Solving Graph Partitioning on Sparse Graphs: Cuts, Projections, and Extended Formulations

Demetrios V. Papazaharias , Jose L. Walteros*
Department of Industrial and Systems Engineering
University at Buffalo, Bell Hall, Buffalo, New York, 14260
{dvpapaza, josewalt}@buffalo.edu

Abstract

This paper explores integer programming formulations for solving graph partitioning problems that impose an upper limit on the weight of the partition clusters. Traditional efforts have concentrated on studying a model commonly known as the triangular formulation, which has some undesirable properties, like requiring a cubic number of constraints. We study some alternative formulations arising from different perspectives. In particular, we consider the idea of modeling the problem from the standpoint of an attacker who wants to deteriorate the graph's integrity by removing edges and show that some of the structural properties of the proposed formulations can be exploited to speed up the solution times. To compare the strength of the formulations' LP bounds, we study their projection into the space of the edges and show that all of them concentrate on partitioning the subtrees of the input graph. Inspired by this observation, we develop a formulation based on dynamic programming for the problem on trees and show how to use it to derive strong valid inequalities for the problem on general graphs. As part of the technical developments of the paper, we also expand the polyhedral characterization of the problem's solution space, introducing new families of inequalities and provide empirical evidence of their efficacy to improve the quality of the proposed formulations. Finally, we conduct an extensive computational study to compare the strength of our developments.

Keywords: graph partitioning, integer programming, critical element detection, branch-and-cut, dynamic programming.

Funding: This material is based upon work supported by the Office of Naval Research under contract No. N00014-20-1-2242

1 Introduction and Motivation

The graph partitioning problem over a graph $G = (V, E)$ consists of identifying a partition $\pi = \{N_1, \ldots, N_t\}$ of the vertex set V, so that each of the subsets in π—hereafter called clusters—satisfies a predefined set of properties, while the value of a cost function evaluated on the edges of the multicut is minimized. Several applications of this problem spanning a wide variety of disciplines have been extensively studied over the last few decades. Some of the most notable ones include circuit and VLSI network design [41], load balancing for parallel processing [13], districting [22, 74], community detection in social and biological networks [15, 29, 54], and failure analysis on cyber-physical systems [1, 45]. Furthermore, given that the edges of the graph often represent some

*Corresponding author. Tel: (716)-645-8876, Address: 413 Bell Hall, Buffalo, NY 14260, E-mail: josewalt@buffalo.edu

form of similarity metric between the vertices, graph partitioning has also been effectively used for clustering [34] and image segmentation [64].

Depending on the properties imposed on the clusters of a partition π and the cost function used, there are multiple versions of graph partitioning found in the literature. Among those, the ones that impose restrictions on the size or relative weight of the clusters have received considerable attention over the years [26, 27, 43, 70]. Other notable versions include lower and upper bounds on the number of clusters t [5, 50], as well as further structural requirements on the clusters, such as homogeneity and cohesiveness properties [5, 6, 35, 75]. This paper focuses on the graph partitioning problem that imposes an upper limit on the weight of the clusters. While many approaches for solving this problem have been developed over the years, we will concentrate on exact methods based on mathematical programming. Interested readers can refer to [13, 60] for a general discussion about other solution methods, both exact and heuristic.

Despite the considerable interest in this problem, there seems to be no overall agreement on a specific name for it. Multiple authors have used names such as simple graph partitioning [66, 68, 70], capacitated graph partitioning [27], size-constrained graph partitioning [43], and graph partitioning under knapsack constraints [55], among others. One of the reasons for the lack of consensus seems to be that some of these papers study models that fit multiple variations of the problem [43, 55], thus making the task of finding a generic naming convention difficult. Rather than selecting one of these names or proposing a new one, in what follows, we simply use the generic term graph partitioning and provide further clarifications whenever we discuss other variations.

1.1 Related Work

Formally, given a positive integer weight w_i associated with each vertex $i \in V$, a positive integer cost c_{ij} associated with each edge $\{i,j\} \in E$, and a constant r, the graph partitioning problem considered in this paper can be expressed as the following integer program, which we will refer to as the *triangular formulation* (TRI):

$$\text{(TRI):} \quad \min \quad \sum_{\{i,j\}\in E} c_{ij} y_{ij} \tag{1a}$$

$$\text{s.t.} \quad x_{ij} = 1 - y_{ij}, \qquad \{i,j\} \in E; \tag{1b}$$

$$\left.\begin{array}{r} x_{ij} + x_{ik} - x_{jk} \le 1 \\ x_{ij} - x_{ik} + x_{jk} \le 1 \\ -x_{ij} + x_{ik} + x_{jk} \le 1 \end{array}\right\}, \qquad \{i,j,k\} \subseteq \binom{V}{3}; \tag{1c}$$

$$\sum_{j\in V\setminus\{i\}} w_j x_{ij} \le r - w_i, \qquad i \in V; \tag{1d}$$

$$x_{ij} \in \{0,1\}, \quad \{i,j\} \in \binom{V}{2}; \tag{1e}$$

$$y_{ij} \in \{0,1\}, \quad \{i,j\} \in E. \tag{1f}$$

Here, binary variables $\mathbf{x}$, defined for every pair of vertices i and j in V, indicate whether i and j belong to the same cluster of the partition, and variables $\mathbf{y}$, defined for every edge $\{i,j\} \in E$, indicate whether edge $\{i,j\}$ is contained in the partition's multicut. Furthermore, the objective function (1a) minimizes the total cost of the edges in the multicut; constraints (1b) ensure that if two adjacent vertices are not in the same cluster, the corresponding edge connecting them is in the multicut; constraints (1c)—which are called *triangular constraints*—are defined for every triplet $\{i,j,k\} \in V$ and characterize the transitive relationship that exists among vertices of the same

cluster; and, constraints (1d) ensure that the weight of the cluster that contains each vertex is at the most r. Although the $\mathbf{y}$ variables are not required to be part of this formulation and may be projected out using the relationship given in (1b), we intentionally keep them in as they will help us to compare this formulation with others.

One drawback of this formulation is that independent of the graph's density, the total number of triangular constraints is $O(n^3)$, making it impractical to solve even for small graphs. Some recent efforts have been devoted to reducing the size of this formulation. For example, Nguyen et al. [55] showed that it is possible to remove many of the triangular constraints by only considering vertex triplets for which at least two of the vertices are adjacent, thus reducing the number of constraints to $O(nm)$ and making it more viable for sparse graphs. In Section 4, we will discuss some other properties of this formulation that lead to a similar reduction in the number of constraints.

Despite the limitations posed by its size, the triangular formulation has been at the cornerstone of graph partitioning since its introduction by Fraigle et al. [26]. Indeed, multiple publications have been devoted to studying several properties of this formulation and many of its variations. In particular, special attention has been given to the polyhedral analysis of the polytope given by the convex hull of its solutions [16, 34, 35, 36, 43, 56, 66, 67, 68, 70]. These formulations have received considerable attention because the transitive relationship enforced by the triangular constraints can be used to model several properties of the connected components of a graph [8, 30, 31, 57]. In fact, the use of these constraints has recently permeated other areas outside of graph partitioning.

Indeed, the triangular constraints have been used to model connectivity properties when solving *critical element detection* problems. In critical element detection, one solves the problem of an attacker that wants to identify a collection of "critical" elements of a graph (i.e., vertices or edges), under some budgetary constraint, whose removal maximizes the deterioration of a connectivity metric of the graph. Traditional metrics used to measure connectivity often involve accounting for the total number of connected vertex pairs and the size or weight of the largest connected component of the residual graph after the critical elements are removed. Interestingly, since most of these metrics can be easily defined in terms of the $\mathbf{x}$ variables and the triangular constraints from (1a)-(1f), it is common to find similar formulations applied to model such types of problems [8, 57, 77].

In fact, the graph partitioning problem we solve in this paper can be seen from the lens of critical element detection (cf. [57]). By taking the stance of an attacker, one can solve the problem of identifying a set of edges of minimum cost so that, when removed from the graph, each of the resulting connected components has a weight that is less than the given threshold r. In contrast to graph partitioning, from this perspective, one allows the attacker to remove edges that are not necessarily in the multicut of the resulting partition π and account for them in the objective function (i.e., the attacker can remove edges that do not help to disconnect the graph). However, under the assumption that the costs associated with all the edges are positive, any optimal solution for this problem matches an optimal solution of graph partitioning.

In addition to the triangular formulation, a second popular model introduced by Johnson et al. [40] and often named the *vertex-to-cluster formulation* (VTC) is given below:

$$\text{(VTC):} \quad \min \quad \sum_{\{i,j\}\in E} c_{ij} y_{ij} \tag{2a}$$

$$\text{s.t} \quad \sum_{q=1}^{t} v_i^q = 1, \quad i \in V; \tag{2b}$$

$$\sum_{i\in V} w_i v_i^q \leq r, \quad q \in \{1,\ldots,t\}; \tag{2c}$$

$$y_{ij} \geq v_i^q - v_j^q, \quad q \in \{1,\ldots,t\}, \quad \{i,j\} \in E; \tag{2d}$$

$$y_{ij} \geq v_j^q - v_i^q, \quad q \in \{1, \ldots, t\}, \quad \{i,j\} \in E; \tag{2e}$$

$$y_{ij} \in \{0,1\}, \quad \{i,j\} \in E; \tag{2f}$$

$$v_i^q \in \{0,1\}, \quad i \in V, \in \{1, \ldots, t\}. \tag{2g}$$

In this formulation, as before, variable y_{ij} represents whether edge $\{i,j\} \in E$ is in the multicut and variable v_i^q, defined for each vertex $i \in V$ and each potential cluster $q \in \{1, \ldots, t\}$, indicates whether vertex i is contained in cluster q. Constraints (2b) ensure that each vertex is assigned to a cluster; constraints (2b) bound the cluster weightsß; and constraints (2d)-(2e) ensure that if two adjacent vertices belong to separate clusters, then the edge between them must be in the multicut.

Contrary to the triangular formulation, this model is mainly used to solve variations of graph partitioning that impose an upper limit on the number of clusters t [40, 50] or further structural requirements on the clusters [75], and seldom used in other contexts because of its weak LP bound. Note, for example, that when the vertex weights are all equal to one, the solution where $v_i^q = 1/n$, for all $i \in V$ and $q \in \{1, \ldots, n\}$, and $y_{ij} = 0$, for all $\{i,j\} \in E$ is a feasible solution for the linear relaxation with an objective value of zero. Furthermore, the formulation is highly symmetric, as it admits any possible permutation of the cluster labels; thus, severely affecting the performance of any branch-and-bound based approach. Recent work has provided efforts in reducing the symmetries of the formulation [5]; however, the effectiveness of these approaches seems to weaken as the number of clusters t increases. Other approaches that seem to work well in practice involve solving a set partitioning reformulation of VTC via branch-and-price [50]. Despite the burden of implementing a column generation subroutine, the reformulation helps to eliminate most of the undesired symmetries, thus improving the overall strength of the formulation. An alternative approach to solve VTC was proposed by Ferreira et al. [27, 28]. Instead of dealing with the formulation directly, they investigated the convex hull of its projection into the space of the $\mathbf{y}$ variables, deriving several classes of valid inequalities and analyzing their strength. Parts of the developments of this paper analyze such a projection as well (see Sections 4 and 5).

1.2 Our Contributions

In this paper, we expand the polyhedral characterization of the multicuts of a graph. We also study several formulations for graph partitioning arising from different perspectives. We find that some of these formulations are equal in strength to the triangular formulation when the edge costs are assumed to be positive and show that some structural properties of the formulations can be exploited to speed up the solution times. To compare the strength of their linear programming relaxation, we study their projection into the space of the $\mathbf{y}$ variables and observe that all the formulations concentrate on partitioning the subtrees of the input graph. We develop a formulation based on dynamic programming for the problem on trees and show how to use it to derive strong valid inequalities for the problem on general graphs. These inequalities are then added to the proposed formulations under a traditional branch-and-cut framework, using both exact and heuristic separation routines. We provide the specific details of our implementation and test our models on a wide variety of networks collected from previous works on graph partitioning and randomly generated graphs to compare their performance and identify potential improvements. Our implementation and the data sets used in our computational experiments are publicly available at `https://github.com/Dpapazaharias1/graph-partitioning`.

The rest of this paper is structured as follows. In Section 2 we introduce the notation we will use throughout the paper and formally characterize the graph partitioning problem with respect to the multicuts. In Section 3, we study the polyhedral description of the multicuts and describe a set covering formulation aimed at partitioning the trees of connected subgraphs with vertex weight

greater than r. We also analyze families of valid inequalities that generalize others found in the literature. In Section 4, we introduce two extended formulations for graph partitioning and prove that they are equal in strength to the triangle formulation. In Section 5, we study graph partitioning for trees. We introduce a pseudo-polynomial dynamic programming algorithm and extend it into a linear programming model. We also show how to use the proposed linear program to derive strong inequalities for the problem on general graphs. Finally, we discuss implementation details and test the performance of our solution techniques on a series of randomly generated instances and real-life networks in Section 6, and present our conclusions in Section 7.

2 Notation and Background

We begin by providing the notation and terminology used throughout the paper. We the give a general description of the graph partitioning problem we intend to solve. Finally, we discuss some initial developments needed to introduce the formulations discussed later in Sections 3 and 4.

We work with a simple, nonempty graph $G = (V, E)$, where V is a set of n vertices and $E \subseteq \binom{V}{2}$ a set of m edges. We will refer to each edge by its index $e = 1, \ldots, m$ or by its unordered pair of endpoints $\{i, j\}$. We say that two vertices i and j are adjacent in G if $\{i, j\} \in E$. The neighborhood of a vertex $i \in V$ in G, which we denote $\mathcal{N}(i)$, is the set vertices adjacent to i in G and its cardinality, denoted $\deg(i) = |\mathcal{N}(i)|$, is the vertex's degree. Two vertices i and j are connected in G if there exists a path between them in G. We say that H is a subgraph of G if the vertices and edges of H, denoted $V(H)$ and $E(H)$, are subsets of V and E, respectively. For any subset of vertices $N \subseteq V$, $G[N]$ represents the subgraph induced by N. That is, $G[N] = G(N, E')$, where $E' = E \cap \binom{N}{2}$. Furthermore, we say that N induces a connected component of G if every pair of vertices in N are connected in G and every vertex in $V \setminus N$ is disconnected from all vertices in N. A subgraph T of G is called a *tree* if every pair of vertices in $V(T)$ is connected by a unique path in T. A vertex of a tree with degree one is called a *leaf* whereas a vertex with a larger degree is called an *internal vertex*. Removing an edge $\{i, j\}$ from a tree T splices it into two subtrees; we then denote T^i_{ij} the subtree containing i and T^j_{ij} the subtree containing j resulting from removing edge $\{i, j\}$ from T.

Given the vertex weights $\mathbf{w} = [w_i]_{i \in V}$ and edge costs $\mathbf{c} = [c_e]_{e \in E}$, we define weight and cost functions $w(\cdot) : 2^V \to \mathbb{Z}$ and $c(\cdot) : 2^E \to \mathbb{Z}$, respectively, so that for any vertex set $N \subseteq V$ and edge set $S \subseteq E$, $w(N) = \sum_{i \in N} w_i$ and $c(S) = \sum_{e \in S} c_e$. We use $r \in \mathbb{Z}$ to denote the upper bound imposed on the weights of each cluster for the graph partitioning problem. Given a tree T, we will say that T is a *tree cover* if $w(V(T)) > r$. Furthermore, T is a *minimal* tree cover if $w(V(T) \setminus \{i\}) \leq r$ for each leaf $i \in V(T)$. We denote $\mathcal{T}(G, r)$ the set of all tree covers in G given the upper bound r and use $\tilde{\mathcal{T}}(G, r)$ when referring to the tree covers that are minimal.

