithms that we have implemented for them. Thus, CUT corresponds to the branch-and-cut algorithm based on formulation $\mathcal{C}$, while FLOW corresponds to the branch-and-bound algorithm based on $\mathcal{F}$. The algorithms based on the Z-versions of the formulations have names with suffix "-Z". It goes without saying that CUT-Z cannot take advantage of the lifting we proposed for the connectivity inequalities in Proposition 5.1.3. We omit the results for the algorithms based on $\bar{\mathcal{F}}$ and $\bar{\mathcal{F}}'$ because, in our experiments, they were (on average) over 10 times slower than FLOW.

We start by showing in Table 6.1 the impact of separating cross inequalities. Column CUT-CROSS refers to CUT with the cross inequalities. Columns Conn Cuts and Cross Cuts show the average number of connectivity and cross inequalities separated by the algorithm. Notice that, on average, CUT-CROSS was much faster than CUT on all grid instances. Furthermore, only CUT-CROSS was able to solve all the instances with more than 100 vertices within the time limit.

Instead of showing the complete tables (which can be seen in the appendix), we selected some representative instances with the weight distribution of type a = $[1, 100]$ and plotted the execution time in a histogram. This way, the reader can easily visualize the bulk of our results without carefully (and painfully) analyzing each row of the tables. Figures 6.2 and 6.3 show histograms with the average execution time — note the y-axis is in logarithmic scale. Unlike the tables, when computing the average for these plots, we assume that an algorithm took 1800 seconds to solve instances which were not solved

	CUT			CUT-CROSS			
Instance	Sol	Conn Cuts	Time	Sol	Conn Cuts	Cross Cuts	Time
gg_05_05_a	10	3,504	0.50	10	449	478	**0.15**
gg_05_05_b	10	24,252	3.32	10	2,817	3,841	**0.84**
gg_05_05_c	10	110,142	16.22	10	9,852	14,505	**3.77**
gg_05_06_a	10	6,184	0.80	10	1,056	1,286	**0.32**
gg_05_06_b	10	21,125	2.76	10	3,311	4,536	**1.02**
gg_05_06_c	10	156,230	22.97	10	12,587	19,210	**4.94**
gg_05_10_a	10	44,635	5.63	10	2,187	2,147	**0.55**
gg_05_10_b	10	67,926	9.37	10	3,776	3,884	**1.10**
gg_05_10_c	10	358,226	55.92	10	14,897	23,184	**7.80**
gg_05_20_a	6	2,821,710	521.39	10	9,578	8,768	**2.43**
gg_05_20_b	10	1,836,157	325.73	10	23,832	23,613	**6.47**
gg_05_20_c	8	3,590,827	751.55	10	30,836	34,307	**14.43**
gg_07_07_a	10	41,279	6.84	10	2,128	2,220	**0.78**
gg_07_07_b	10	184,730	30.99	10	3,035	3,647	**1.29**
gg_07_07_c	10	896,587	167.58	10	19,739	28,066	**10.11**
gg_07_10_a	10	301,289	57.34	10	2,154	1,942	**0.81**
gg_07_10_b	10	427,441	87.24	10	5,954	6,243	**2.69**
gg_07_10_c	10	1,715,120	390.97	10	31,894	44,412	**31.56**
gg_10_10_a	10	1,373,429	389.42	10	2,987	2,432	**1.20**
gg_10_10_b	9	1,289,465	370.06	10	3,652	3,343	**2.12**
gg_10_10_c	9	1,373,725	406.28	10	28,492	32,264	**28.53**
gg_15_15_a	0	-	-	10	13,032	7,371	**8.97**
gg_15_15_b	0	-	-	10	15,411	10,182	**13.88**
gg_15_15_c	0	-	-	10	70,108	52,280	**106.23**

Table 6.1: Computational results for BCP_2 on grid graphs showing the efficiency of the cross inequalities.

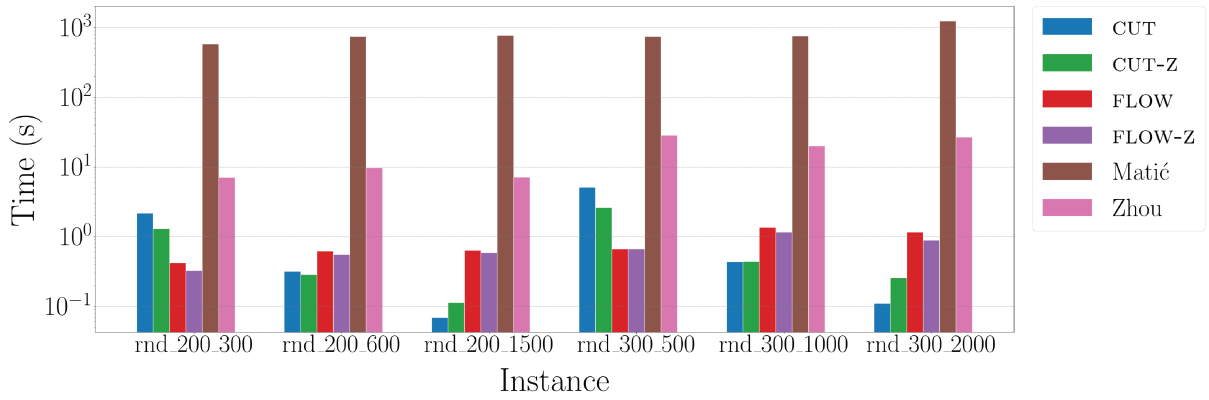


Figure 6.3: Computational results for BCP_2 on random graphs. Time is in logarithmic scale.