A partition $\pi = \{N_1, \ldots, N_t\}$ of vertex set V is a collection of $t \geq 1$ nonempty vertex subsets (clusters) so that $\bigcup_{q=1}^{t} N_q = V$, and $N_p \cap N_q = \emptyset$, for $p \neq q$. We will say that $t = |\pi|$ is the size of the partition and assume that t is not fixed and may vary between partitions. Given a partition π, its *multicut* is denoted $\delta(N_1, \ldots, N_t)$—or simply $\delta(\pi)$—and comprises the edges that are incident to vertices from different clusters. That is, $\delta(\pi) := \{\{i, j\} \in E \mid i \in N_p, j \in N_q, p \neq q\}$. We say that a partition is *clean* if each cluster $N_q \in \pi$ induces a connected component in $G(V, E \setminus \delta(\pi))$. We define Π to be the collection of all possible clean partitions of V and $\Pi(r)$ the set of all clean partitions for which the weights of its clusters are less that or equal to r. That is, $\Pi(r) := \{\pi \in \Pi \mid w(N_q) \leq r, N_q \in \pi\}$. Furthermore, we define $\Delta := \{\delta(\pi) \mid \pi \in \Pi\}$ to be the collections of multicuts associated with the clean partitions of V and denote $\Delta(r) := \{\delta(\pi) \mid \pi \in \Pi(r)\}$, accordingly. We will say that a multicut $S \in \Delta(r)$ is minimal, if $S \setminus \{e\} \notin \Delta(r)$, for all $e \in S$.

It is easy to see that multiple partitions may share the same multicut, but each multicut is

associated with a unique clean partition. Therefore, there is a bijective mapping between Π and Δ, and consequently between $\Pi(r)$ and $\Delta(r)$. In what follows, we deal exclusively with clean partitions and will refer to those simply as partitions, unless further clarifications are required.

The graph partitioning problem we consider in this paper can be described as follows:

$$\min\{c(S) \mid S \in \Delta(r)\}. \tag{3}$$

That is, one would like to identify a multicut of minimum cost, whose corresponding partition π is composed of clusters with a weight no larger than r. An obvious feasibility condition for this problem requires that $w_i \leq r$, for all $i \in V$. Furthermore, to avoid some trivialities, we will assume without loss of generality that $w_i + w_j \leq r$, for all $\{i, j\} \in E$, otherwise edge $\{i, j\}$ must belong to all multicuts in $\Delta(r)$. In such a case, we might as well remove such edge from E, solve the problem on the residual graph, and then add $\{i, j\}$ to the given optimal multicut. We also note that under the aforementioned conditions the graph partitioning problem is always feasible because $E \in \Delta(r)$.

One of the key elements of most mathematical formulations for graph partitioning is the way in which set $\Delta(r)$ is represented. As discussed in Section 1, most recent efforts have been focused on studying the TRI and VTC formulations. These two models are what the literature often refers to as *extended formulations*, in the sense that they are defined over an "extended" dimensional space as a result of introducing additional variables that help to represent the problem's set of feasible solutions (see [20] for further reference on extended formulations). While the multicuts in $\Delta(r)$ can be represented directly over the $\mathbf{y}$-space, a formulation of this type, as we will show in Section 3, may require an exponential number of constraints. Formulations TRI and VTC, on the other hand, make use of connectivity variables $\mathbf{x}$ and vertex to cluster variables $\mathbf{v}$, respectively, to model the problem in a higher-dimensional space, but with only a polynomial number of constraints.

To the best of our knowledge, the seminal work by Ferreira et al. [27, 28] is the only published work aimed to model the multicuts in the space of the edges. First, in [27], the authors study the convex hull of the incidence vectors of the multicuts in $\Delta(r)$, providing multiple families of valid inequalities and analyzing their strength under different conditions. Then, in the computational part of the study [28], they use the VTC formulation as their base model, and solve it via branch-and-cut after strengthening it via the generation of the inequalities developed in [27]. There are multiple papers that also study polyhedrons associated with multicuts in other contexts [17, 21, 35, 44]. However, most of them either focus on extended formulations similar to TRI, or assume that the input graph G is complete. In this paper we are particularly interested in studying the graph partitioning problem over the edge space as well.

3 Polyhedral Characterization of the Multicuts

3.1 Definitions and Basic Properties

Given a set of edges $S \subseteq E$, let $\mathbf{y}^S$ be the indicator vector of the edges in S. That is, $y_e = 1$ if $e \in S$ and 0, otherwise. Furthermore, let $Y := \{\mathbf{y}^S \in \{0,1\}^m \mid S \in \Delta(r)\}$ denote the set of incidence vectors of the multicuts in $\Delta(r)$, and $\mathscr{P}(\Delta(r)) := \text{conv}(Y)$ be the convex hull of its elements. On some occasions, instead of studying directly the facial structure of a polyhedron $\mathscr{P}$, it is often easier to study the facets of a relaxation instead. Of particular interest, are relaxations that share the optimal face with $\mathscr{P}$ for a given objective function. Motivated by this idea, instead of studying $\mathscr{P}(\Delta(r))$, we analyze the convex hull of the incidence vectors of the upper closure of $\Delta(r)$.

Consider the partially ordered set (poset) composed of the elements of power set 2^E, while being ordered by inclusion. We define the *upper closure* $\uparrow\Delta(r)$ to be the collection of supersets of the

multicuts in $\Delta(r)$. That is, $\uparrow\Delta(r) := \{S \supseteq S' \mid S' \in \Delta(r)\}$. Since E is a superset of any element in $\uparrow\Delta(r)$ (i.e., E is the *join* of the upper closure), $\uparrow\Delta(r)$ is also a join-semilattice and thus is closed under union. The following proposition provides an important property of the upper closure $\uparrow\Delta(r)$.

Proposition 1. *Given a positive integer cost vector* $\mathbf{c} = [c_e]_{e\in E}$ *so that* $c(S) = \sum_{e\in S} c_e$,

$$\min\{c(S) \mid S \in \Delta(r)\} = \min\{c(S) \mid S \in \uparrow\Delta(r)\}. \tag{4}$$

Proof. By definition, the minimal sets of $\Delta(r)$ and $\uparrow\Delta(r)$ are the same and therefore are also multicuts, as elements of $\Delta(r)$. Furthermore, since $c_e > 0$ for all $e \in E$, then cost function $c(\cdot)$ is increasing implying that all optimal solutions for both problems in (4) are minimal multicuts. □

We now provide a characterization of the edge sets in $\uparrow\Delta(r)$. To this end, consider the following problem seen from the perspective of an attacker whose aim is to remove edges to cause a fragmentation on graph G. In this context, the attacker would like to identify a set of edges $S \subseteq E$ such that the sum of the vertex weights for each of the connected components in $G(V, E \setminus S)$ is not greater than r. For any feasible attacking strategy S, the vertex sets of the connected components of $G(V, E \setminus S)$ compose a partition $\pi \in \Pi(r)$; therefore, $S \in\uparrow\Delta(r)$. Furthermore, since the elements of $\uparrow\Delta(r)$ are multicuts from $\Delta(r)$ or their supersets, an edge set S is a valid solution for the attacker if and only if $S \in\uparrow \Delta(r)$. From this observation, by analyzing the feasible solutions of the attacker, it is possible to obtain the following characterization of the feasible elements of $\uparrow\Delta(r)$.

Theorem 1. *Given a graph* $G = (V, E)$*, a positive integer cost vector* $\mathbf{c} = [c_e]_{e\in E}$*, and a constant* $r \in \mathbb{Z}$*, an edge set* $S \subseteq E$ *is a member of* $\uparrow\Delta(r)$ *if and only if* $S \cap E[T] \neq \emptyset$*, for each minimal tree cover* $T \in \tilde{\mathcal{T}}(G, r)$*.*

Proof. To show necessity, suppose that $S \in\uparrow\Delta(r)$ and thus S is a feasible solution for the attacker, but there exists a tree cover $T \in \tilde{\mathcal{T}}(G, r)$ for which $S\cap T = \emptyset$. In such a case, after set S is removed from G, tree T remains in $G(V, E \setminus S)$ implying that the vertices in $V(T)$ belong to a connected component of $G(V, E \setminus S)$ with a vertex weight larger than r, thus contradicting the feasibility of S.

To show sufficiency, suppose that for the given edge set S, we have that $S \cap E[T] \neq \emptyset$, for each tree cover $T \in \mathcal{T}(G, r)$, but there is still a connected component H in $G(V, E \setminus S)$ for which $w(V(H)) > r$. In such a case, since H is a connected component in $G(V, E \setminus S)$, any spanning tree T of H must also exist in $G(V, E \setminus S)$. Since $w(V(T)) = w(V(H)) > r$, T must contain a minimal tree cover T' for which $S \cap E(T') = \emptyset$, a contradiction. □

This characterization yields the following set-covering type of model for solving $\min\{c(S) \mid S \in \uparrow\Delta(r)\}$, which we label the *tree covering formulation* (TCF).

$$\text{(TCF):} \quad \min \quad \sum_{e\in E} c_e y_e \tag{5a}$$

$$\text{s.t} \quad \sum_{e\in E(T)} y_e \geq 1, \quad T \in \tilde{\mathcal{T}}(G, r); \tag{5b}$$

$$y_e \in \{0, 1\}, \quad e \in E. \tag{5c}$$

Here, constraint set (5b) ensures that each minimal tree cover has one of its edges removed; otherwise, there exists a connected component in the residual graph with a vertex weight larger than r.

Set covering formulations, like (5a)–(5c), are quite common in the literature for modeling *attacker-defender games*, where an attacker desires to block some structures from the defender's graph [19, 38, 48, 49, 79]. Based on the relationship stated in Proposition 1, we can also use this

formulation to find optimal solutions for the graph partitioning problem, provided that $\mathbf{c} > \mathbf{0}$. The main idea behind this formulation is that the feasible solutions for the attacker are not necessarily multicuts, as the attacker may remove edges connecting vertices of the same cluster; however, under the assumption that $\mathbf{c} \geq \mathbf{0}$, any optimal solution for the attacker will be a multicut.

Ferreira et al. [27] were the first to identify the tree cover inequalities (5b) to be valid for $\mathscr{P}(\Delta(r))$ and then successfully used them to strengthen the VTC formulation. However, they did not use them directly as a valid formulation for the problem as they also consider other restrictions on the number of clusters. In this paper we will also provide necessary and sufficient conditions for these inequalities to be facet-defining for the polyhedron given by the incidence vector of the sets in $\uparrow\Delta(r)$ and use them directly as part of standalone formulation TCF. Before we proceed with our analysis of set $\uparrow\Delta(r)$, we first study some computational aspects of solving TCF.

Considering the density of G, the vertex weights $\mathbf{w}$, and the upper bound r, the number of minimal tree covers in $\tilde{\mathcal{T}}(G, r)$ may grow exponentially large and therefore solving TCF requires a separation algorithm for inequalities (5b). As proven in [27], separating fractional solutions is NP-hard, however, the separation of integer values of $\mathbf{y}$ can be done in polynomial time.

Theorem 2. *Given a candidate solution $\hat{\mathbf{y}}$, the separation problem for the tree cover inequalities (5b) is:*

(1) NP-Hard if $\hat{\mathbf{y}} \in [0,1]^m$ (proven by Ferreira et al. [27] by a reduction from minimum Steiner tree [46]).

(2) solvable in $O(n+m)$ if $\hat{\mathbf{y}} \in \{0,1\}^m$ and $w_i = 1$, for all $i \in V$.

(3) solvable in $O(n \log n + m)$ if $\hat{\mathbf{y}} \in \{0,1\}^m$ and $\mathbf{w} \in \mathbb{Z}^n$.

Proof.

(2) Consider the residual graph $G(V, E \setminus S)$ where $S = \{e \in E \mid \hat{y}_e = 1\}$. First, we identify the set of connected components of $G(V, E \setminus S)$, which can be done in $O(m+n)$ via depth-first-search [37]. If all the connected components have at the most r vertices, then no tree cover inequality (5b) is violated. Otherwise, let H be a connected component such that $|V(H)| > r$, then we can grow a tree T in $O(m+n)$ using depth-first-search on H until $V(T) = r+1$. Since $\hat{y}_e = 0$ for all $e \in E(T)$, then T is a minimal tree cover whose inequality is violated by $\hat{\mathbf{y}}$.

(3) As before, we first identify the set of connected components of $G(V, E \setminus S)$ and if all the connected components of this graph have a vertex weight less than or equal to r, then no tree cover inequality (5b) is violated. Otherwise, let H be a connected component such that $w(V(H)) > r$, then we can grow a tree T in $O(m+n)$ using depth-first-search on H until $w(T) > r$. Although T is a tree cover, it may not be minimal. Thus, we may require to extract a minimal tree cover from T. To do so, we sequentially remove from T the leaf i of minimum weight unless $w(V(T \setminus \{i\})) \leq r$. This procedure can be accomplished in $O(n \log n)$ time by keeping the leaves of T in a binary heap sorted by weight. Furthermore, since at each iteration the leaf that is removed is the one of minimum weight, the tree produced by this algorithm is a minimal tree cover and its corresponding inequality is violated by $\hat{\mathbf{y}}$. □

We now provide some additional properties of $\uparrow\Delta(r)$. Let $\overline{Y} := \{\mathbf{y}^S \in \{0,1\}^n \mid S \in \uparrow\Delta(r)\}$ denote the set of incidence vectors of the sets in $\uparrow\Delta(r)$, and $\mathscr{P}(\uparrow\Delta(r)) := \text{conv}(\overline{Y})$ be the convex hull of its elements. In Theorem 3, we provide some basic properties of $\mathscr{P}(\uparrow\Delta(r))$. Since this polytope is the convex hull of feasible solutions of TCF, some of the following properties of $\mathscr{P}(\uparrow\Delta(r))$ are inherited directly from set covering [9].

Theorem 3. *Given a graph $G = (V, E)$, a positive integer cost vector $\mathbf{c} = [c_e]_{e \in E}$, a positive integer weight vector $\mathbf{w} = [w_i]_{i \in V}$, and a constant $r \in \mathbb{Z}$ so that $w_i \leq r$, for all $i \in V$ and $w_i + w_j \leq r$, for all $\{i, j\} \in E$, the following statements about $\mathscr{P}(\uparrow\Delta(r))$ are true:*

(1) $\mathscr{P}(\uparrow\Delta(r))$ is a full-dimensional polytope.

(2) Given an edge $e \in E$, inequality $y_e \leq 1$ induces a facet of $\mathscr{P}(\uparrow\Delta(r))$

(3) Given an edge $\{i, j\} \in E$, inequality $y_{ij} \geq 0$ induces a facet of $\mathscr{P}(\uparrow\Delta(r))$ if and only if, for each vertex $k \in (\mathcal{N}(i) \cup \mathcal{N}(j)) \setminus \{i, j\}$, $w_i + w_j + w_k \leq r$.

(4) Given minimal tree cover $T \in \tilde{\mathcal{T}}(G, r)$, the corresponding minimal tree cover inequality $\sum_{e \in E(T)} y_e \geq 1$ induces a facet of $\mathscr{P}(\uparrow\Delta(r))$ if and only if:

(a) T is not contained by a cycle in G, and

(b) there is no edge $\{i, k\} \in E$, with $i \in V(T)$ and $k \notin V(T)$, such that $w(V(T^i_{ij})) + w_k > r$, for all $j \in \mathcal{N}(i) \cap V(T)$.

Proof.

(1) A proof for the case where all vertex weights are equal to one can be found in [27]. We consider here the general case. First, since $w_i + w_j \leq r$, for all $\{i, j\} \in E$, then $\mathbf{y}^{E \setminus \{e\}} \in \mathscr{P}(\uparrow \Delta(r))$, for all $e \in E$. Also, since E is the join of $\uparrow\Delta(r)$, then $\mathbf{y}^E \in \mathscr{P}(\uparrow\Delta(r))$. These are $m + 1$ affinely independent points in $\mathscr{P}(\uparrow \Delta(r))$.

(2) For any given edge $e \in E$, the face of $\mathscr{P}(\uparrow\Delta(r))$ induced by inequality $y_e \leq 1$ contains the incidence vectors of the following m edge sets in $\uparrow\Delta(r)$: the complete edge set E, and the edge sets $E \setminus \{e'\}$ for all $e' \in E \setminus \{e\}$. Since these incidence vectors are affinely independent, the dimension of this face is $m - 1$ and therefore, it is a facet of $\mathscr{P}(\uparrow\Delta(r))$.

(3) To show sufficiency, we prove that for any edge $e = \{i, j\} \in E$, the face induced by $y_e \geq 0$ is of dimension $m - 1$ if $w_i + w_j + w_k \leq r$, for all $k \in (\mathcal{N}(i) \cup \mathcal{N}(j)) \setminus \{i, j\}$. Consider the edge sets $E \setminus \{e\}$ and $E \setminus \{e, e'\}$ for all $e' \in E \setminus \{e\}$. We observe that $E \setminus \{e\} \in \uparrow\Delta(r)$, as $w_i + w_j \leq r$. Furthermore, consider the set of connected components of graph $G(V, \{e, e'\})$. First, if $e \cap e' = \{k\}$, then the only connected component in such a graph that is nor a singleton vertex contains vertices i, j, k and its weight is $w_i + w_j + w_k \leq r$. Alternatively, if $e \cap e' = \emptyset$ for $e' = \{i', j'\}$, such a graph contains two non-singleton connected components whose weights are $w_i + w_j \leq r$ and $w'_i + w'_j \leq r$, respectively. Therefore, $E \setminus \{e, e'\} \in \uparrow\Delta(r)$. It is easy to see that the incidence vectors of these m edge sets are affinely independent.

To show necessity, we prove the contrapositive. Suppose there exists an edge e' connecting either i or j to a vertex k such that $w_i + w_j + w_k > r$. Then, vertex set $\{i, j, k\}$ induces a minimal tree cover T whose tree cover inequality is $y_e + y_{e'} \geq 1$. Since we can obtain the valid inequality $y_e \geq 0$ by adding this minimal tree cover inequality and the upper bound inequality $-y_{e'} \geq -1$, $y_e \geq 0$ cannot induce a facet of $\mathscr{P}(\uparrow\Delta(r))$.

(4) To show sufficiency, we will prove that the dimension of the face induced by the tree cover inequality of tree T that satisfies (a) and (b) is $m - 1$. To this end, we will identify m affinely independent solutions contained in such a face. First, consider the $|E(T)|$ edge sets comprising one edge e from $E(T)$ and all the edges in $E \setminus E(T)$. Since T is a minimal tree cover,

such edge sets are in $\uparrow\Delta(r)$, and the corresponding incidence vectors belong to the face. To obtain the other $m - |E(T)|$ solutions, we consider the edge sets composed of all edges in $E \setminus E(T)$ except for one edge e' and one edge $e \in E$ that is selected depending on e' as follows: First, if e' is not incident to any vertex of $V(T)$, then we are free select any edge $e \in E(T)$. Second, if both endpoints of e' belong to $V(T)$, then adding e' to T produces a cycle that contains some vertices from $V(T)$, but not all as per condition (a). For this case, we are free to select any edge $e \in E(T)$ that does not lie on the cycle. Third, if exactly one of the endpoints of $e' = \{i, k\}$, say i, belongs to $V(T)$, then we can pick any edge $e = \{i, j\}$ for which $w(V(T_{ij}^i)) + w_k \leq r$, which we know that exists because of condition (b). We can repeat this procedure for each edge $e' \in E \setminus E(T)$. Since T is a minimal tree cover, all these edge sets are in $\uparrow\Delta(r)$ and their incidence vectors are in the face and are affinely independent.

We will show necessity by contraposition. First, suppose there exists a minimal tree cover T and edge $e' \in E \setminus E(T)$ which violates condition (a). Namely, $E(T) \cup \{e'\}$ creates a cycle with $|V(T)|$ nodes. Notice this cycle contains $|V(T)|$ minimal tree covers. We can take each of the corresponding tree cover inequalities and apply the Chvátal-Gomory rounding procedure [18, 32] to generate a new valid inequality. Since each edge of the cycle will appear in $|V(T)| - 1$ minimal tree covers, we obtain the valid inequality

$$\sum_{e \in E(T)} y_e + y_{e'} \geq \left\lceil \frac{|V(T)|}{|V(T)| - 1} \right\rceil = 2 \tag{6}$$

Adding the trivial inequality $-y_{e'} \geq -1$ to (6) produces the tree cover inequality for T, which implies that such inequality cannot induce a facet of $\mathscr{P}(\uparrow\Delta(r))$.

Second, suppose there exists a minimal tree cover T and an edge $\{i, k\} \in E \setminus E(T)$ that violates condition (b). Therefore, for each $j \in \mathcal{N}(i) \cap V(T)$, the tree formed by attaching k to tree T_{ij}^i is a tree cover (not necessarily minimal). We can take the corresponding tree cover inequalities of these trees, as well as the tree cover inequality of T and apply the Chvátal-Gomory rounding procedure to generate a new valid inequality. Since each edge in $E(T) \cup \{i, k\}$ will appear in $|\mathcal{N}(i) \cap V(T)|$ tree covers, we obtain the valid inequality

$$\sum_{e \in E(T)} y_e + y_{ik} \geq \left\lceil \frac{|\mathcal{N}(i) \cap V(T)| + 1}{|\mathcal{N}(i) \cap V(T)|} \right\rceil = 2 \tag{7}$$

Adding the upper bound inequality $-y_{ik} \geq -1$ to (7) produces the tree cover inequality for T, which implies that such inequality cannot induce a facet of $\mathscr{P}(\uparrow\Delta(r))$. □

The following sections will be focused on other families of valid inequalities for $\mathscr{P}(\uparrow\Delta(r))$ that can broadly be seen as generalizations of the tree cover constraints (5b).

3.2 Knapsack Tree Inequalities

In this section, we study some properties of the *knapsack tree inequalities* originally introduced in [27, 28]. Given a tree cover $T \in \mathcal{T}(G, r)$, Ferreira et al. [27, 28] observed that the upper bound r on the tree weight $w(V(T))$ defines an intrinsic knapsack constraint that can be exploited to generate valid inequalities for $\mathscr{P}(\uparrow\Delta(r))$.

To describe these inequalities, we first provide the following definition about *rooted out-trees*, also called *arborecences* [23]. A rooted out-tree—denoted $\vec{T}$ here—is a directed tree with a special

vertex k called the *root* so that for every other vertex i in the tree there is a unique (simple) path that connects k with i. For any two vertices i and j, if there is a directed path from i to j in $\vec{T}$, then j is called a *descendant* of i. Furthermore, if i and j are adjacent, then i is the *parent* of j and j is a *child* of i. We will denote $\vec{T}^i$ the rooted subtree of $\vec{T}$ stemming from vertex i (i.e., the subtree that contains i and its descendants). Note that any tree T can be transformed into a rooted out-tree, for a given a root vertex $k \in V(T)$, by replacing each edge $\{i,j\} \in E(T)$ with an arc (i,j) oriented accordingly. We will use $A(\vec{T})$ to denote the set of such arcs.

Given a tree cover $T \in \mathcal{T}(G,r)$ and any vertex $k \in V(T)$, let $\vec{T}$ be the corresponding out-tree rooted at k. Then, the resulting knapsack tree inequality is given by:

$$\sum_{(i,j)\in A(\vec{T})} a_{ij} y_{ij} \geq w(V(T)) - r, \tag{8}$$

where $a_{ij} = \min\{w(V(\vec{T}^j)), r, w(V(T)) - r\}$. Here, $w(V(\vec{T}^j))$ is the weight of subtree $\vec{T}^j$ that gets severed from $\vec{T}$ when placing edge $\{i,j\}$ into the multicut.

Ferreira et al. [27] use the name knapsack tree inequalities not only in reference to (8), but also for any other valid inequality of the corresponding knapsack polytope given by r and the weight of the vertices in T (e.g., any of the the well-known *cover* and *extended cover* inequalities [53] widely used to solve knapsack problems). As in [28], we will focus our analysis on (8).

Notice that depending on which vertex is chosen as the root, the value of coefficients a_{ij} might be different for the subtree T^i associated with vertex i may change. In fact, a poor choice of root can lead to a weaker inequality. To overcome this issue, [28] provided a method for tightening the a_{ij} coefficients to produce a strengthened version of this constraint given a root choice k. Interestingly, there is in fact an efficient way to identify the root that yields the strongest cut without the need of either testing multiple candidate roots or conducting the strengthening procedure proposed in [28].

In what follows, we provide an algorithm that runs in linear time and computes the coefficients of such a knapsack cut (8) for a given tree cover $T \in \mathcal{T}(G,r)$. The algorithm works by sequentially contracting the leaves of the tree with its neighbors, each time calculating the coefficient of the edge connecting them and combining their weights. The algorithm uses the following components:

- `Q`: A queue that progressively stores the leaves of the residual tree. It is initialized with the leaves of T.
- `W`: A scalar that indicates the total weight $w(V(T))$ of the vertices in T.
- `deg`: An array that indicates the degree of each vertex in the residual tree. The corresponding entry of vertex i is initialized as $|\mathcal{N}(i) \cap V(T)|$.
- `mrk`: An array that marks whether a vertex has been processed by the algorithm and whose entries are initialized as `false`.
- `wgt`: An array that stores the weight of the vertices in the residual tree. The corresponding entry for vertex i is initialized as w_i and increases after each contraction.
- `par`: An array that indicates the parent vertex i in the computed rooted out-tree $\vec{T}$. The corresponding entry for each vertex i is initialized as i. After completion, the vertex whose parent is the vertex itself is the root of the out-tree.

Proposition 2. *Given a tree cover $T \in \mathcal{T}(G,r)$, Algorithm 1 identifies in $O(|V(T)|)$ time the rooted out-tree $\vec{T}$ that produces the strongest knapsack tree inequality (8) associated with T, as well as the corresponding coefficients a_{ij} for each edge $\{i,j\} \in E$.*

Algorithm 1 Computes the coefficient of each edge for the knapsack tree inequality.

Input: A graph $G = (V, E)$, a positive integer weight vector $\mathbf{w} = [w_i]_{i \in V}$, a constant $r \in \mathbb{Z}$, and a tree cover $T \in \mathcal{T}(G, r)$
Output: The coefficients a_{ij} of the knapsack tree inequality associated with T, and the root k of the out-tree $\vec{T}$ that yields such inequality.

1: **procedure** KnapsackTree($G, r, \mathbf{w}, T$)
2: Initialize the queue `Q` and the arrays `deg`, `mrk`, `wgt`, and `par`, and compute $\mathtt{W} = w(V(T))$.
3: **while** `Q` is not empty **do**
4: $i \leftarrow$ `Q.pop`
5: $\mathtt{mrk}[i] \leftarrow \mathtt{true}$
6: $M = \{v \in \mathcal{N}(i) \cap V(T) \mid \mathtt{mrk}[v] = \mathtt{false}\}$
7: $j \leftarrow v \in \mathcal{N}(i) \cap V(T)$ such that $\mathtt{mrk}[v] = \mathtt{false}$
8: **if** $j \neq$ `null` **then** ▷ If i has no unmarked neighbor, then i is the last vertex left in the tree
9: $a_{ij} \leftarrow \min\{\mathtt{wgt}[i], \mathtt{W} - \mathtt{wgt}[i]\}$ ▷ This choice identifies subtree $\vec{T}^i$
10: **if** $\mathtt{wgt}[i] < \mathtt{W} - \mathtt{wgt}[i]$ **then**
11: $\mathtt{par}[i] \leftarrow j$
12: **else**
13: $\mathtt{par}[j] \leftarrow i$
14: **end if**
15: $a_{ij} \leftarrow \min\{a_{ij}, r, \mathtt{W} - r\}$
16: $\mathtt{wgt}[j] \leftarrow \mathtt{wgt}[j] + \mathtt{wgt}[i]$ ▷ A contraction occurs; leaf i is fused into its neighbor j.
17: $\mathtt{deg}[j] \leftarrow \mathtt{deg}[j] - 1$
18: **if** $\mathtt{deg}[j] = 1$ **then** ▷ Vertex j is pushed into the queue if it becomes a leaf.
19: `Q.insert`(j)
20: **end if**
21: **end if**
22: **end while**
23: **return** $\mathbf{a}, \mathtt{par}, \mathtt{sub}$
24: **end procedure**

Proof. For the time complexity, we assume that the tree is stored using adjacency lists, i.e., for each vertex $i \in V(T)$, there is a list composed of the vertices in $\mathcal{N}(i) \cap V(T)$. The initialization (step (2)) clearly takes $O(|V(T)|)$. For the while loop (step (3)), each vertex in $V(T)$ is marked exactly once and since all the updates to `Q`, `deg`, `mrk`, `wgt`, `par`, and $\mathbf{a}$ take constant time, the processing time adds up to $O(V(T))$ time. Also, note that finding the neighbor in step (7) requires traversing the adjacency list of each vertex once, which also adds up to $O(V(T))$ time.

To prove correctness, we begin by showing that array `par` indeed represents a rooted out-tree. First, note that there cannot be cycles in `par`, for that requires a marked vertex to be selected as j in step (7). Second, we show that there is only one root, i.e., there is only one vertex i for which $\mathtt{par}[i] = i$, which only occurs when $\mathtt{par}[\mathrm{i}]$ is never updated by the algorithm. We claim this only happens when every vertex j being fused into i satisfies $\mathtt{wgt}[j] < \mathtt{W} - \mathtt{wgt}[j]$ and then, when vertex i is being processed, $\mathtt{wgt}[i] \geq \mathtt{W} - \mathtt{wgt}[i]$. Note that, since $\mathtt{W} - \mathtt{wgt}[i]$ includes the weights of all vertices that have not been marked, if there is a vertex j being processed after i for which $\mathtt{wgt}[j] \geq \mathtt{W} - \mathtt{wgt}[j]$, then i must be one of the vertices contained by j after some contractions (step (16)). Finally, a similar argument can be used to show that $\mathtt{par}[i]$ gets updated at the most once for each vertex i. If $\mathtt{par}[i]$ gets updated more than once (step (13)), that requires vertex i to have two neighbors j and l in T that satisfy $\mathtt{wgt}[j] \geq \mathtt{W} - \mathtt{wgt}[j]$ and $\mathtt{wgt}[l] \geq \mathtt{W} - \mathtt{wgt}[l]$ when fused into i which is not possible since the vertex weights in $\mathbf{w}$ are assumed positive.

To complete the proof and show that the algorithm produces the strongest knapsack tree inequality, it suffices to note that step (9) identifies the subtree $\vec{T}^i$ that minimizes the value of coefficient a_{ij} for all $i \in V(T)$.