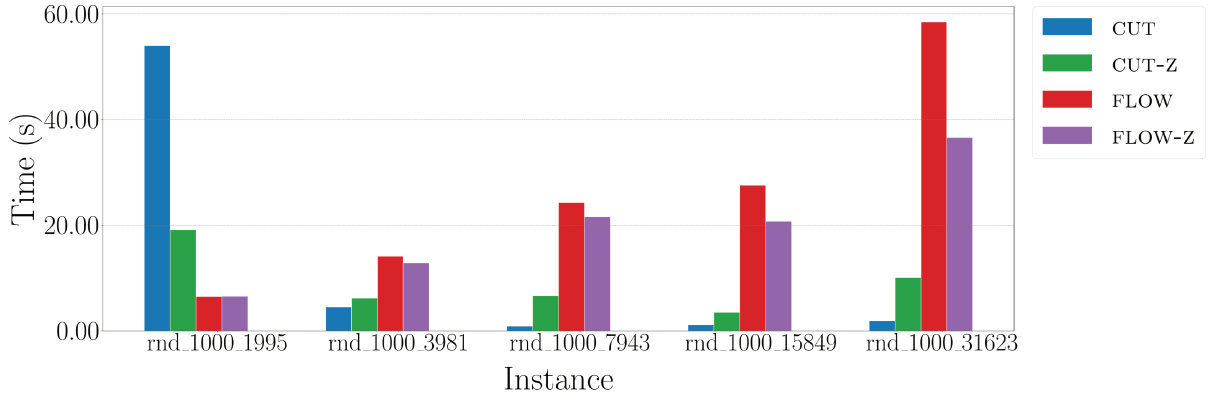


Figure 6.4: Experiments for BCP_2 on random graphs with 1000 vertices and different densities. Time is in linear scale.

within the time limit.

As can be seen, our best algorithms significantly outperform previous exact methods. Figure 6.2 illustrates that FLOW had better execution times than CUT-CROSS on most of the grid instances. Moreover, on grids with over 100 vertices, Matić and Zhou formulations were not able to solve the majority of the instances, while CUT-CROSS and FLOW solved all of them. Furthermore, Figure 6.3 shows that both of our formulations are also faster (on average) on random graphs instances. The execution time for CUT was better than FLOW on some of the random graphs. Looking more closely at the number of vertices and edges, we noticed that the density of edges on the input graphs was a crucial factor for the performance of both algorithms.

In order to further explore the influence of the density of edges, we generated random graphs with a greater number of vertices ($n = 500$ and $n = 1000$) and different values for density ($m = n^{\alpha}$, where α varies between $\{1.1, 1.2, \ldots, 1.5\}$). The experiments on instances with 1000 vertices are visually represented in Figure 6.4. Table A.4 (in the appendix) shows that random graphs with 500 vertices exhibited a similar trend. Observe that in this histogram, time is in linear scale, since when doing comparisons between our own algorithms the difference in execution times is less prominent. Our experiments indicate that algorithms based on Cut formulation perform better than algorithms based on flow formulation when $m \geq n^{1.2}$. One fact that might explain this behavior is that a higher percentage of the vertices are adjacent in graphs with greater density. Thus, because we do not separate connectivity inequalities for adjacent vertices, the algorithms might spend less time in the separation routines. Moreover, as we increase the density of the graphs, it becomes easier to guarantee the connectedness of the classes. Thus, fewer connectivity inequalities might be violated in the branch-and-cut algorithms. Indeed, this was observed in our experiments (and shown in Table A.4): increasing the number of edges reduced drastically the number of separated connectivity inequalities, in a way that once m gets greater or equal to $n^{1.4}$, no connectivity inequality is separated at all.

Looking at Figures 6.2 and 6.3 we also observe that, on some classes of graphs, ordering the classes by their weights was not beneficial. In fact, the Z-versions of the algorithms were, on average, slightly faster than their correspondent ordering counterparts. However,

as the size of the grids grows, the number of symmetric solutions also increases; thus, on larger grids, ordering the classes might improve the execution times. Such a behavior can be seen in Table 6.2, where we show instances of grid graphs with many vertices that could only be solved by FLOW and FLOW-Z. Notice how FLOW was superior on almost all the rows of the table.

	FLOW			FLOW-Z		
Instance	Sol	Nodes	Time	Sol	Nodes	Time
gg_30_30_a	10	466	2.71	10	333	**2.15**
gg_30_30_b	10	731	3.14	10	542	**2.61**
gg_30_30_c	10	2,064	**5.09**	10	5,259	9.57
gg_60_60_a	10	420	20.06	10	318	**12.30**
gg_60_60_b	10	751	**19.49**	10	883	20.30
gg_60_60_c	10	2,810	56.19	10	861	**25.17**
gg_90_90_a	10	727	**89.21**	10	483	130.51
gg_90_90_b	10	874	**88.78**	10	327	203.92
gg_90_90_c	10	1,796	116.32	10	596	**102.75**
gg_120_120_a	10	575	**148.93**	10	143	626.39
gg_120_120_b	10	736	**207.57**	10	164	496.14
gg_120_120_c	10	1,077	**178.66**	10	242	254.92
gg_150_150_a	10	546	**302.97**	5	230	1281.25
gg_150_150_b	10	1,087	**334.19**	7	129	997.63
gg_150_150_c	10	1,226	**383.69**	9	197	643.20
gg_180_180_a	10	305	**571.58**	10	599	717.65
gg_180_180_b	10	451	**599.53**	9	374	1112.19
gg_180_180_c	10	784	**716.84**	10	1,321	938.36
gg_210_210_a	10	400	**1100.32**	10	298	1117.10
gg_210_210_b	9	325	**1064.07**	10	194	1172.61
gg_210_210_c	10	1,199	**1232.04**	3	1,482	1491.76

Table 6.2: Performance of FLOW to solve BCP_2 on large grids.

In our experiments, FLOW and FLOW-Z were the only algorithms able to solve the police patrolling instances within the time limit; with the latter algorithm having a better overall performance. As Table 6.3 indicates, the problem becomes harder to solve as the value of k increases.

In this sense, we also carried out experiments for $k > 2$ on grid graphs and random graphs, as shown in Table 6.4. To compare with the formulation proposed by Zhou et al., which is intended to MIN-MAX BCP_k, we considered cut and flow formulations with min-max objective (i.e. minimizing the weight of the k-th class). These algorithms are denoted by the string "(MIN-MAX)" at the end of their names. Since FLOW-Z (MIN-MAX) had the best performance in these instances, we omit the results for the other algorithms in Table 6.4 (the interested reader can check the Table A.5 for the expanded results). We remark that although Zhou et al.'s branch-and-bound solved one specific instance in 508.93 seconds, FLOW-Z (MIN-MAX) solved 5 more instances of the same graph class.