3.3 Generalizing the tree cover inequalities

Given a connected graph $G = (V, E)$, the *edge connectivity* of G, denoted $\lambda(G)$, is the minimum number of edges that must be removed from G to disconnect it. Furthermore, a vertex $i \in V$ is

called an *articulation point* if its removal disconnects G. A graph is said to be *biconnected* if it has no articulation points and a biconnected component of G—aka a *block* of G—is a maximal biconnected subgraph of G. Any connected graph can be decomposed into its set of blocks, which are attached to each other in G at the shared articulations points. In what follows, we use $\mathcal{B}(G)$ to denote the set of blocks of G. If a graph is biconnected, the graph itself is its unique block; if the graph is a tree, all the internal vertices are articulation points and each pair of adjacent vertices comprises one of the graph's blocks. Figure 1 depicts a connected graph that has five blocks, each of those being the subgraphs induced by vertex sets $\{1,2,3,4\}, \{4,5\}, \{5,6\}, \{5,8,7\}$, and $\{8,9,10,11\}$, and three articulation points given by the vertices 4, 5, and 8. Notice that any set of edges whose removal disconnects G (i.e., an *edge cut* of G) must disconnect at least one of its blocks. Therefore, $\lambda(G) = \min\{\lambda(B) \mid B \in \mathcal{B}(G)\}$. For example, for the graph depicted in Figure 1, the blocks listed previously have an edge-connectivity of 3, 1, 1, 2, 2, respectively; thus, the graph's edge connectivity is 1.

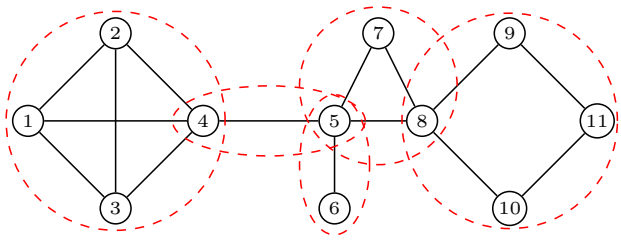


Figure 1: Example of a connected graph having five blocks and three articulation points.

Consider a connected subgraph H of graph G with vertex set $N \subseteq V$ such that $w(N) > r$. Given the weight of its vertices, it should be clear that any multicut $S \in \Delta(r)$ contains an edge cut of H, otherwise in the partition π associated with S, subgraph H would be contained by a cluster whose vertex weight is greater than r. Consequently, S should contain an edge cut of at least one block in $\mathcal{B}(H)$. If H is a tree, this property is captured by the tree cover inequalities (5b), as each pair of adjacent vertices comprises a block. For other connected subgraphs, there exist a family of valid inequalities that captures this property too. The following theorem summarizes this result.

Theorem 4. *Given a graph $G = (V, E)$, a positive integer weight vector $\mathbf{w} = [w_i]_{i \in V}$, a constant $r \in \mathbb{Z}$, a connected subgraph H of graph G with vertex set $N \subseteq V$ such that $w(N) > r$, the inequality*

$$\sum_{B \in \mathcal{B}(H)} \sum_{e \in E(B)} \frac{1}{\lambda(B)} y_e \geq 1 \tag{9}$$

is valid for $\mathscr{P}(\uparrow\Delta(r))$.

Proof. Notice that any multicut $S \in \Delta(r)$ (and thus any of its supersets) contains an edge cut $S' \subseteq S$ of at least one block $B \in \mathcal{B}(H)$ and since $|S'| \geq \lambda(B)$, the result follows.

Interestingly, this family of valid inequalities, which we denote the *block-decomposition inequalities*—not to be confused with the well-known path-block cycle inequalities [69]—not only generalizes the tree cover inequalities, but also other non-trivial valid inequalities that often induce facets of $\mathscr{P}(\uparrow\Delta(r))$, like the cycle inequalities [21, 27], the cycle-with-tails inequalities [27], as well as inequalities of other graph structures with well-known block decompositions like cacti. In fact, it is easy to see that for any subgraph H of G, if $w(N \setminus \{i\}) \leq r$, for all $i \in N$, the corresponding inequality (9) induces a proper face of polytope $\mathscr{P}(\uparrow\Delta(r))$, alas not always of dimension $m - 1$ (i.e., the inequality is not always facet inducing).

Since these inequalities include the tree cover inequalities, their separation problem is NP-hard (see Theorem 3). However, separating an integer candidate solution $\hat{\mathbf{y}}$, with $S = \{e \in E \mid \hat{y}_e = 1\}$, can be done in polynomial time via the following sequence of steps: (1) find a minimal tree cover T in $G(V, E \setminus S)$, (2) select any connected subgraph H that spans $V(T)$ in $G(V, E \setminus S)$, (3) identify $\mathcal{B}(H)$, and (4) compute the edge connectivity of each block. Step one can be done in $O(n \log n + m)$ (see Theorem 2), steps two and three can be done in $O(m+n)$ [37], and calculating the edge connectivity of each block in step four can be done in $O(nm + n^2 \log n)$ [71], although one may expect this last step to be significantly faster, as the blocks of H are often much smaller than G.

In the following section we move on to study two extended formulations to model $\min\{c(S) \mid S \in \uparrow\Delta(r)\}$ and compare their strength. We also provide some interesting relationships about their projection into the space of the edges.

4 Extended Formulations

Following the attacker-defender perspective introduced in Section 2, we begin our discussion with a formulation inspired by a class of problems known as network interdiction. In general, network interdiction comprises different types of adversarial games staged over networks, where an attacker—who typically plays first—performs a set of interdiction actions targeting the vertices or edges of the network to optimally deteriorate the defender's objective. In turn, the defender's objective is often the solution of a flow problem such as the shortest path problem, maximum flow problem, or minimum cost flow problem over the given network. We refer the interested reader to [65] for a comprehensive survey about network interdiction.

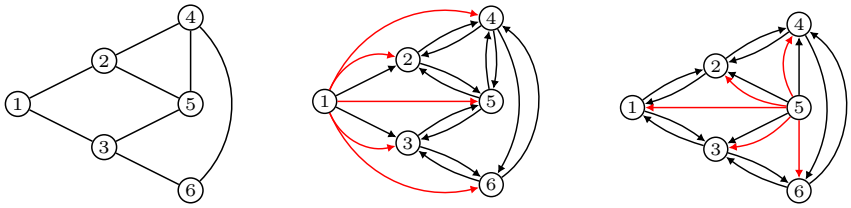
The formulation we describe in the following section uses multiple maximum flow subproblems to quantify the weight of the connected components of a given graph. Similar uses of flow formulations to analyze other connectivity properties of graphs (e.g., the number of connected components) can be found in [62]. Furthermore, reformulations that use similar network flow models for solving other problems related to graph partitioning have also been proposed in [30, 31, 36].

4.1 Maximum Flow Interdiction Formulation

The proposed formulation uses the following auxiliary digraph to model the defender's flow problem. Given a graph partitioning instance defined by graph $G = (V, E)$, we construct the following digraph D_k for each vertex $k \in V$. First, we define the vertex set of D_k as V. Second, we create a set of arcs A_E composed of two arcs (i, j) and (j, i) for each edge $\{i, j\} \in E$ with $i, j \neq k$ and an arc (k, i) for each $i \in \mathcal{N}(k)$. Third, we create a second set of arcs called A_k containing an auxiliary arc (k, i) for each vertex $i \in V \setminus \{k\}$. The resulting digraph is then $D_k = (V, A_E \cup A_k)$. Note that if the edge $\{k, i\}$ exists in E for some vertex $i \in V$, then there will be two parallel arcs (k, i) in D_k, one in A_E and the other in A_k. Whenever there is the need of differentiating these two arcs, we will use the arc set that contains them for clarification. Figure 2 depicts an example of the digraph construction for a 6-vertex graph.

Given digraph D_k associated with vertex $k \in V$, the weight vector $\mathbf{w} = [c_i]_{i \in V}$, an edge set $S \subseteq E$, and its corresponding incidence vector $\mathbf{y}^S$, consider the following network flow problem in which each vertex $i \in V \setminus \{k\}$ demands w_i units of a commodity that must be transported from source vertex k. For each edge $\{i, j\} \in E$, we use both y_{ij} and y_{ji} interchangeably to represent whether edge $\{i, j\} \in S$.

$$F(k, \mathbf{y}^S) = \max \sum_{(k,i) \in A_E} f^k_{ki} - \sum_{(i,j) \in A_E} y_{ij} f^k_{ij} \tag{10a}$$



(a) The original graph G. (b) The corresponing digraph D_1. (c) The corresponing digraph D_5.

Figure 2: Example of the construction of the auxiliary digraphs. In subfigures (b) and (c), the arcs in A_E and A_k are shown in black and red, respectively.

$$\text{s.t.} \quad \sum_{(k,i)\in A_E} f^k_{ki} + \sum_{(k,i)\in A_k} f'_{ki} = \sum_{i\in V\setminus\{k\}} w_i; \tag{10b}$$

$$f'_{ki} + \sum_{\{j:(j,i)\in A_E\}} f^k_{ji} - \sum_{\{j:(i,j)\in A_E\}} f^k_{ij} = w_i, \qquad i \in V\setminus\{k\}; \tag{10c}$$

$$f'_{kj} \geq 0, \quad (k,j) \in A_k; \tag{10d}$$

$$f^k_{ij} \geq 0, \quad (i,j) \in A_E. \tag{10e}$$

Here, variables f^k_{ij} and f'_{ki} represent the amount of flow traversing arcs $(i,j) \in A_E$ and $(k,i) \in A_k$, respectively; flow balance constraints (10b) and (10c) ensure that all the units of the commodity emanate from source vertex k and fulfill the demands of the other vertices. Furthermore, the first term in the objective function (10a) is set to maximize the flow emanating from k via the arcs in A_E, whereas the second term acts as a penalty for sending flow through the arcs associated with the edges in S. In what follows, we will use $(\mathbf{f}_k, \mathbf{f}'_k)$ in reference to the flow variables of problem k.

Proposition 3. *Given a graph $G = (V, E)$, a positive vector $\mathbf{w} = [w_i]_{i\in V}$, and an edge set $S \subseteq E$, $F(k, \mathbf{y}^S) + w_k$ yields the weight of the connected component that contains vertex k in $G(V, E \setminus S)$.*

Proof. First, note that the arcs in A_k guarantee that there is a path from k to all the other vertices in $V \setminus \{k\}$, thus (10a)-(10e) is always feasible. Second, to show that $F(k, \mathbf{y}^S)$ is finite it suffices to state that there is no arc in $A_E \cup A_k$ pointing towards source vertex k, thus there is no directed cycle in D_k that could be used to raise the objective function of this problem above $w(V \setminus \{k\})$.

Let $\overline{w}^k$ be the weight of the connected component that contains vertex k in graph $G(V, E \setminus S)$. To complete the proof we begin by showing that $F(k, \mathbf{y}^S) + w_k \geq \overline{w}^k$. For this case, it is easy to see that for each vertex $i \in V \setminus \{k\}$ that is connected with k in $G(V, E \setminus S)$, there exists a directed path from k to i in D_k composed of arcs in A_E that are not associated with any edge in S. Since such paths could be used to satisfy the demand of those vertices, then there exists a feasible solution to the flow problem whose objective is $\overline{w}^k - w_k$. Now, assume for a contradiction that $F(k, \mathbf{y}^S) + w_k > \overline{w}^k$. This implies that there is a vertex $i \in V \setminus \{k\}$ that is not connected to k in $G(V, E \setminus S)$ for which at least a part $0 < w'_i \leq w_i$ of its weight is counted in $F(k, \mathbf{y}^S)$. Notice that the flow from k to i must have been sent through paths that do not include arcs associated with edges in S, otherwise, the second term in the objective function would have subtracted the value of w' from $F(k, \mathbf{y}^S)$; however, if at least one path exists in D_k, that would imply that the edges in E associated with the arcs of such a path onnect k with i in $G(V, E \setminus S)$, a contradiction. □

Based on the result given in Proposition 3, we can use the maximum flow problem (10a)-(10e)

to define the following valid formulation for $\min\{c(S) \mid S \in \uparrow\Delta(r)\}$.

$$\begin{aligned} \min \quad & \sum_{\{i,j\}\in E} c_{ij} y_{ij} \\ \text{s.t.} \quad & F(k, \mathbf{y}) \leq r - w_k, \quad k \in V; \\ & y_{ij} \in \{0,1\}, \quad \{i,j\} \in E. \end{aligned}$$

To solve this problem, we can use the so-called *dualize-and-combine* approach often used to solve interdiction problems with linear follower problems [65]. To this end, we first define the dual of network flow problem (10a)-(10e) for a given vertex $k \in V$. Since the flow balance constraints (10b) and (10c) are linearly dependent [10], it is possible to remove constraint (10b) from this model. Now, letting α_{ki} be the dual variable of constraint (10c) for each $i \in V \setminus \{k\}$, the corresponding dual formulation follows.

$$\min \quad \sum_{i\in V\setminus\{k\}} w_i \alpha_{ki} \tag{11a}$$

$$\text{s.t.} \quad \alpha_{ki} \geq 1 - y_{ki}, \quad i \in \mathcal{N}(k); \tag{11b}$$

$$\alpha_{ki} - \alpha_{kj} \geq -y_{ij}, \qquad (i,j) \in A_E; i,j \neq k; \tag{11c}$$

$$\alpha_{kj} - \alpha_{ki} \geq -y_{ij}, \qquad (j,i) \in A_E; i,j \neq k; \tag{11d}$$

$$\alpha_{ki} \geq 0, \quad i \in V \setminus \{k\}. \tag{11e}$$

In this model, constraints (11b) are associated with flow variables f^k_{ki}, for each $i \in \mathcal{N}(k)$; constraints (11c) and (11d) are related to flow variables f^k_{ij} and f^k_{ji}, respectively, for arcs (i,j) and (j,i) in A_E with $i,j \neq k$; and constraints (11e) are associated with flow variables f'_{ki}, for $i \in \mathcal{N}(i)$.

Importantly, since the network flow problem is always feasible and bounded, so is (11a)-(11e). Furthermore, by the weak duality theorem, any feasible solution to this dual problem yields an upper bound on $F(k, \mathbf{y}^S)$ for any $S \subseteq E$. Therefore, by enforcing $\sum_{i\in V\setminus\{k\}} w_i \alpha_{ki} \leq r - w_k$ for each $k \in V$, one can use the dual problem of each vertex $k \in V$ to derive the following model. We will refer to it as the *flow interdiction formulation* (FLO).

$$(\text{FLO}): \quad \min \quad \sum_{\{i,j\}\in E} c_{ij} y_{ij} \tag{12a}$$

$$\text{s.t.} \quad \sum_{i\in V\setminus\{k\}} w_i \alpha_{ki} \leq r - w_k, \quad k \in V; \tag{12b}$$

$$\alpha_{ki} \geq 1 - y_{ki}, \quad k \in V, i \in \mathcal{N}(k); \tag{12c}$$

$$\left.\begin{aligned} \alpha_{ki} - \alpha_{kj} &\geq -y_{ij} \\ \alpha_{kj} - \alpha_{ki} &\geq -y_{ij} \end{aligned}\right\}, \qquad k \in V; \{i,j\} \in E; i,j \neq k; \tag{12d}$$

$$\alpha_{ki} \geq 0, \qquad k \in V, i \in V \setminus \{k\}; \tag{12e}$$

$$y_{ij} \in \{0,1\}, \quad \{i,j\} \in E. \tag{12f}$$

For our analysis regarding the FLO formulation, we first note that the bases of flow formulations like (10a)-(10e) are known to be the incidence matrices of spanning rooted out-trees, where the corresponding vertex k is the root [10]. Thus, any basic solution of these problems comprises the flow resulting from pushing w_i units through the unique path connecting k with i in the rooted out-tree associated with the corresponding basis.