Instance	2	3	4	5	6
barao_1913_2752	40.171	438.481	1658.954	-	-
campinas_centro_579_942	1.998	53.857	-	185.487	-
chicago_englewood_1560_2579	29.872	60.041	300.252	1649.001	-
chicago_lakeview_1004_1563	2.997	32.129	146.288	154.851	-
chicago_loop_624_971	1.772	52.727	57.769	132.141	-
la_hollywood_1368_2030	16.444	194.314	222.594	1108.832	-
la_skidrow_1667_2459	6.956	73.676	1142.841	-	-
nyc_chelsea_822_1228	2.72	25.885	125.357	147.829	1637.057
nyc_hellskitchen_498_746	1.168	11.511	155.053	303.183	-
unicamp_624_901	1.974	10.524	58.919	1663.362	-

Table 6.3: Running time of FLOW-Z on police patrolling instances.

		FLOW-Z (MIN-MAX)			Zhou et al.		
Instance	k	Sol	Nodes	Time	Sol	Nodes	Time
gg_07_10_a	3	10	1,654	**0.58**	2	1,072,107	603.97
gg_07_10_a	4	10	13,488	**2.47**	1	99,880	68.95
gg_07_10_a	5	10	208,118	**38.01**	0	-	-
gg_07_10_a	6	7	1,976,217	417.59	0	-	-
rnd_100_150_a	3	10	586	**0.41**	9	109,799	70.04
rnd_100_150_a	4	10	7,572	**2.19**	4	1,019,886	641.36
rnd_100_150_a	5	10	188,771	**74.49**	0	-	-
rnd_100_150_a	6	6	1,516,926	838.78	1	205,247	508.93

Table 6.4: Computational results for MIN-MAX BCP$_k$ when $k \in \{3, 4, 5, 6\}$.

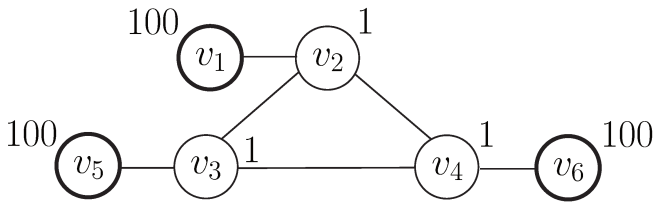


Figure 6.5: Instance of BCP_2 with $w(v_1) = w(v_5) = w(v_6) = 100$ and $w(v_2) = w(v_3) = w(v_4) = 1$.

Our computational experiments show that the algorithms based on the formulations we presented substantially outperform the previous exact methods in the literature. In most of the tested instances, algorithms based on the flow formulations had the best average execution time. On the other hand, the algorithms based on the cut formulations explored (on average) a smaller number of nodes in the branch-and-bound tree (see the appendix) and were the fastest on dense graphs.

6.3 On the linear relaxation bounds

Table A.1 (in the appendix) shows that FLOW solved only 4 (and FLOW-Z solved only 6) out of the 10 instances of class gg_05_05_c. Examining closely, we noticed that, when the optimal solution cost is considerably smaller than the trivial upper bound of $w(G)/k$, FLOW frequently fails to find a provably optimal solution within the time limit. In this section, we elaborate on this matter by looking at the strength of formulations $\mathcal{C}$ and $\mathcal{F}$. In order to do so, we compare the cost of optimal (integral) solutions with the cost of optimal solutions of the corresponding linear relaxations.

It is not difficult to see that for any instance of BCP_k, an optimal solution for the linear relaxation of $\mathcal{F}_k(G, w)$ and also of $\mathcal{C}_k(G, w)$ has cost $w(G)/k$, a trivial upper bound for the optimal value.

Let us first show that this happens for the linear relaxation of $\mathcal{F}_k(G, w)$. For that, take the vector $(\tilde{f}, \tilde{y}) \in \mathbb{R}^{kn+2m} \times \mathbb{B}^{kn+2m}$, defined as follows. For each source s_i, $i \in [k]$, and each arc (s_i, v), we set $\tilde{f}_{s_iv} = w(v)/k$ and $\tilde{y}_{s_iv} = w(v)/(kw(G))$. For every arc a not incident to a source, we set $\tilde{f}_a = \tilde{y}_a = 0$. Clearly, $(\tilde{f}, \tilde{y})$ is a feasible solution, and $\tilde{f}(\delta^+(s_i)) = w(G)/k$ for every $i \in [k]$.

Now, consider the linear relaxation of $\mathcal{C}_k(G, w)$. It is immediate that the vector $\tilde{x} \in \mathbb{R}^{nk}$ defined as $\tilde{x}_{v,i} = 1/k$, for each $v \in V$ and each $i \in [k]$, is a feasible solution for this relaxation, and moreover, $\sum_{v \in V} w(v)\tilde{x}_{v,i} = w(G)/k$ for each $i \in [k]$. Thus, $\tilde{x}$ is an optimal solution.

This linear relaxation may be strengthened by adding the cover inequalities mentioned in Section 5.3.1. As an example, consider the instance (G, w) of BCP_2 illustrated in Figure 6.5. Note that $\{v_1, v_5\}$ is a cover since $w(v_1) + w(v_5) > w(G)/2$, and thus the lifted cover inequality $x_{v_1,1} + x_{v_5,1} + x_{v_6,1} \leq 1$ is valid for the polytope associated with the linear relaxation of $\mathcal{C}_2(G, w)$. Such an inequality cuts off any optimal solution for the linear relaxation of $\mathcal{C}_2(G, w)$ whose cost is $w(G)/2 = 151.5$, as we observed before.

To get a better understanding of the effectiveness of the lifted cover inequalities, we

constructed instances for BCP_k consisting of grid graphs in which all the vertices are assigned unit weights, except for exactly $(k+1)$ random vertices that have a given (large) weight $p > 1$. Hence, the gap between an optimal integer solution and an optimal fractional solution for the flow-based formulation can be arbitrarily large on these instances. In our experiments, we set $p = 100$ and generated 10 instances consisting of grid graphs with height 5 and width 10. FLOW could not solve these instances within a time limit of 1800 seconds, while CUT (with the lifted cover inequalities) solved each of them in less than 1 second.

This gives an evidence that the lifted cover inequalities are useful in cutting off fractional optimal solutions of the linear relaxation of the cut-based formulation, and yield a better approximation of the convex hull.

Chapter 7

Conclusion

We proposed two MILP and one ILP formulation for the Balanced Connected k-Partition Problem. These formulations are all based on the idea of imposing an ordering of the classes $\{V_i\}_{i\in[k]}$, such that $w(V_i) \leq w(V_{i+1})$, for all $i \in [k-1]$. Therefore, one may easily modify the objective function to capture diverse concepts of "balance", such as minimize the heaviest class or the maximum difference of weights between the classes.