The following proposition provides a relationship between the linear relaxations of both the TRI and FLO formulations.

Proposition 4. *Let* $\mathscr{P}_{\text{TRI}} := \{(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^{\binom{n}{2}} \times \mathbb{R}^m \mid (\mathbf{x}, \mathbf{y})$ *satisfies (1b)-(1d)}* $\}$ *denote the LP feasible region of the triangular formulation;* $\mathscr{P}_{\text{FLO}} := \{(\alpha, \mathbf{y}) \in \mathbb{R}^{n \times (n-1)} \times \mathbb{R}^m \mid (\alpha, \mathbf{y})$ *satisfies (12b)-(12e)}* $\}$ *denote the LP feasible region of the flow interdiction formulation; and* z^*_{TRI} *and* z^*_{FLO} *be the optimal solution of the corresponding LP relaxations. Then,* $z^*_{\text{TRI}} = z^*_{\text{FLO}}$.

Proof. As mentioned in Section 1, Nguyen et al., [55] showed that in formulation TRI, it is possible to reduce the number of triangular constraints (1c) by only considering vertex triplets for which at least two of such vertices are adjacent. Given a triplet $\{i, j, k\} \in \binom{V}{3}$, assume i and j are adjacent; then, using the relationship given by (1b), it is possible to reformulate the triangular constraints for such a triplet as follows:

$$x_{ik} - x_{jk} \leq y_{ij}, \tag{13a}$$

$$-x_{ik} + x_{jk} \leq y_{ij}, \tag{13b}$$

$$x_{ik} + x_{jk} + y_{ij} \leq 2 \tag{13c}$$

From this observation, it is easy to see that FLO is in fact a relaxation of TRI in which constraints (13c) are dropped and each variable x_{ik}, for $\{i, k\} \in \binom{V}{2}$, is replaced by a pair of variables α_{ik} and α_{ki}, as per constraints (12d). Furthermore, since for any solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \in \mathscr{P}_{\text{TRI}}$, the counterpart $(\bar{\alpha}, \bar{\mathbf{y}})$, where $\bar{\alpha}_{ik} = \bar{\alpha}_{ki} = \bar{x}_{ik}$, belongs to $\mathscr{P}_{\text{FLO}}$, hence $z^*_{\text{TRI}} \geq z^*_{\text{FLO}}$.

To complete the proof and show that $z^*_{\text{TRI}} \leq z^*_{\text{FLO}}$, it suffices to show that there exists an optimal LP solution $(\alpha^*, \mathbf{y}^*)$ for FLO, where $\alpha^*_{ik} = \alpha^*_{ki}$, for each pair of vertices $\{i, k\} \in \binom{V}{2}$; $\alpha^*_{ki} = 1 - y^*_{ki}$, for all $(k, i) \in A_E$; and $\alpha^*_{ik} + \alpha^*_{jk} + y^*_{ij} \leq 2$, for all $k \in V, \{i, j\} \in E$. To this end, we show that given an optimal LP solution $(\bar{\alpha}, \mathbf{y}^*)$, it is possible to use $\mathbf{y}^*$ to construct an optimal solution $(\mathbf{f}^*_k, \mathbf{f}'^*_k)$ for each $F(k, \mathbf{y}^*)$, whose corresponding dual solution α^* satisfy such conditions.

First, given a vertex $k \in V$ and its digraph D_k, we say that a vertex $i \in V \setminus \{k\}$ is *reachable* with respect to $\mathbf{y}^*$ if there is a path P connecting k with i in D_k, whose arc set $A(P)$ includes only arcs from A_E, and for which $\sum_{(u,v) \in A(P)} y^*_{uv} < 1$. Clearly, if i is unreachable in D_k with respect to $\mathbf{y}^*$, k is also unreachable from i in digraph D_i. For such a pair of vertices, one can set $f'^*_{ki} = w_i$ and $f'^*_{ik} = w_k$, for $F(k, \mathbf{y}^*)$ and $F(i, \mathbf{y}^*)$, respectively, which by complementary slackness (with respect to dual constraints (11e)) implies that $\alpha^*_{ki} = \alpha^*_{ik} = 0$.

Then, for each vertex $i \in V \setminus \{k\}$ that is reachable in D_k, it is easy to see that there is a basic solution for $F(k, \mathbf{y}^*)$ characterized by a rooted out-tree in which there is a path P connecting k with i that is a shortest path with respect to $\mathbf{y}^*$ (i.e., assuming that the length of each arc $(u, v) \in A_E$ is y^*_{uv}). Since such a path carries at least w_i units of flow, the corresponding flow variables are basic; then, by complementary slackness (with respect to dual constraints (12c)–(12d)), it is possible to set variable $\alpha^*_{ki} = 1 - \sum_{(u,v) \in A(P)} y^*_{uv}$. Furthermore, since path P exists in the opposite direction in digraph D_i, there is a basic solution for $F(i, \mathbf{y}^*)$ that either uses such a path or a different one with the same length with respect to $\mathbf{y}^*$ to connect i with k, implying that it is possible to set $\alpha^*_{ik} = \alpha^*_{ki}$. Also, since $1 - \alpha^*_{ki}$ accounts for the length of a shortest path from k to i in D_k with respect to $\mathbf{y}^*$, the fact that $\alpha^*_{ki} = 1 - y^*_{ki}$ for all $(k, i) \in A_E$ follows directly from $\mathbf{y}^*$ being optimal.

Finally, we proceed to prove that the proposed construction produces an optimal solution that satisfies (13c) (i.e., $\alpha^*_{ki} + \alpha^*_{kj} + y^*_{ij} \leq 2$). To begin, note that if either i or j (or both) are unreachable from k in D_k with respect to $\mathbf{y}^*$, the result follows directly. Now, assuming that both i and j are reachable, we consider the following two cases. Let P_1 and P_2 be the paths that connect k with i and j, respectively in the rooted out-tree associated with the constructed basic solution (i.e., both are shortest paths with respect to $\mathbf{y}^*$, connecting k with i and j, respectively). For the first case, assume that neither P_1 goes through j, nor P_2 goes through i. Based on the proposed construction,

we have:

$$\alpha^*_{ki} = 1 - \sum_{(u,v)\in A(P_1)} y^*_{uv}, \tag{14}$$

$$\alpha^*_{kj} = 1 - \sum_{(u,v)\in A(P_2)} y^*_{uv} \tag{15}$$

Furthermore, notice that the path $\bar{P}$ created by concatenating P_1 with the reverse of P_2 connects vertex i and j in D_i. Then, we have that:

$$1 - \sum_{(u,v)\in A(P_1)} y^*_{uv} - \sum_{(u,v)\in A(P_2)} y^*_{uv} \leq \alpha^*_{ij} = 1 - y^*_{ij}, \tag{16}$$

where the inequality follows because: (a) if j is unreachable from i in D_i, then

$$\sum_{(u,v)\in A(P_1)} y^*_{uv} + \sum_{(u,v)\in A(P_2)} y^*_{uv} \geq y^*_{ij} = 1;$$

and (b) if j is reachable from i in D_i, either $\bar{P}$ is a shortest path with respect to $\mathbf{y}^*$ in D_i or, there is a shorter path connecting such vertices with length $1 - \alpha^*_{ij}$. Then, combining (14)–(16), we obtain

$$\alpha^*_{ki} + \alpha^*_{kj} + y^*_{ij} \leq 2,$$

proving the result. For the second case, without loss of generality, assume P_2 goes through i. Then,

$$\alpha^*_{kj} = 1 - \sum_{(u,v)\in A(P_2)} y^*_{uv} = 1 - \sum_{(u,v)\in A(P_1)} y^*_{uv} - y^*_{ij} = \alpha^*_{ki} - y^*_{ij} \tag{17}$$

The first equality stands because both P_2 is a shortest path with respect to $\mathbf{y}^*$, connecting k with and j, respectively; the second equality follows from the optimality of $\mathbf{y}^*$, and the third equality from the way α^* is constructed. Then, reorganizing (17), we obtain

$$\alpha^*_{kj} = y^*_{ij} + \alpha^*_{ki},$$

which implies that $\alpha^*_{ki} + \alpha^*_{kj} + y^*_{ij} = 2\alpha^*_{kj} \leq 2$; thus proving the result. □

We will proceed to show that it is possible to identify the projection of FLO onto the $\mathbf{y}$-space, which allows us to easily compare its strength against other formulations. Furthermore, given the result from Proposition 4, one can also use it to analyze other properties of formulation TRI.

Consider the rooted out-tree that results from taking a tree cover $T \in \mathcal{T}(G, r)$, setting a vertex $k \in V(T)$ as the root, and replacing each edge in $\{i, j\} \in E(T)$ by an arc, either (i, j) or (j, i), so that there is a directed path between k and each vertex $i \in V(T) \setminus \{k\}$. Furthermore, let f^k_{ij} be the flow through arc (i, j) required to push w_i units of flow from k to i, for all $i \in V(T) \setminus \{k\}$. Using this construction we proceed to prove the following theorem.

Theorem 5. *The projection* $\text{proj}_{\mathbf{y}}(\mathscr{P}_{\text{FLO}})$ *of the flow interdiction formulation onto the* $\mathbf{y}$*-space is given by the following polytope:*

$$\sum_{\{i,j\}\in E(T)} f^k_{ij} y_{ij} \geq \sum_{i\in V(T)} w_i - r, \quad \forall T \in \mathcal{T}(G, r), k \in V(T); \tag{18a}$$

$$0 \leq y_{ij} \leq 1 \quad \forall \{i, j\} \in E. \tag{18b}$$

Proof. The prove the result using the Farkas' theorem of alternatives. Given any edge set $S \subseteq E$, the system of inequalities given by (12b)-(12e) has a solution if and only if there is at least one vertex $k \in V$ for which no solution of the form $(h_k, \mathbf{f}_k, \mathbf{f}'_k) \in \mathbb{R} \times \mathbb{R}^{2m} \times \mathbb{R}^{(n-1)}$ satisfies the following:

$$(r - w_k)h_k - \sum_{i \in \mathcal{N}(k)} (1 - y_{ki}) f^k_{ki} + \sum_{\{\{i,j\} \in E : i,j \neq k\}} y_{ij}(f^k_{ij} + f^k_{ji}) < 0; \tag{19a}$$

$$f'_{ki} + \sum_{\{j:\{j,i\} \in E\}} f_{ji} - \sum_{\{j:\{i,j\} \in E, j \neq k\}} f_{ij} - h_k w_i = 0, \qquad i \in V \setminus \{k\}; \tag{19b}$$

$$f'_{kj} \geq 0, \quad (k,j) \in A_k; \tag{19c}$$

$$f^k_{ij} \geq 0, \quad (i,j) \in A_E; \tag{19d}$$

$$h_k \geq 0. \tag{19e}$$

Therefore, such a system has no solution if the following condition is enforced:

$$(r - w_k)\bar{h}_k - \sum_{i \in \mathcal{N}(k)} (1 - y_{ki}) \bar{f}^k_{ki} + \sum_{\{\{i,j\} \in E : i,j \neq k\}} y_{ij}(\bar{f}^k_{ij} + \bar{f}^k_{ji}) \geq 0, \tag{20}$$

for every solution $(\bar{h}_k, \bar{\mathbf{f}}_k, \bar{\mathbf{f}}'_k)$ that satisfies constraints (19b)–(19e). Furthermore, it is easy to see that these constraints define a polyhedral cone centered at the zero solution $(h_k, \mathbf{f}_k, \mathbf{f}'_k) = (0, \mathbf{0}, \mathbf{0})$; thus, it suffices to simply enforce (20) on the extreme rays of such cone.

Considering the way in which the flow interdiction formulation is constructed, it should be no surprise that (19a)–(19e) resembles the constraint set of $F(k, \mathbf{y}^S)$. In fact, some of the properties of such formulation can be used to characterize the extreme rays of this cone. To this end, we first consider the extreme rays for which $h_k = 0$. For this case, the resulting system is a flow polytope defined over digraph D_k where all vertices have a demand/supply equal to zero. The extreme rays for such a case are given by all the characteristic vectors of the arcs comprising directed cycles in D_k. By construction, since there is no arc in D_k that enters vertex k, then there is no directed cycle in D_k containing k either. Let C be any directed cycle in D_k. Then, the corresponding extreme ray is given by $(h_k, \mathbf{f}_k, \mathbf{f}'_k) = (0, \mathbf{1}^{A(C)}, \mathbf{0})$, where $\mathbf{1}^{A(C)}$ is the characteristic vector of the arcs in C. Thus, enforcing (20) for such a ray, results in the inequality $\sum_{(i,j) \in A(C)} y_{ij} \geq 0$, which is a requirement already implied by the non-negativity of $\mathbf{y}$.

We now assume that $h_k = \hat{h} > 0$. In such a case, the system (19a)–(19e) becomes the constraint set of $F(k, \mathbf{y}^S)$, albeit with the demand w_i scaled by $\hat{h}$, for all $i \in V \setminus \{k\}$. Therefore, the extreme rays are then given by all basic feasible solutions of $F(k, \mathbf{y}^S)$, but with the respective flows scaled by $\hat{h}$ as well; for this reason, it suffices to consider the case where $h_k = 1$. As discussed previously, all basic feasible solutions of $F(k, \mathbf{y}^S)$ are associated with spanning out-trees rooted at vertex k. Assume $(\bar{\mathbf{f}}_k, \bar{\mathbf{f}}'_k)$ is one of such basic solutions, then the corresponding extreme ray of system (19a)–(19e) is given by $(1, \bar{\mathbf{f}}_k, \bar{\mathbf{f}}'_k)$. Let $\vec{T}$ denote the rooted out-tree associated with this solution and suppose $N_k \subseteq V$ is a vertex set composed of root k and the vertices that can be reached from k in $\vec{T}$ via a path that contains no auxiliary arcs from set A_k. Note that since the induced subtree $\vec{T}[N_k]$ contains vertex k, such a subtree is also a rooted out-tree, whereas $\vec{T}[V \setminus N_k]$ is a directed forest that contains some arcs from A_E. Figure 3 provides a few examples of rooted out-trees associated with some basic solutions of problem $F(1, \mathbf{y}^S)$, for the graph G of Figure 2.
Since the flows associated with extreme ray $(1, \bar{\mathbf{f}}_k, \bar{\mathbf{f}}'_k)$ are defined over the arcs of $\vec{T}$, it follows that enforcing (20) on this extreme ray results in:

$$r - w_k - \sum_{i \in \mathcal{N}(k)} (1 - y_{ki}) \bar{f}^k_{ki} + \sum_{\{\{i,j\} \in E : i,j \neq k\}} y_{ij}(\bar{f}^k_{ij} + \bar{f}^k_{ji}) =$$

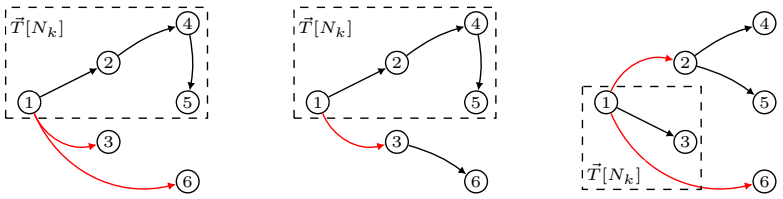


Figure 3: Example of multiple spanning rooted out-trees $\vec{T}$ associated with different basic solutions for the problem $F(1, \mathbf{y}^S)$, for the graph of Figure 2.

$$
\begin{aligned}
&= r - w_k - \sum_{i \in \mathcal{N}(k)} \bar{f}^k_{ki} + \sum_{i \in \mathcal{N}(k)} y_{ki} \bar{f}^k_{ki} + \sum_{\{\{i,j\} \in E : i,j \neq k\}} y_{ij} (\bar{f}^k_{ij} + \bar{f}^k_{ji}) = \\
&= r - \sum_{i \in N_k} w_i + \sum_{\{(i,j) \in A(\vec{T}[N_k])\}} y_{ij} \bar{f}^k_{ij} + \sum_{\{(i,j) \in A(\vec{T}[V \setminus N_k])\}} y_{ij} \bar{f}^k_{ij} \geq 0, \qquad (21)
\end{aligned}
$$

where the first summation in (21) comes from the fact that the demand w_i of vertex $i \in V \setminus \{k\}$ is pushed out of k via some arc $(k, j) \in A_E$; therefore, $\sum_{i \in N} w_i = \sum_{i \in \mathcal{N}(k)} \bar{f}^k_{ki} + w_k$. The other two summations result from the fact that the flows in $\bar{\mathbf{f}}_k$ through the arcs that are not in $\vec{T}_k$ are zero.

Now, consider an extreme ray $(1, \bar{\mathbf{f}}_k, \bar{\mathbf{f}}'_k)$ associated with a rooted out-tree $\vec{T}$ and the corresponding set N_k. It is easy to see that the demand of any vertex in $i \in V \setminus N_k$ can be sent through auxiliary arc $(k, i) \in A_k$. For example in Figure 3(a), the w_6 units demanded by vertex 6 are sent directly through auxiliary arc $(1, 6)$ instead of sending those via vertex 3, as shown in Figure 3(b). For such type of extreme rays, the inequality (21) then becomes

$$
r - \sum_{i \in N_k} w_i + \sum_{\{(i,j) \in A(\vec{T}[N_k])\}} y_{ij} \bar{f}^k_{ij} \geq 0. \qquad (22)
$$

Note that these inequalities for this type of extreme rays dominate the inequalities associated with the extreme rays for which $A(\vec{T}[V \setminus N_k]) \neq \emptyset$, as one can generate (21) by a linear combination of (22) and the nonnegativity constraints $y_{ij} \geq 0$ of the edges associated with the arcs in $A(\vec{T}[V \setminus N_k]) \neq \emptyset$.

Finally, to complete the proof, it suffices to note that if $w(N_k) > r$, the arcs of rooted subtree $\vec{T}[N_k]$ are associated with the edges of a tree cover $T \in \mathcal{T}(G, r)$. Moreover, since there is a basic solution for every spanning rooted out-tree in D_k; every tree cover $T \in \mathcal{T}(G, r)$ yields $|V(T)|$ inequalities (22) by setting each vertex $k \in V(T)$ as the root. □

Interestingly, the inequalities in (18a) are a weaker version of the knapsack tree inequalities discussed in Section 3. Here, note that the coefficients of the y_{ij} variables are essentially the weight $w(V(\vec{T}^j))$ of the subtree rooted at j. It is surprising however to see the deep connection of those inequalities with both the FLO and TRI formulations and it is perhaps the reason why TRI is widely used, performing generally well in practice.

4.2 Path-based Formulation

We complete this section by studying an alternative extended formulation named the *path-based formulation* (PAT). This model is defined over the same set of variables $(\mathbf{x}, \mathbf{y})$ as in TRI; i.e., variables $\mathbf{x}$, defined for every pair of vertices i and j in V, indicate whether vertices i and j belong to the same cluster of the partition; and variables $\mathbf{y}$, defined for every edge $\{i, j\} \in E$, define

the edges in the partition's multicut. As for TCF and FLO, some of the feasible solutions—yet suboptimal assuming $\mathbf{c} > \mathbf{0}$—may represent multicut supersets. In this formulation, we denote $\mathcal{P}^{ij}$ the set of all paths connecting vertices $i, j \in V$ in graph G. The PAT formulation then follows:

$$\text{(PAT):} \quad \min \quad \sum_{\{i,j\} \in E} c_{ij} y_{ij} \tag{23a}$$

$$\text{s.t.} \quad x_{ij} \geq 1 - \sum_{e \in E(P)} y_e, \quad P \in \mathcal{P}_G^{ij}, \, \{i,j\} \in \binom{V}{2}; \tag{23b}$$

$$\sum_{j \in V \setminus \{i\}} w_j x_{ij} \leq r - w_i, \quad i \in V; \tag{23c}$$

$$x_{ij} \geq 0, \quad \{i,j\} \in \binom{V}{2}; \tag{23d}$$

$$y_{ij} \in \{0,1\}, \quad \{i,j\} \in E, \tag{23e}$$

where the objective function (23a) minimizes the cost of the edges in the multicut; constraints (23b) ensure that if there is at least one path between i and j whose edges are not in the multicut, then both vertices are in the same cluster; and constraints (23c) bound the cluster weights.

Two important remarks about this model must be discussed. First, notice that there are no constraints that directly impose upper bounds on variables $\mathbf{x}$ with respect to the values of $\mathbf{y}$. This implies that it is possible for a solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ to be feasible for PAT and yet have a pair of vertices i and j from different clusters for which $\bar{x}_{ij} = 1$. We argue however that because of constraints (23b) and (23c), this may only happen if both i and j belong to clusters whose weight is strictly less than r. Since constraints (23c) are not binding for these vertices, there is room for having $x_{ij} > 0$. To avoid this type of solution, it is possible to introduce a new set of binary variables q_P, for each $P \in \mathcal{P}^{ij}$, indicating whether any of the edges of path P is in the multicut defined by variables $\mathbf{y}$, as well as the following set of inequalities for every pair of vertices i and j in V.

$$q_P \leq 1 - y_e, \quad e \in E(P), P \in \mathcal{P}^{ij}; \tag{24}$$

$$x_{ij} \leq \sum_{P \in \mathcal{P}^{ij}} q_P. \tag{25}$$

However, given the potential size of path-set P_G^{ij}, generating such inequalities directly might be computationally challenging. Moreover, since the actual partition is in fact given by the multicut, which is defined by $\mathbf{y}$, one might as well omit these inequalities and identify the actual values of $\mathbf{x}$ in a post-processing step. Furthermore, notice that by a similar argument, it is easy to see that the path constraints (23b) allow us to also relax the integrality requirement of the $\mathbf{x}$ variables.

The second remark is the fact that this extended formulation differs from TRI, VTC, and FLO in that it has potentially an exponential number of path constraints (23b); thus, solving it may require a constraint generation subroutine. It is easy to see however that separating these constraints can be done in polynomial time and therefore the LP relaxation of PAT can be solved in polynomial time as well [33]. It is well-known that formulations that use similar path constraints (see [52, 77]) use separation subroutines that run in $O(nm)$ for integer solutions and $O(m^2 \log n)$ for fractional solutions. Such implementations can be trivially adapted for PAT.

To the best of our knowledge, formulations like PAT have not been used before for solving graph partitioning problems like the one we consider in this paper. At first, we found this to be rather surprising because integer models that use path constraints similar to (23b) have been quite successful in other contexts that require identifying connected components of graphs] such as when

solving the k-edge survivability problem [52], districting [73], and various critical element detection problems [24, 59, 77]. Moreover, in some cases, path-based formulations are known to produce stronger LP relaxations compared to models that use triangular constraints (1c). Indeed, this is the case of some critical element detection problems that seek to find a set of vertices whose removal from a graph minimizes the number of connected vertex pairs left in the residual graph [24, 77]. In such cases though, the path constraints differ slightly from (23b), for in those models vertices instead of edges are removed from the graph.

From this perspective, one may wonder whether the PAT formulation produces a better LP relaxation than formulations FLO and TRI. If so, in addition to having a stronger model, one may also study the projection of the PAT formulation onto the space of the edges as it may yield other nontrivial inequalities to be studied. Alas, no. The following theorem shows that in fact, these formulations have the same LP relaxation and share the same projection.

Theorem 6. *Let* $\mathscr{P}_{\text{PAT}} := \{(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^{\binom{n}{2}} \times \mathbb{R}^m \mid (\mathbf{x}, \mathbf{y})$ *satisfies (23b)-(23d)*$\}$ *denote the LP feasible region of the path-based formulation. Then,* $\text{proj}_{\mathbf{y}}(\mathscr{P}_{\text{PAT}}) = \text{proj}_{\mathbf{y}}(\mathscr{P}_{\text{FLO}})$.

Proof. We will use the same approach as for Theorem 5. The Farkas' alternative of the system given by (23b)-(23d) is the following system:

$$-\sum_{\{i,j\}\in\binom{V}{2}} \sum_{P\in\mathcal{P}^{ij}} \left(1 - \sum_{e\in E(P)} y_e\right) \gamma_P^{ij} + \sum_{i\in V} (r - w_i)\beta_i < 0; \tag{26a}$$

$$-\sum_{E(P)\in\mathcal{P}^{ij}} \gamma_P^{ij} + w_j\beta_i + w_i\beta_j \geq 0, \quad \{i,j\} \in \binom{V}{2}; \tag{26b}$$

$$\gamma_P^{ij} \geq 0, \quad P \in \mathcal{P}^{ij}, \{i,j\} \in \binom{V}{2}; \tag{26c}$$

$$\beta_i \geq 0, \quad i \in V, \tag{26d}$$

where the variables γ and β are associated with constraints (23b) and variables (23c), respectively. Then, (23b)-(23d) is nonempty if the following condition is enforced

$$-\sum_{\{i,j\}\in\binom{V}{2}} \sum_{P\in\mathcal{P}^{ij}} \left(1 - \sum_{e\in E(P)} y_e\right) \gamma_P^{ij} + \sum_{i\in V} (r - w_i)\beta_i \geq 0, \tag{27}$$

for every solution $(\bar{\gamma}, \bar{\beta})$ that belongs to the polyhedral cone (26b)–(26d). We now proceed with the characterization of the extreme rays of this cone. First, note that having $\beta = \mathbf{0}$ forces $\gamma = \mathbf{0}$; therefore, any ray of this cone satisfies $\sum_{i\in V} \beta_i > 0$.

A common way of identifying the extreme rays of a cone consists of introducing a constraint that bounds the cone so that the extreme rays become the extreme points of the resulting polytope [10]. To this end, we impose the following constraint

$$\sum_{i\in V} \beta_i = 1 \tag{28}$$

to bound cone (26b)–(26d). Let $(\bar{\gamma}, \bar{\beta})$ be a solution of this polytope that satisfies the following:

(a) there is exactly one $k \in V$ for which $\bar{\beta}_k = 1$;

(b) for all $P \in \mathcal{P}^{ij}$, with $\{i,j\} \in \binom{V\setminus\{k\}}{2}$, $\gamma_P^{ij} = 0$; and

(c) for each $i \in V \setminus \{k\}$, either $\bar{\gamma}_P^{ki} = 0$ for all $P \in \mathcal{P}^{ki}$, or there is exactly one path $P_i \in \mathcal{P}^{ki}$ for which $\bar{\gamma}_{P_i}^{ki} = w_i$. We call such path "open".

Claim 1. *Any solution $(\bar{\gamma}, \bar{\beta})$ of the polytope given by (26b)–(26d), (28) satisfying conditions (a),(b), and (c) is an extreme point.*

Proof. Verifying feasibility is trivial. Thus, to prove the claim, it suffices to show that

$$n + \sum_{\{i,j\} \in \binom{V}{2}} |\mathcal{P}^{ij}|$$

constraints of this polytope are active at $(\bar{\gamma}, \bar{\beta})$. To see this, note first that conditions (a) and (b) ensure that the $n-1$ nonegativity constraints (26d) for all $i \in V \setminus \{k\}$ are active, as well as all the $|\mathcal{P}^{ij}|$ nonnegativity constraints (26c), for all $\{i,j\} \in \binom{V\setminus\{k\}}{2}$. Then, condition (c) ensures that for each vertex $i \in V \setminus \{k\}$, either all the $|\mathcal{P}^{ki}|$ nonegativity constraints (26d) are active, or all except for the one associated with open path P_i. Since $\bar{\gamma}_{P_i}^{ki} = w_i$, the corresponding constraint (26b) is active. Finally, since the imposed bound (28) is also active, the claim follows. ■

It is easy to see that this property is only true for solutions that satisfy these three conditions.

We now proceed to analyze the constraints resulting from evaluating these extreme rays over expression (27). For a given extreme ray $(\bar{\gamma}, \bar{\beta})$ of (26b)–(26d), assume $\bar{\beta}_k = 1$ for some $k \in V$. First, consider the case where $\gamma = \mathbf{0}$. Evaluating such ray in (27) results in the tautology $r \geq w_k$. Second, assuming $\gamma > \mathbf{0}$, let $V' \subseteq V \setminus \{k\}$ be the set of vertices such that, for each $i \in V'$, there is an open path $P_i \in \mathcal{P}^{ki}$ for which $\bar{\gamma}_{P_i}^{ki} = w_i$. Then, evaluating such a ray in (27) yields

$$-\sum_{i \in V'} \left(1 - \sum_{e \in E(P_i)} y_e\right) \bar{\gamma}_{P_i}^{ki} + (r - w_k)\bar{\beta}_k \geq 0,$$

and thus,

$$\sum_{i \in V'} \sum_{e \in E(P_i)} w_i y_e \geq \sum_{i \in V'} w_i + w_k - r, \tag{29}$$

after replacing the corresponding values of $\bar{\gamma}$ and $\bar{\beta}$.

Now, to prove the desired equivalency, consider a tree cover $T \in \mathcal{T}(G, r)$ and let $k \in V(T)$ be any of its vertices. Since there is a unique path connecting k with each vertex $i \neq k$ in T, one can use such paths to create a ray satisfying the three conditions given before. Evaluating such a ray in (29) yields the constraint (18a) associated with tree cover T.

To complete the proof, consider a ray $(\bar{\gamma}, \bar{\beta})$ with $\bar{\beta}_k = 1$ and $w(V') + w_k > r$, for which the union of the open paths P_i, for all $i \in V'$, does not form a tree. In this case, there must exist two vertices i and j in V' such that the open path P_i for which $\bar{\gamma}_{P_i}^{ki} = w_i$ goes though j before reaching i, but the subpath P_j of P_i that goes from k to j is different than the open path P'_j for which $\bar{\gamma}_{P'_j}^{kj} = w_j$. Notice then that there is an alternative path $P'_i \in \mathcal{P}_G^{ki}$ that results from replacing subpath P_j with P'_j in P_i. Therefore, we can create the alternative extreme ray $(\hat{\gamma}, \hat{\beta})$ by setting P'_i and P'_j as the open paths for i and j, respectively, as well as the alternative extreme ray $(\tilde{\gamma}, \tilde{\beta})$ by setting P_i and P_j as the open paths for i and j. By construction, both $(\hat{\gamma}, \hat{\beta})$ and $(\tilde{\gamma}, \tilde{\beta})$ satisfy (a), (b), and (c). Moreover, it is easy to see that the constraint that results from evaluating $(\bar{\gamma}, \bar{\beta})$ in (29) can be produced as a linear combination of the constraints given by $(\hat{\gamma}, \hat{\beta})$ and $(\tilde{\gamma}, \tilde{\beta})$. This procedure can be repeated as needed showing that it suffices to enforce (29) for extreme rays with open paths that compose trees. This completes the proof. □

Theorems 5 and 6 showed that formulations TRI and PAT have the same power in the sense that they yield the same LP relaxation; therefore, their effectiveness is in essence given exclusively by their size. Since the number of path constraints required to define PAT may grow exponentially with the size of the graph, one may see TRI as the better option. Interestingly, the number of path constraints (23b) that need to be generated when solving PAT in practice is often small compared to the actual number of constraints; especially when G is sparse. As will be shown in Section 6, it is not uncommon for PAT to outperform TRI when solving problems of multiple sizes. Moreover, as will be shown in the following sections, the separation of the path constraints in TRI can often be interweaved with the separation subroutine of other valid inequalities to reduce its solution time.