The first two formulations we introduced are based on flows in a digraph created from the input graph. The first of them, $\mathcal{F}$, has a polynomial number of variables and constraints. To overcome the apparent disadvantages of this formulation, like symmetries and dependency on the weights of the vertices, we designed $\bar{\mathcal{F}}$. Although more complex than the former, this formulation avoids some symmetries and has less dependency on the weights of the vertices. However, in our computational experiments, the performance of the branch-and-bound algorithm based on formulation $\mathcal{F}$ was significantly superior to the one based on $\bar{\mathcal{F}}$.

The third of our formulations, denoted by $\mathcal{C}$, is defined on the input graph and has an exponential number of connectivity inequalities, to which we presented a polynomial-time separation algorithm and a lifting procedure. Moreover, we introduced a new class of valid inequalities for this formulation, and showed how to separate a special case of them (cross inequalities) on planar graphs in polynomial time. The experiments showed that the addition of these inequalities improves greatly the performance of the corresponding branch-and-cut algorithm.

We also considered the impact of ordering the classes by their weights. To this end, we derived formulations $\mathcal{F}'$, $\bar{\mathcal{F}}'$ and $\mathcal{C}'$, which add a real variable z and avoid such an ordering. Although these Z-versions of the formulations have additional symmetries, computational experiments demonstrated that, in some cases, they might yield faster algorithms. In particular, our experiments seem to indicate that ordering is beneficial only for instances with over 10^4 vertices. Further investigation is still needed to fully understand the effectiveness of breaking symmetries.

When the vertices have uniform weight, we characterized that some inequalities of the model define facets of the associated polytope. Moreover, we also conducted a polyhedral study for the Z-version of Cut formulation, namely $\mathcal{C}'$. To the best of our knowledge, no previous polyhedral study was reported for BCP_k or MIN-MAX BCP_k.

Formulation	# Binary Variables	# Real Variables	# Constraints
Cut $\mathcal{C}$	kn	0	$\mathcal{O}(2^n)$
Flow $\mathcal{F}$	$\mathcal{O}(kn+m)$	$\mathcal{O}(kn+m)$	$\mathcal{O}(kn+m)$
Asymmetric Flow $\bar{\mathcal{F}}$	$\mathcal{O}(k(n+m))$	$\mathcal{O}(k(n+m))$	$\mathcal{O}(k(n^2+m))$
Matic [37]	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$
Zhou et al. [50]	$\mathcal{O}(k(n+m))$	$\mathcal{O}(n+m)$	$\mathcal{O}(km)$

Table 7.1: Formulations in terms of the amount of variables and constraints for an instance of BCP_k with n vertices and m edges.

Table 7.1 summarizes the number of variables and constraints in the formulations proposed in this work.

The computational experiments indicated that the methods proposed in this thesis have a considerably better performance than the previous exact algorithms in the literature. In particular, FLOW was able to solve grids with over 400 times more vertices than the grids solved by the previous exact solving methods in the literature. Moreover, only the algorithms based on flow formulation were able to solve real-world instances. Hence, in a practical context, if one wants to find an optimal solution for an instance of BCP_k, one could begin by carrying out experiments with FLOW. Not only FLOW is easier to implement than the other proposed algorithms, but it also had the most promising computational results.

Although CUT was more efficient on random graphs with a higher density of edges, FLOW had a superior behavior overall. Therefore, future work is still needed in order to make CUT more competitive. For example, one could investigate new valid inequalities and design (if needed) efficient separation routines. Furthermore, our experiments show that our formulations had difficulties for larger values of k. New ideas and formulations may be needed to solve efficiently in such cases. Finally, we only conducted a polyhedral study based on Cut formulation. One could also investigate the facial structure of the polytope associated with $\mathcal{F}$, however such a polytope may not be full-dimensional, which might give rise to technical difficulties.

All the instances we used in our computational experiments will soon be available at the website of the Laboratory of Optimization and Combinatorics at the University of Campinas [1].

[1] https://www.loco.ic.unicamp.br/files/instances

Bibliography

[1] P. Alimonti and T. Calamoneri. On the complexity of the max balance problem. In *Argentinian Workshop on Theoretical Computer Science (WAIT'99)*, pages 133–138, 1999.

[2] Mattias Andersson, Joachim Gudmundsson, Christos Levcopoulos, and Giri Narasimhan. Balanced partition of minimum spanning trees. In Peter M. A. Sloot, Alfons G. Hoekstra, C. J. Kenneth Tan, and Jack J. Dongarra, editors, *Computational Science — ICCS 2002*, pages 26–35. Springer Berlin Heidelberg, 2002.

[3] David Applegate, Robert Bixby, William Cook, and Vasek Chvátal. On the solution of traveling salesman problems. 1998.

[4] Thiago Assunção and Vasco Furtado. A heuristic method for balanced graph partitioning: An application for the demarcation of preventive police patrol areas. In Hector Geffner, Rui Prada, Isabel Machado Alexandre, and Nuno David, editors, *Advances in Artificial Intelligence (IBERAMIA 2008)*, pages 62–72, 2008.

[5] Eduardo Uchoa Barboza. Problemas de classificação com restrições de conexidade flexibilizadas. Master's thesis, Universidade Estadual de Campinas, 1997.

[6] Ronald I. Becker, Isabella Lari, Mario Lucertini, and Bruno Simeone. Max-min partitioning of grid graphs into connected components. *Networks*, 32(2):115–125, 1998.

[7] Ronald I Becker and Yehoshua Perl. Shifting algorithms for tree partitioning with general weighting functions. *Journal of Algorithms*, 4(2):101–120, 1983.

[8] Geoff Boeing. OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017.

[9] Ralf Borndörfer, Ziena Elijazyfer, and Stephan Schwartz. Approximating balanced graph partitions. Technical Report 19-25, ZIB, Takustr. 7, 14195 Berlin, 2019.

[10] Frédéric Chataigner, Liliane R. B. Salgado, and Yoshiko Wakabayashi. Approximation and inapproximability results on balanced connected partitions of graphs. *Discrete Mathematics & Theoretical Computer Science*, 9(1), 2007.

[11] Guangting Chen, Yong Chen, Zhi-Zhong Chen, Guohui Lin, Tian Liu, and An Zhang. Approximation algorithms for the maximally balanced connected graph tripartition problem. *Journal of Combinatorial Optimization*, 2020.

[12] Janka Chlebíková. Approximating the maximally balanced connected partition problem in graphs. *Information Processing Letters*, 60(5):225–230, 1996.

[13] Marcus Poggi de Aragão and Eduardo Uchoa. The γ-connected assignment problem. *European Journal of Operational Research*, 118(1):127–138, 1999.

[14] Balázs Dezső, Alpár Jüttner, and Péter Kovács. Lemon – an open source C++ graph template library. *Electronic Notes in Theoretical Computer Science*, 264(5):23–45, 2011.

[15] M.E. Dyer and A.M. Frieze. On the complexity of partitioning graphs into connected subgraphs. *Discrete Applied Mathematics*, 10(2):139–153, 1985.

[16] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

[17] Carlos Eduardo Ferreira and Yoshiko Wakabayashi. *Combinatória poliédrica e planos-de-corte faciais*. UNICAMP-Instituto de Computacao, 1996.

[18] Greg N. Frederickson. Optimal algorithms for tree partitioning. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '91, page 168–177, 1991.

[19] Ambros Gleixner, Michael Bastubbe, Leon Eifler, Tristan Gally, Gerald Gamrath, Robert Lion Gottwald, Gregor Hendel, Christopher Hojny, Thorsten Koch, Marco E. Lübbecke, Stephen J. Maher, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Franziska Schlösser, Christoph Schubert, Felipe Serrano, Yuji Shinano, Jan Merlin Viernickel, Matthias Walter, Fabian Wegscheider, Jonas T. Witt, and Jakob Witzig. The SCIP Optimization Suite 6.0. T. Report, Optimization Online, July 2018.

[20] Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, 1988.

[21] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[22] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2. Springer Science & Business Media, 2012.

[23] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2019.

[24] E. Györi. On division of graph to connected subgraphs. In *Combinatoris (Proc. Fifth Hungarian Colloq., Koszthely, 1976), vol. I, Colloq. Math. Soc. János Bolyai*, volume 18, pages 485–494, 1978.

[25] Christopher Hojny, Tristan Gally, Oliver Habeck, Hendrik Lüthen, Frederic Matter, Marc E Pfetsch, and Andreas Schmitt. Knapsack polytopes: a survey. *Annals of Operations Research*, pages 1–49, 2019.

[26] Christopher Hojny, Imke Joormann, Hendrik Lüthen, and Martin Schmidt. Mixed-integer programming techniques for the connected max-k-cut problem. *Mathematical Programming Computation*, 2018.

[27] Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2):424–435, 2012.

[28] Leonid Genrikhovich Khachiyan. A polynomial algorithm in linear programming. In *Doklady Akademii Nauk*, volume 244, pages 1093–1096. Russian Academy of Sciences, 1979.

[29] László Lovász. A homology theory for spanning tress of a graph. *Acta Mathematica Academiae Scientiarum Hungarica*, 30:241–251, 1977.

[30] Mario Lucertini, Yehoshua Perl, and Bruno Simeone. Image enhancement by path partitioning. In Virginio Cantoni, Reiner Creutzburg, Stefano Levialdi, and G. Wolf, editors, *Recent Issues in Pattern Analysis and Recognition. Lecture Notes in Computer Science*, volume 399, pages 12–22, 1989.

[31] Mario Lucertini, Yehoshua Perl, and Bruno Simeone. Most uniform path partitioning and its use in image processing. *Discrete Applied Mathematics*, 42(2):227–256, 1993.

[32] Jun Ma and Shaohan Ma. An $O(k^2n^2)$ algorithm to find a k-partition in a k-connected graph. *Journal of Computer Science and Technology*, 9(1):86–91, Jan 1994.

[33] Thomas L. Magnanti and Laurence A. Wolsey. Chapter 9 optimal trees. In *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 503 – 615. Elsevier, 1995.

[34] Maurizio Maravalle, Bruno Simeone, and Rosella Naldini. Clustering on trees. *Computational Statistics & Data Analysis*, 24(2):217–234, 1997.

[35] Dragan Matić and Milan Božić. Maximally balanced connected partition problem in graphs: application in education. *The Teaching of Mathematics*, XV(29):121–132, 2012.

[36] D. Matić and M. Grbić. Partitioning weighted metabolic networks into maximally balanced connected partitions. In *2020 19th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 1–6, 2020.

[37] Dragan Matić. A mixed integer linear programming model and variable neighborhood search for maximally balanced connected partition problem. *Applied Mathematics and Computation*, 237:85–97, 2014.

[38] Flávio K Miyazawa. Programação inteira. 2003.

[39] Flávio K. Miyazawa, Phablo F. S. Moura, Matheus J. Ota, and Yoshiko Wakabayashi. Cut and flow formulations for the balanced connected k-partition problem. In *International Symposium on Combinatorial Optimization*, pages 128–139. Springer, 2020.

[40] Flávio K. Miyazawa, Phablo F.S. Moura, Matheus J. Ota, and Yoshiko Wakabayashi. Partitioning a graph into balanced connected classes: Formulations, separation and experiments. *European Journal of Operational Research*, 2021.

[41] Shin-ichi Nakano, Md.Saidur Rahman, and Takao Nishizeki. A linear-time algorithm for four-partitioning four-connected planar graphs. *Information Processing Letters*, 62(6):315–322, 1997.

[42] George L Nemhauser and Laurence A Wolsey. Integer programming and combinatorial optimization. *Wiley, Chichester. GL Nemhauser, MWP Savelsbergh, GS Sigismondi (1992). Constraint Classification for Mixed Integer Programming Formulations. COAL Bulletin*, 20:8–12, 1988.

[43] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

[44] Saleh Soltan, Mihalis Yannakakis, and Gil Zussman. Doubly balanced connected graph partitioning. *ACM Trans. Algorithms*, 16(2), 2020.

[45] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51(3):385–463, May 2004.

[46] Hitoshi Suzuki, Naomi Takahashi, and Takao Nishizeki. A linear algorithm for bipartition of biconnected graphs. *Information Processing Letters*, 33(5):227–231, 1990.

[47] David Bruce Wilson. Generating random spanning trees more quickly than the cover time. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 296–303, 1996.

[48] Kati Wolter. Implementation of cutting plane separators for mixed integer programs. *Master's thesis, Technische Universität Berlin*, 2006.

[49] Bang Ye Wu. Fully Polynomial-Time Approximation Schemes for the Max-Min Connected Partition Problem on Interval Graphs. *Discrete Mathematics, Algorithms and Applications*, 04(01):1250005, 2012.