The results from these theorems also provide a way of comparing the strength of TRI and PAT versus TCF. Given a minimal tree cover $T \in \tilde{\mathcal{T}}(G, r)$, note that the constraint (5b) in TCF associated with such a tree dominates its counterpart in (18a). Then, one may expect TCF to produce a better LP relaxation than TRI and PAT. Nevertheless, the fact that constraints (18a) are defined for all tree covers, not just the ones that are minimal, as in (5b), often gives the upper hand to TRI and PAT, particularly when r is significantly smaller than $w(V)$.

5 Graph Partitioning on Trees

One commonality of the TRI, FLO, PAT, and TCF formulations is that their constraints focus on partitioning the tree structures of the graph. However, none of these formulations project to the convex hull of the problem on trees. Given that separating the tree cover inequalities is NP-hard for general graphs, it is unlikely that there exists a compact extended formulation whose projection implies such constraints [14]. Nevertheless, studying linear programs to solve the problem on trees may be useful to derive additional inequalities that can strengthen the aforementioned formulations. This motivates the results of this section.

Most of the literature devoted to solving the problem on trees has focused on dynamic programming. In this section, we provide a dynamic program that runs in pseudo-polynomial time for the version with non-negative vertex weights and edge costs, and in polynomial time for the case where the vertex weights are equal to one. While our approach shares the same worst-case complexity as its predecessors, its value comes from a recursive structure that can be used to derive linear programming formulations. Furthermore, we will also introduce a separation algorithm based on such formulation to generate strong inequalities associated with tree covers for the problem on general graphs. We begin by providing a brief discussion about the complexity of graph partitioning on trees and review some previous work.

5.1 The Complexity of Graph Partitioning on Trees

To the best of our knowledge, the first algorithm for this problem was introduced by Lukes in the mid-70s with a worst-case complexity of $O(r^2 n)$ time [47]. A few years later, Kundu and Misra developed an $O(n)$-algorithm for the specific case where both the vertex weights and edge costs are equal to one, as well as for the case with vertex weights equal to one and edge costs that are equivalent at each level of a rooted tree and monotonically increase with respect to the tree's depth [42]. From the computational perspective, the problem has been proven to be weakly NP-hard even on stars and binary trees by using a reduction from subset-sum [11]. Table 1 provides a summary of the problem's computational complexity. Our algorithm maintains the same complexity as Lukes' algorithm for trees with nonnegative vertex weights and edge costs.

There have been several dynamic programming algorithms to solve other versions of graph partitioning on trees as well; some examples are the version with both lower and upper bounds

Table 1: Complexity results for the graph partitioning on trees.

c_e	w_i	Complexity	Citation
$= 1$	$= 1$	solvable in $O(n)$	[42]
$= 1$	≥ 0	solvable in $O(n)$	[42]
≥ 0	$= 1$	solvable in $O(n^3)$	[47]
≥ 0	≥ 0	weakly NP-Hard, $O(r^2 n)$	[11, 47]

on the cluster weight [39] and the bottleneck version which minimizes the maximum weight of the clusters [2]. Also, besides studying the problem on trees, several papers have focused on approaches for solving graph partitioning on other basic structures like paths, cycles, and cacti [11, 12]. A common approach among all of these algorithms is to first order the tree from a selected root vertex. Then, starting from the leaves, begin computing non-dominated partitions of the subtrees at the lower levels and aggregating the partitions at the parent vertices, until the root is reached. This approach is also popular for related problems such as critical element detection on trees [7, 61].

5.2 Dynamic Program for Trees

In this section, we first require to expand the notation about rooted out-trees introduced in Section 3. For a given input tree $T = (V, E)$, let $\vec{T}$ be the rooted out-tree that stems from any arbitrary root vertex $k \in V$ and whose arcs represent the edges in E oriented accordingly. Given a set $S \subseteq E$, let $\vec{T}(S)$ represent the forest that results from removing the arcs in S from $\vec{T}$. The number of children of vertex $i \in V$ is thus equal to its out-degree in $\vec{T}$, which is denoted $\deg^+(i)$. We will assume that the children of each vertex $i \in V$ are given in any arbitrary order and will refer to the t^{th} child as i_t. We call $\vec{T}^i_t$ the rooted subtree of $\vec{T}^i$ that only contains the branches of the first t children of i. This definition implies that $\vec{T}^i_{\deg^+(i)} = \vec{T}^i$. Figure 4(a) provides an example of rooted subtrees.

Our dynamic program records a set of the non-dominated multicut solutions for each rooted subtree $\vec{T}^i$, starting from the leaves, traversing up the tree, and terminating at the root vertex k. For each vertex $i \in V$ and each value $o = 1, \ldots, r$, we maintain the value $f(i, o)$ that represents the minimum cost of a multicut S of $\vec{T}^i$ such that the cluster that contains i in $\vec{T}^i(S)$ has a weight equal to o and every other cluster therein has weight less than r. Then, the value $\min_{o=1,\ldots,r}\{f(k, o)\}$ represents the optimal cost of a multicut $S \in \Delta(r)$ for T. Figure 4(b) provides an example of the calculation of $f(i, o)$, assuming both unit vertex weights and edge costs.

In order to initialize our dynamic program, we have several boundary cases to consider. First, for each vertex $i \in V$ and all $o < w_i$, we set $f(i, o) = \infty$, as any cluster that contains i must have a weight of at least w_i. Next, we set $f(i, o) = \infty$ for all $o > w(V(\vec{T}^i))$ and $f(i, w(V(\vec{T}^i)) = 0$. Finally, for each leaf vertex $i \in V$, we set $f(i, w_i) = 0$ and $f(i, o) = \infty$, otherwise.

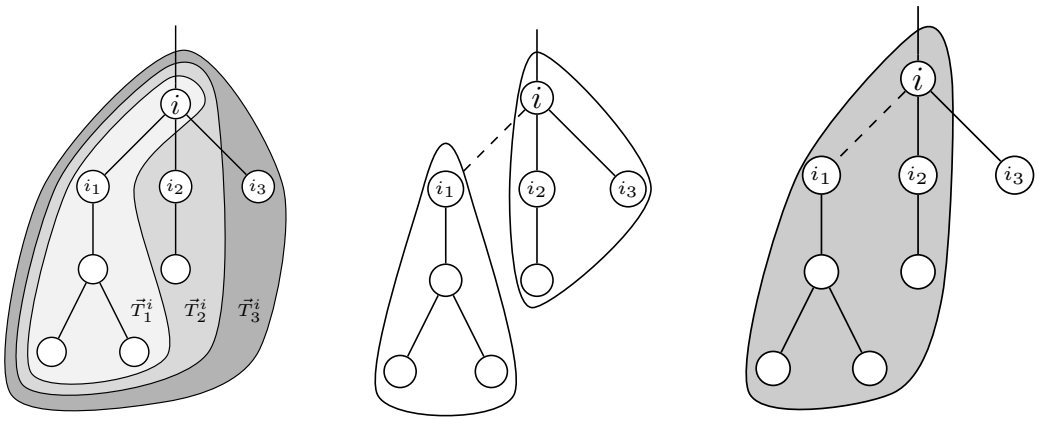
In order to calculate $f(i, o)$, we define the following auxiliary variables. Let $g(i, t, o)$ represent the minimum cost of a multicut S of $\vec{T}^i_t$ such that the weight of the cluster containing vertex i in $\vec{T}^i_t(S)$ is equal to o and every other cluster therein has a weight less than r. Figure 4(c) shows an example of the calculation of $g(i, t, o)$, assuming both unit vertex weights and edge costs.

Given these variables and their corresponding base cases, the proposed dynamic program comprises the following recursions for each $i \in V$:

$$f(i, o) = g(i, \deg^+(i), o); \tag{30a}$$

$$g(i, 1, w_i) = c_{i,i_1} + \min_{q=1,\ldots,r}\{f(i_1, q)\}; \tag{30b}$$

$$g(i, t, w_i) = c_{i,i_t} + g(i, t-1, w_i) + \min_{q=1,\ldots,r}\{f(i_t, q)\}, \quad t = 2, \ldots, \deg^+(i); \tag{30c}$$



(a) The rooted subtree $\vec{T}_i$. (b) The solution $f(i,4)=1$. (c) Subtree from $g(i,2,3)=1$.

Figure 4: A subtree $\vec{T}_i$ with a visual representation $f(i,o)$ and $g(i,t,o)$.

$$g(i,1,o) = f(i_1, o - w_i); \tag{30d}$$

$$g(i,t,o) = \min \begin{cases} c_{i,i_t} + g(i,t-1,o) + \min\limits_{q=1,\dots,r} \{f(i_t,q)\} \\ \min\limits_{q=1,\dots,o} \{g(i,t-1,q) + f(i_t, o-q)\} \end{cases}, \quad t = 2,\dots,\deg^+(i); \tag{30e}$$

where (30a) follows because $\vec{T}^i_{\deg^+(i)} = \vec{T}^i$. Also, expressions (30b) and (30c) correspond to the cases where the cluster of the resulting partition that contains i is just the vertex itself and expressions (30d) and (30e) comprise all other cases. We also note that the representation of this dynamic program can be simplified and expressed only in terms of (30a) and (30e) by adding an artificial "zeroth" child for each internal vertex with $w_{i_0} = 0$ and $c_{i,i_0} = 0$ and setting $g(i,0,o) = 0$, if $o = w_i$ and $g(i,0,o) = \infty$ otherwise. We use this representation in our linear program in the next section.

Proposition 5. *Given a tree $T = (V,E)$, where $|V| = n$, a positive weight vector $\mathbf{w} = [w_i]_{i \in V}$, a constant $r \in \mathbb{Z}$, and a positive cost vector $\mathbf{c} = [c_e]_{e \in E}$, computing the dynamic program (30a)-(30e) takes $O(r^2 n)$ time and $O(rn)$ space.*

Proof. To compute the space complexity of our dynamic program we must analyze the storage required for all of the values $f(i,o)$ and $g(i,j,o)$. We can represent the values for f in a $n \times r$ matrix, thus we require $O(rn)$ space to store f. For each value of $o = \{1\dots,r\}$, there is a g value for each child of every vertex. Since our graph is a tree, $|E| = n-1$. Thus, we must store $O(rn)$ values for g. Therefore the space complexity of our dynamic program is $O(rn)$. The time complexity of our dynamic program is the effort required to fill f and g. For each value we fill in g may involve taking a minimum of values from $q = 1,\dots,r$. Therefore, the time complexity of our dynamic program is $O(r^2 n)$. Furthermore, the backtracking procedure in order to recover the edges in S can be accomplished in $O(rn)$. □

Let non-negative, continuous variables f_{io} and g_{ito} represent the values $f(i,o)$ and $g(i,t,o)$, respectively, and $L \subseteq V$ represent the set of leaves in V. The *value* linear program associated with the above recursion is then:

$$\max \; z \tag{31a}$$

$$\text{s.t.} \quad z \le f_{ko} \quad o = 1,\dots,r; \tag{31b}$$

$$f_{io} = g_{i,\deg^+(i),o} \quad i \in V \setminus L,\, o = 1, \ldots, r; \tag{31c}$$

$$g_{ito} \leq c_{i,i_t} + g_{i,t-1,o} + f_{i_t q} \quad i \in V,\, t = 1, \ldots, \deg^+(i),\, o = 1, \ldots, r,\, q = 1, \ldots, r; \tag{31d}$$

$$g_{ito} \leq g_{i,t-1,q} + f_{i_t,o-q} \quad i \in V,\, t = 1, \ldots, \deg^+(i),\, o = 1, \ldots, r,\, q = 1, \ldots, o; \tag{31e}$$

$$f_{io} \geq 0 \quad i \in V,\, o = 1, \ldots, r; \tag{31f}$$

$$g_{ito} \geq 0 \quad i \in V,\, t = 1, \ldots, \deg^+(i),\, o = 1, \ldots, r. \tag{31g}$$

The optimal value of $z^* = \min_{o=1,\ldots,r}\{f(k,o)\}$ represents the optimal cost of any feasible multicut of input tree T. The dual formulation of (31a)-(31g) is known as the *policy* linear program and any of its feasible solutions represents a feasible multicut for the graph partitioning problem on T. Let α_o, β_{io}, γ_{it}^{oq}, and ρ_{it}^{oq} be the dual variables for (31b)-(31e), respectively. Then, the policy linear program is as follows:

$$\min \quad \sum_{i \in V} \sum_{t=1}^{\deg^+(i)} \sum_{o=1}^{r} \sum_{q=1}^{r} c_{i,i_t} \gamma_{it}^{oq} \tag{32a}$$

$$\text{s.t.} \quad \sum_{o=1}^{r} \alpha_o \geq 1; \tag{32b}$$

$$-\alpha_o + \beta_{ko} \geq 0 \quad o = 1, \ldots, r; \tag{32c}$$

$$\beta_{i_t,q} - \sum_{o=1}^{r} \gamma_{it}^{oq} - \sum_{o=1}^{r-q} \rho_{it}^{o+q,o} \geq 0 \quad i \in V,\, t = 1, \ldots, \deg^+(i),\, q = 1, \ldots, r; \tag{32d}$$

$$-\beta_{io} + \sum_{q=1}^{r} \gamma_{is}^{oq} + \sum_{q=1}^{o-1} \rho_{is}^{o,o-q} \geq 0 \quad i \in V \setminus L,\, o = 1, \ldots, r; \tag{32e}$$

$$\sum_{q=1}^{r} \gamma_{it-1}^{oq} - \sum_{q=1}^{r} \gamma_{it}^{oq} + \sum_{q=1}^{o-1} \rho_{it-1}^{o,o-q} - \sum_{q=1}^{r-o} \rho_{it}^{o+q,o} \geq 0 \quad i \in V,\, t = 1, \ldots \deg^+(i),\, o = 1, \ldots, r; \tag{32f}$$

$$\alpha_o \geq 0 \quad o = 1, \ldots, r; \tag{32g}$$

$$\gamma_{it}^{oq} \geq 0 \quad i \in V,\, t = 1, \ldots, \deg^+(i),\, o = 1, \ldots, r, q = 1, \ldots, r; \tag{32h}$$

$$\rho_{it}^{oq} \geq 0 \quad i \in V,\, t = 1, \ldots, \deg^+(i),\, o = 1, \ldots, r, q = 1, \ldots, o. \tag{32i}$$

In this formulation, the variables γ_{it}^{oq} represent the decision of whether or not arc (i, i_t) is in the multicut, such that the weight of the cluster containing i is o and the weight of the cluster containing i_t is q. Then, this formulation can be extended to include the multicut variables $\mathbf{y}$, resulting in the following linear program named the *tree dynamic program formulation* (TDP).

$$\text{(TDP):} \quad \min \quad \sum_{\{i,j\} \in E} c_{ij} y_{ij} \tag{33a}$$

$$\text{s.t.} \quad -\sum_{o=1}^{r} \sum_{q=1}^{r} \gamma_{it}^{oq} \geq -y_{i,i_t} \quad i \in V,\, t = 1, \ldots, \deg^+(i); \tag{33b}$$

$$(32b) - (32i) \tag{33c}$$

$$y_{ij} \geq 0 \quad \{i,j\} \in E. \tag{33d}$$

Let $\mathscr{P}_{\text{TDP}}(T)$ denote the LP feasible region of TDP for a given tree T. There are several observations that can be drawn from this formulation. First, since the number of variables and

constraints depend on the cluster-weight bound r, TDP can become impractical for cases where such a bound is big. One particular case worth of analysis is when the vertex weights are all equal to one, as such case yields a polynomially-sized formulation; we will pay special attention to this case in the computational experiments. Second, because one can solve the problem on trees faster using the dynamic program directly, the value of TDP comes from using some of its features to help solving the problem on general graphs. One naïve approach would be to introduce a block of variables and constraints of the form (33b)-(33c) for each spanning tree of G. This indeed will result in a formulation that yields a better (at least not worse) linear relaxation bound compared to TRI, TCF, FLO, and PAT. However, even for sparse graphs, this approach might not be viable because the number of spanning trees of a graph grows rapidly as the graph's density grows.