[50] Xing Zhou, Huaimin Wang, Bo Ding, Tianjiang Hu, and Suning Shang. Balanced connected task allocations for multi-robot systems: An exact flow-based integer program and an approximate tree-based genetic algorithm. *Expert Systems with Applications*, 116:10–20, 2019.

[51] Xing Zhou, Huaimin Wang, Bo Ding, Tianjiang Hu, and Suning Shang. Balanced connected task allocations for multi-robot systems: An exact flow-based integer program and an approximate tree-based genetic algorithm. *Expert Systems with Applications*, 116:10–20, 2019.

Appendix A

Detailed Results

	CUT-CROSS			CUT-CROSS-Z			FLOW			FLOW-Z			Matic			Zhou		
Instance	Sol	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time
gg_05_05_a	10	56	0.12	10	32	0.11	10	773	0.09	10	291	**0.08**	10	4,659	0.55	10	2,564	0.45
gg_05_05_b	10	913	**0.65**	10	964	0.68	10	162,312	8.04	10	383,285	21.78	10	540,447	68.18	10	7,341	1.11
gg_05_05_c	10	5,491	2.81	10	5,241	**2.72**	4	1,849,980	75.40	6	4,766,750	270.03	10	1,921,209	258.44	10	33,096	5.24
gg_05_06_a	10	189	0.26	10	98	0.17	10	601	0.08	10	348	**0.08**	10	75,140	8.95	10	1,843	0.41
gg_05_06_b	10	844	0.75	10	612	0.53	10	3,155	0.23	10	1,445	**0.13**	10	32,185	5.60	10	4,454	0.88
gg_05_06_c	10	5,903	3.60	10	6,359	3.91	10	851,461	36.21	10	43,833	**1.87**	10	1,838,715	262.33	10	26,562	4.61
gg_05_10_a	10	151	0.45	10	103	0.27	10	505	0.12	10	215	**0.10**	10	368,184	51.78	10	12,542	2.98
gg_05_10_b	10	468	0.90	10	404	0.68	10	1,484	0.23	10	685	**0.12**	10	144,469	24.57	10	18,258	4.61
gg_05_10_c	10	5,544	5.71	10	5,455	5.25	10	23,966	1.62	10	18,304	**1.21**	7	2,714,873	437.21	10	65,451	11.97
gg_05_20_a	10	246	2.29	10	294	2.19	10	399	0.21	10	189	**0.16**	1	844,935	221.98	2	116,374	43.13
gg_05_20_b	10	950	5.92	10	555	2.62	10	1,281	0.41	10	1,279	**0.33**	0	-	-	4	2,221,308	527.21
gg_05_20_c	10	3,174	10.70	10	8,865	17.89	10	27,025	3.02	10	9,221	**1.10**	0	-	-	2	1,966,330	443.98
gg_07_07_a	10	200	0.56	10	122	0.32	10	530	0.12	10	315	**0.10**	10	211,299	29.08	10	10,346	2.70
gg_07_07_b	10	497	0.87	10	622	0.94	10	1,232	0.20	10	1,310	**0.20**	9	1,360,869	163.15	10	15,657	3.32
gg_07_07_c	10	6,113	6.65	10	8,522	7.82	10	28,445	1.88	10	28,146	**1.83**	5	857,382	109.26	10	75,448	12.33
gg_07_10_a	10	136	0.62	10	177	0.57	10	467	0.17	10	163	**0.13**	8	871,395	150.27	10	529,336	107.03
gg_07_10_b	10	650	1.83	10	696	1.41	10	892	**0.23**	10	1,313	0.29	5	2,977,315	477.76	9	400,859	60.73
gg_07_10_c	10	13,661	18.95	10	10,886	13.26	10	20,226	1.79	10	24,288	**1.50**	3	3,124,113	636.81	7	1,113,012	150.25
gg_10_10_a	10	100	0.97	10	171	0.89	10	181	0.19	10	222	**0.17**	2	336,860	68.38	7	2,435,810	561.15
gg_10_10_b	10	301	1.56	10	354	1.44	10	959	0.33	10	642	**0.24**	1	404,182	89.97	5	544,515	132.06
gg_10_10_c	10	6,694	16.49	10	5,099	10.63	10	21,230	2.30	10	9,869	**1.22**	1	2,083,860	433.93	4	1,276,329	279.89
gg_15_15_a	10	125	9.95	10	156	2.86	10	181	0.60	10	152	**0.29**	0	-	-	0	-	-
gg_15_15_b	10	458	10.37	10	627	6.97	10	683	0.75	10	363	**0.36**	0	-	-	0	-	-
gg_15_15_c	10	6,780	71.81	10	14,470	91.24	10	8,018	**2.32**	10	19,554	3.82	0	-	-	0	-	-

Table A.1: Computational results for BCP_2 on grid graphs.

	CUT-CROSS			CUT-CROSS-Z			FLOW			FLOW-Z			Matic			Zhou		
Instance	Sol	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time
rnd_50_70_a	10	201	0.47	10	157	0.32	10	315	0.10	10	209	**0.09**	10	6,518	1.61	10	958	0.47
rnd_50_70_b	10	847	1.41	10	379	0.55	10	1,190	0.17	10	678	**0.13**	10	8,527	2.08	10	1,872	0.64
rnd_50_70_c	10	12,987	13.73	10	11,649	9.18	10	11,596	0.88	10	10,093	**0.75**	10	569,056	111.91	10	4,634	1.20
rnd_50_100_a	10	46	0.14	10	37	**0.09**	10	286	0.11	10	173	0.10	10	2,763	0.90	10	399	0.43
rnd_50_100_b	10	251	0.38	10	219	0.30	10	352	**0.12**	10	395	0.13	10	31,672	6.89	10	648	0.46
rnd_50_100_c	10	4,804	3.82	10	2,725	1.98	10	13,564	1.23	10	12,273	**0.93**	10	40,608	9.77	10	2,186	0.95
rnd_50_400_a	10	15	0.06	10	7	**0.04**	10	1	0.14	10	5	0.14	10	4,874	3.03	10	58	1.09
rnd_50_400_b	10	129	0.19	10	80	**0.12**	10	4	0.16	10	6	0.12	10	596,363	126.56	10	177	1.52
rnd_50_400_c	10	816	0.65	10	573	**0.46**	10	23,577	3.64	10	8,529	1.43	10	239,624	54.31	10	1,080	2.22
rnd_70_100_a	10	121	0.55	10	104	0.33	10	292	0.12	10	210	**0.11**	10	3,183	1.35	10	1,185	0.78
rnd_70_100_b	10	389	1.15	10	398	0.83	10	1,070	0.25	10	520	**0.16**	10	10,067	2.48	10	1,710	0.78
rnd_70_100_c	10	10,816	19.66	10	15,893	20.83	10	9,232	0.92	10	5,028	**0.66**	10	141,533	33.96	10	6,173	1.92
rnd_70_200_a	10	13	**0.08**	10	36	0.11	10	14	0.13	10	21	0.12	9	85,214	7.83	10	156	0.95
rnd_70_200_b	10	168	0.29	10	67	**0.13**	10	169	0.19	10	99	0.15	10	3,871	1.49	10	357	0.84
rnd_70_200_c	10	970	1.12	10	2,425	2.11	10	6,774	1.22	10	7,563	**1.05**	10	45,147	16.32	10	1,866	1.49
rnd_70_600_a	10	10	0.06	10	2	**0.04**	10	1	0.16	10	1	0.13	10	31,943	17.88	10	19	1.87
rnd_70_600_b	10	71	0.17	10	44	**0.11**	10	11	0.22	10	1	0.16	8	3,800	3.99	10	160	2.29
rnd_70_600_c	10	326	**0.47**	10	488	0.58	10	10,052	2.84	10	8,841	2.46	10	73,626	69.30	10	1,739	6.15
rnd_100_150_a	10	244	1.48	10	120	0.71	10	51	0.16	10	62	**0.15**	10	558,004	118.38	10	1,918	1.92
rnd_100_150_b	10	1,614	7.14	10	435	1.62	10	417	0.21	10	299	**0.16**	10	230,281	45.13	10	1,711	1.61
rnd_100_150_c	10	9,774	28.34	10	7,910	16.08	10	8,109	1.19	10	7,326	**0.93**	7	1,655,772	417.66	10	4,635	3.00
rnd_100_300_a	10	30	0.17	10	38	**0.16**	10	12	0.20	10	14	0.17	8	370,546	42.71	10	247	1.24
rnd_100_300_b	10	91	0.28	10	69	0.21	10	149	0.21	10	50	**0.17**	8	250,040	40.55	10	565	2.15
rnd_100_300_c	10	1,422	2.38	10	1,478	2.30	10	7,711	1.69	10	6,999	**1.30**	10	218,716	140.04	10	2,472	3.15
rnd_100_800_a	10	3	0.05	10	2	**0.05**	10	1	0.28	10	1	0.25	10	104,655	55.52	10	29	2.98
rnd_100_800_b	10	41	0.15	10	43	**0.12**	10	48	0.33	10	12	0.26	10	228,209	197.16	10	71	2.64
rnd_100_800_c	10	418	0.88	10	318	**0.62**	10	10,858	4.26	10	6,601	2.74	7	167,306	99.33	10	2,317	9.46

Table A.2: Computational results for BCP_2 on random graphs with up to 100 vertices.

	CUT-CROSS			CUT-CROSS-Z			FLOW			FLOW-Z			Matic			Zhou		
Instance	Sol	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time
rnd_200_300_a	10	80	2.16	10	68	1.30	10	535	0.42	10	336	**0.32**	7	61,372	56.27	10	1,951	7.07
rnd_200_300_b	10	397	8.11	10	360	3.66	10	711	0.46	10	521	**0.36**	7	9,413	11.13	10	2,857	10.53
rnd_200_300_c	10	11,104	97.69	10	14,908	87.37	10	6,836	2.07	10	4,862	**1.68**	7	460,086	284.91	10	5,631	14.06
rnd_200_600_a	10	14	0.32	10	28	**0.29**	10	618	0.62	10	501	0.55	6	87,323	33.46	10	728	9.67
rnd_200_600_b	10	107	0.80	10	72	0.51	10	695	0.72	10	121	**0.47**	6	64,853	68.40	10	995	9.75
rnd_200_600_c	10	584	2.72	10	3,407	13.55	10	7,894	3.50	10	3,422	**1.76**	8	366,248	298.45	10	1,771	8.96
rnd_200_1500_a	10	1	**0.07**	10	3	0.11	10	1	0.63	10	1	0.58	6	42,437	86.79	10	3	7.14
rnd_200_1500_b	10	8	**0.13**	10	10	0.14	10	684	1.23	10	15	0.80	5	347,862	372.11	10	241	14.85
rnd_200_1500_c	10	135	**0.89**	10	540	2.86	10	5,110	5.27	10	2,904	3.45	9	181,998	426.49	10	1,414	25.40
rnd_300_500_a	10	62	5.10	10	92	2.62	10	629	**0.66**	10	668	0.66	6	48,026	40.54	10	2,811	28.60
rnd_300_500_b	10	303	8.08	10	325	5.32	10	923	**0.83**	10	936	0.83	5	8,562	17.01	10	3,629	21.78
rnd_300_500_c	10	12,942	196.70	10	10,001	100.85	10	2,619	1.85	10	3,187	**1.66**	0	-	-	10	5,986	26.73
rnd_300_1000_a	10	8	**0.43**	10	22	0.44	10	1,134	1.36	10	816	1.15	6	53,284	57.45	10	982	20.13
rnd_300_1000_b	10	57	**0.85**	10	83	0.99	10	1,102	1.31	10	528	1.08	5	23,831	36.13	10	1,030	19.60
rnd_300_1000_c	10	836	7.85	10	3,279	21.75	10	4,936	**4.07**	10	10,621	6.11	4	136,102	110.01	10	2,376	30.97
rnd_300_2000_a	10	1	**0.11**	10	7	0.26	10	147	1.15	10	1	0.88	4	234,170	385.37	10	35	26.73
rnd_300_2000_b	10	28	0.55	10	10	**0.29**	10	1,616	2.88	10	15	1.33	5	96,513	200.00	10	54	80.40
rnd_300_2000_c	10	74	**0.97**	10	1,296	11.08	10	8,372	12.06	10	6,844	8.52	1	378,134	520.19	10	1,760	72.02

Table A.3: Computational results for BCP_2 on random graphs with over 100 vertices.

		CUT				CUT-Z				FLOW			FLOW-Z		
Instance	m	Sol	Sep Cuts	Nodes	Time	Sol	Sep Cuts	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time
rnd_500_931_a	$n^{1.1}$	10	9,554	35	12.41	10	9,653	55	4.38	10	1,397	2.01	10	912	**1.70**
rnd_500_931_b	$n^{1.1}$	10	12,062	309	26.86	10	10,758	475	15.49	10	1,100	1.67	10	1,291	**1.65**
rnd_500_931_c	$n^{1.1}$	10	44,421	10,696	422.38	10	19,515	5,814	127.51	10	5,501	6.24	10	5,426	**4.22**
rnd_500_1733_a	$n^{1.2}$	10	395	7	**1.16**	10	569	41	1.47	10	1,243	2.98	10	1,374	3.27
rnd_500_1733_b	$n^{1.2}$	10	626	42	2.58	10	558	101	**2.37**	10	2,101	3.45	10	1,172	2.51
rnd_500_1733_c	$n^{1.2}$	10	462	582	16.97	10	584	4,727	63.53	10	5,746	**8.87**	10	7,561	12.17
rnd_500_3226_a	$n^{1.3}$	10	22	2	**0.31**	10	42	45	1.51	10	99	2.17	10	899	3.98
rnd_500_3226_b	$n^{1.3}$	10	11	22	**0.79**	10	11	30	1.17	10	495	3.72	10	251	4.21
rnd_500_3226_c	$n^{1.3}$	10	11	317	**9.05**	10	18	814	17.09	10	5,994	17.94	10	7,689	19.78
rnd_500_6005_a	$n^{1.4}$	10	0	1	**0.30**	10	0	3	0.67	10	42	4.74	10	1	3.59
rnd_500_6005_b	$n^{1.4}$	10	0	9	**0.77**	10	0	10	1.02	10	1,257	13.55	10	211	9.96
rnd_500_6005_c	$n^{1.4}$	10	0	219	9.24	10	0	291	**7.78**	10	11,495	65.76	10	4,634	42.08
rnd_500_11180_a	$n^{1.5}$	10	0	1	**0.53**	10	0	1	1.31	10	40	3.08	10	4	2.28
rnd_500_11180_b	$n^{1.5}$	10	0	4	**0.86**	10	0	4	1.45	10	1,666	23.59	10	19	9.87
rnd_500_11180_c	$n^{1.5}$	10	0	230	**9.37**	10	0	333	9.66	10	12,482	88.52	10	4,839	54.26
rnd_1000_1995_a	$n^{1.1}$	10	23,613	31	54.04	10	23,065	52	19.16	10	2,163	**6.48**	10	1,710	6.59
rnd_1000_1995_b	$n^{1.1}$	10	22,278	64	77.14	10	20,504	90	21.13	10	2,726	6.27	10	2,134	**5.68**
rnd_1000_1995_c	$n^{1.1}$	10	32,958	5,075	641.39	10	30,917	4,473	436.17	10	8,555	19.07	10	2,810	**7.08**
rnd_1000_3981_a	$n^{1.2}$	10	639	12	**4.56**	10	686	52	6.24	10	1,942	14.19	10	1,815	12.90
rnd_1000_3981_b	$n^{1.2}$	10	821	12	5.29	10	756	30	**4.98**	10	3,545	20.95	10	2,002	11.08
rnd_1000_3981_c	$n^{1.2}$	10	715	1,342	112.18	10	718	3,403	243.15	10	8,776	44.04	10	5,574	**30.42**
rnd_1000_7943_a	$n^{1.3}$	10	6	1	**0.91**	10	5	55	6.70	10	800	24.32	10	1,277	21.60
rnd_1000_7943_b	$n^{1.3}$	10	7	1	**0.88**	10	6	34	5.50	10	2,891	48.37	10	1,164	32.64
rnd_1000_7943_c	$n^{1.3}$	10	5	147	**11.20**	10	5	575	47.14	10	7,652	85.78	10	1,753	35.04
rnd_1000_15849_a	$n^{1.4}$	10	0	1	**1.13**	10	0	1	3.50	10	148	27.59	10	4	20.75
rnd_1000_15849_b	$n^{1.4}$	10	0	1	**1.08**	10	0	1	3.48	10	10,155	134.55	10	1,031	46.12
rnd_1000_15849_c	$n^{1.4}$	10	0	48	**12.21**	10	0	377	32.35	10	5,919	127.30	10	1,606	76.87
rnd_1000_31623_a	$n^{1.5}$	10	0	1	**1.94**	10	0	1	10.11	10	523	58.47	10	1	36.59
rnd_1000_31623_b	$n^{1.5}$	10	0	1	**1.98**	10	0	1	9.43	10	2,343	144.46	10	1,185	182.83
rnd_1000_31623_c	$n^{1.5}$	10	0	180	**37.44**	10	0	285	47.11	10	2,957	154.93	10	1,112	179.64

Table A.4: Performance of CUT and FLOW to solve BCP_2 on large random graphs.

		CUT (MIN-MAX)			CUT-Z (MIN-MAX)			FLOW (MIN-MAX)			FLOW-Z (MIN-MAX)			Zhou		
Instance	k	Sol	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time
gg_07_10_a	3	0	-	-	6	71,511	758.51	10	4,177	0.93	10	1,654	**0.58**	2	1,072,107	603.97
gg_07_10_a	4	0	-	-	0	-	-	10	25,242	3.86	10	13,488	**2.47**	1	99,880	68.95
gg_07_10_a	5	0	-	-	0	-	-	10	1,578,669	247.61	10	208,118	**38.01**	0	-	-
gg_07_10_a	6	0	-	-	0	-	-	3	3,077,826	482.45	7	1,976,217	417.59	0	-	-
rnd_100_150_a	3	10	9,922	223.21	10	5,012	44.81	10	1,933	0.74	10	586	**0.41**	9	109,799	70.04
rnd_100_150_a	4	0	-	-	6	42,021	845.76	10	13,404	3.14	10	7,572	**2.19**	4	1,019,886	641.36
rnd_100_150_a	5	0	-	-	2	82,442	1384.88	10	627,636	149.27	10	188,771	**74.49**	0	-	-
rnd_100_150_a	6	0	-	-	0	-	-	6	2,268,233	682.23	6	1,516,926	838.78	1	205,247	508.93

Table A.5: Computational results for MIN-MAX BCP_k when $k \in \{2, 3, 4, 5, 6\}$.