A natural alternative would be to utilize this formulation to strengthen the others. Ideally, given a tree cover $T \in \mathcal{T}(G, r)$, one would like to find the projection $\text{proj}_{\mathbf{y}}(\mathscr{P}_{\text{TDP}}(T))$ into the space of the $\mathbf{y}$ variables and use the resulting constraints to improve the LP bound of the other formulations. Alas, finding a general representation to fully describe such a projection for trees with arbitrary weights appears to be a more complex task than for other projections, such as $\text{proj}_{\mathbf{y}}(\mathscr{P}_{\text{PAT}})$ and $\text{proj}_{\mathbf{y}}(\mathscr{P}_{\text{FLO}})$. Nevertheless, we will show that it is possible to sequentially separate inequalities from such projection à la Benders [53]. To this end, consider the dual of (33a)-(33d):

$$\max \; z - \sum_{\{i,j\} \in E} h_{ij} \bar{y}_{ij} \tag{34a}$$

$$\text{s.t.} \quad z \leq f_{ko} \quad o = 1, \ldots, r; \tag{34b}$$

$$f_{io} = g_{i,\deg^+(i),o} \quad i \in V \setminus L,\, o = 1, \ldots, r; \tag{34c}$$

$$g_{ito} \leq c_{i,i_t} + g_{i,t-1,o} + f_{i_t,q} + h_{i,i_t} \quad i \in V,\, t = 1, \ldots, \deg^+(i),\, o = 1, \ldots, r,\, q = 1, \ldots, r; \tag{34d}$$

$$g_{i,t,o} \leq g_{i,t-1,q} + f_{i,t,o-q} \quad i \in V,\, t = 1, \ldots, \deg^+(i),\, o = 1, \ldots, r,\, q = 1, \ldots, o; \tag{34e}$$

$$f_{io} \geq 0 \quad i \in V,\, o = 1, \ldots, r; \tag{34f}$$

$$g_{ito} \geq 0 \quad i \in V,\, t = 1, \ldots, \deg^+(i),\, o = 1, \ldots, r; \tag{34g}$$

$$h_{it} \geq 0 \quad \{i,t\} \in E; \tag{34h}$$

where $\mathbf{h}$ are the dual variables associated with constraints (33b). Given any fractional solution $\bar{\mathbf{y}}$ and a tree cover $T \in \mathcal{T}(G, r)$, an inequality valid for $\text{proj}_{\mathbf{y}}(\mathscr{P}_{\text{TDP}}(T))$ violated by $\bar{\mathbf{y}}$ can be generated by identifying a direction of unboundedness of this dual program—ideally an extreme ray. Let $(\bar{z}, \bar{\mathbf{f}}, \bar{\mathbf{g}}, \bar{\mathbf{h}})$ one of such directions. From (34a), the corresponding inequality can be obtained by enforcing $\bar{z} - \sum_{\{i,j\} \in E} \bar{h}_{ij} y_{ij} \leq 0$, resulting in an inequality of the form:

$$\sum_{e \in E} \bar{h}_e y_e \geq \bar{z}; \tag{35}$$

We will refer to them as the *dynamic-programming (DP) tree inequalities*. Importantly, since the separation of these cuts requires giving a tree $T \in \mathcal{T}(G, r)$ as an input, one might as well chose a spanning tree, as the resulting cuts will also be valid for any of its subtrees. Also, since these cuts are used to strengthen other formulations, we can heuristically do this by a greedy selection as follows. First, given a fractional solution $\bar{\mathbf{y}}$, we find the minimum spanning tree T with respect to edge weights $\bar{y}_{ij}$ for $\{i,j\} \in E$. This part can be accomplished in $O(m \log n)$ time. Next, we randomly select a root node k and perform a breadth-first search in order to obtain the rooted tree $\vec{T}_k$. For the given tree, we solve the separation algorithm (34a)-(34h) and if the solution is unbounded, we add the corresponding inequality from the given extreme ray.

The TDP formulation is in fact an extended formulation that is also compact for fixed values of r, whose projection onto the $\mathbf{y}$-space yields the convex hull of the graph partitioning problem associated with the input tree T. It is important to notice however that not all the DP tree inequalities associated with extreme rays of (34a)-(34h) may induce facets of such a polytope. Indeed, one may expect the sequential generation discussed above to occasionally produce valid cuts that might be dominated by others, despite properly separating fractional solutions. In practice, the procedure may require generating some extra cuts, but the effect on the LP strength should persist. One may attempt to explore post-processing steps that aim to find nondominated cuts [63], however, that is out of the scope of this paper.

6 Computational Study

In this section, we conduct computational experiments to test the different formulations and developments presented in this work. In particular, with our experiments we would like to gather empirical data to address the following questions:

1. How do the LP relaxations of all the formulations fare when solving sparse instances of multiple sizes? While the strength of the LP relaxations is only one of the multiple factors that affect the quality of a mathematical program, it is in fact one of the most important. We are particularly interested in comparing the TCF formulation, which is defined by the minimal tree cover inequalities, and the TRI/FLO/PAT, which project into a weaker version of the knapsack tree inequalities. Here, we omit the VTC formulation—discussed in Section 1—as its LP bound is always zero for the type of instances used in these experiments.

2. What is the impact of adding the DP tree inequalities for the relaxation on general graphs? We are particularly interested in them as they suffice to solve the problem on trees. Since the generation of these cuts requires providing a tree cover as an input, we limit the scope to a heuristic separation based on Prim's algorithm [3].

3. How does the strength of the block-decomposition inequalities compare to that of the minimal tree cover inequalities? This is particularly important after noticing that all the formulations we analyze target tree structures exclusively. As with the DP tree inequalities, since the algorithm requires a minimal tree cover as an input, we use a heuristic separation as well.

4. In terms of the strong knapsack tree inequalities, since one could add the corresponding cuts associated with tree covers of multiple sizes, we would like to know whether the size has a noticeable effect on the quality of the cuts.

5. Finally, we would like to analyze the difference in the performance of all formulations when strengthened with the different cuts we study throughout the paper.

Before presenting the experiments, we provide a brief discussion about the hardware, the instances, and the general setup.

6.1 Preliminaries

The computational experiments were conducted on a computing cluster with 12 nodes, each featuring 12-core Intel Xeon® E5-2620 v3 2.4 GHz processor, 128 GB of RAM, and running Linux x86_64, CentOS 7.2. All formulations were implemented in C++, compiled with GCC 10.2.0, and

solved using the commercial optimizer Gurobi 9.1.1. Each instance was solved by a single node and given a time limit of 7,200 seconds.

We test our results with multiple weighted and unweighted instances. Here, by unweighted we refer to the oft-called cardinality version of the problem, where the weight of each vertex is equal to one and the partitioning bound r is a number between 3 and $n-1$. Depending on the test being conducted, we use different types of graphs:

- **Random trees:** We generate 5 random trees for each value of $n \in \{20, 40, 60, 80, 100\}$. Each tree is created by first initializing a tree with a single vertex, then for vertices $i = 2, \ldots, n$, we randomly create an edge from vertex i to one of the previous $j-1$ vertices. For each instance, we use three separate values of r, resulting in a total of 75 instances.

- **Random structured graphs:** Motivated by the testbed used in [55], we also consider graphs possessing other types of structural properties. We randomly generate planar grids, toroidal grids, and series-parallel graphs with $n \in [20, 110]$ vertices. We solved each of these instances with $r = 5, \ldots, n-5$. For all other graphs we selected r as 10, 20, and 30 percent of n or $w(V)$. In total, we have 210 of these instances.

- **Random general graphs:** We randomly generate several connected graphs using well-known graph generator models: small-world [78], Erdös-Rényi [25], and Barabási-Albert [4]. For each random graph type, we generate three graphs for each size $n \in \{30, 60, 100\}$ and edge-vertex rations $\frac{m}{n} \in \{2, 4, 6\}$. Since each instance will be solved for three different values of r, we have a total of 243 random weighted instances of this type.

- **Real-life graphs:** We also test our formulations on 27 real instances collected from the Network Data Repository [58] and the Hazmat Network Data [72].

In total, our test-bed consists of 555 instances. For random weighted instances, vertex weights and edge costs were randomly sampled so that each $w_i \sim U\{1, 1000\}$ and $c_e \sim U\{1, 1000\}$, for all $i \in V$ and $e \in E$. Furthermore, we adjust the values of parameter r such that its proportion to $w(V)$ is also 10, 20, and 30 percent.

We measure the performance of our solution approaches mainly through the solution time and the optimality gap. All times reported in this section are measured in seconds and the optimality gap is measured by the percentage $(UB - LB)/LB \times 100$, where UB and LB represent the best upper and lower bounds found by the solver, respectively. Any instance that cannot be solved to optimality within the time limit will have the corresponding runtime of 7,200 seconds

In order to effectively compare the performance of each formulation, we report the shifted geometric mean. We choose this metric (36) over the arithmetic mean, as the latter is known for being biased towards the large run times of difficult instances. Also, the use of the shifted geometric mean to compare algorithms has become a common practice over the years [**?**, 51, 76]. Since several of the easier randomly generated instances have runtimes less than $1.00s$, we select the shift value of $s = 1$. We report the "unscaled" and "scaled" values of the shifted geometric mean. The scaled means in each row are calculated so that the best performing formulation for that group has a scaled mean of one.

$$\gamma_s(t_1, \ldots, t_k) = \left(\prod_{i=1}^{k} \max\{t_i + s, 1\} \right)^{\frac{1}{k}} - s \tag{36}$$

6.2 LP Relaxations comparison on Trees

In this section, we compare the LP relaxation of TRI/FLO/PAT against that of TCF. We also solve both TDP and TDP via Benders (TDP-B) to study their performance. It is important to note that using the dynamic program directly should be faster than solving TDP; however, our aim here is to compare the mathematical programs, so we use those instead. Also, since Section 4 proves that TRI, FLO, and PAT yield the same LP relaxation, we only use TRI for this experiment set.

Rather than using general graphs, we decided to conduct these experiment set on unweighted trees. The reason is twofold: first, as discussed in Section 3, the separation of the minimal tree cover inequalities is NP-hard on general graphs. It is possible to use a mixed-integer program for this purpose; however, the time required to solve such a program may hinder overall the output of the experiments. In general, if one wants to use TCF to solve a problem on other types of graphs, one can then conduct a "lazy" separation (in time polynomial) for integer solutions. However, with such a method it is not possible to obtain the actual LP bound of TCF. Since that is the main objective of this part of the experiments, we decided to use trees. Furthermore, since all the formulations concentrate on partitioning the tree structures of the input graph (see Section 4), we believe that the tree instances can provide a good proxy of the LP strength of these models in general. Second, performing the experiments on trees allows us to compare the solution times of TRI, TCF, and TDP. Since the latter yields optimal integer solutions, one can analyze the computational trade-off between obtaining an integral LP bound and the computational time.

For the TCF formulation, we use a separation algorithm that solves a modified single-commodity flow formulation, often used for finding subtrees of minimum weights. There are dynamic programs available for this purpose on trees, but the formulation is easier to implement and works well in practice. The tree generated depends on the source vertex; thus, we solve this problem for each vertex and add the corresponding minimum tree cover inequality for each violating tree detected.

Table 2: Comparison of the LP relaxation between TRI, TCF, and the cuts generated from the dynamic programming separation algorithm

		TDP	TDP-B		TRI		TCF		
n	r	time	time	cuts	gap	time	gap	time	cuts
20	3	0.00	0.18	21	**7**	0.01	17	0.45	23
20	6	0.01	0.22	17	**29**	0.02	31	0.48	18
20	9	0.01	0.21	11	35	0.02	**0**	0.31	8
40	5	0.01	0.60	119	**22**	0.18	37	2.49	50
40	10	0.02	1.61	115	27	0.23	27	3.41	45
40	15	0.05	1.95	57	32	0.22	**23**	1.89	22
60	6	0.03	1.37	107	**14**	0.78	26	8.33	71
60	12	0.06	6.28	150	25	1.45	**22**	10.22	59
60	18	0.08	12.26	107	25	1.09	**20**	11.49	51
80	10	0.06	9.76	197	**20**	5.35	22	28.54	108
80	16	0.10	40.31	229	23	4.97	23	30.15	78
80	25	0.16	327.74	371	46	6.16	**25**	34.25	73
100	10	0.07	23.65	367	**20**	15.36	28	60.49	147
100	20	0.14	224.25	379	22	17.60	**15**	76.71	111
100	30	0.30	1287	344	36	18.04	**29**	54.34	61

Table 2 compares the average gap and solution time for TRI, TCF, TDP, and TDP-B. For TCF and TDP-B, we also list the average number of cuts required to reach the optimal objective value. Since TDP and TDP-B were developed from the dynamic program, these formulations provide the convex hull for trees. For this reason, we do not list the gap as they produce the optimal

integral solution. We can see that not only does the TDP produce the best relaxation, but it is also significantly faster to solve than the other formulations. Furthermore, we see that TRI and TCF produce comparable relaxations, with TRI being the faster of the two formulations. While the TCF produced the optimal integral solution for some instances, there are other instances where TRI produced a significantly stronger relaxation ($n = 60$). Finally, the cuts generated from TDP-B are effective yet costly to produce as the values of n and r increase even on graphs that are trees. Therefore, it may be beneficial to use this subroutine to generate costs sparingly.

6.3 Dynamic Programming Cuts on General Graphs

In this section, we compare the LP relaxations of the formulations FLO and FLO with the heuristically separated DP inequalities which we refer to as FLO+DP. We are interested to see the effect that these cuts have on the relaxation quality on graphs that are denser than trees. In particular, we will compare these two formulations on unweighted series-parallel graphs.

Given a fractional solution $\bar{\mathbf{y}}$, we first set the weight of each edge $e \in E$ equal to the current edge cut value $\bar{y}_e$ and find a minimum spanning tree T using Prim's algorithm. Then, using formulation (34a)-(34h), we keep separating violated inequalities for T until such formulation is unable to find new violated cuts. Then, we continue by finding new candidate spanning trees, each time replicating this process until the new generated tree fails to yield a violated cut. This speeds up the separation time by reducing the number of times formulation (34a)-(34h) gets instantiated with a new tree T.

Table 3: Comparison of the LP bound of FLO and FLO with the addition of the heuristically separated knapsack cuts on series-parallel graphs.

				FLO		FLO+DP			
n	m	r	z^*	$\bar{z}$	time	$\bar{z}$	time	cuts	Δgap
		5	15	14.08	0.19	14.91	3.54	80	5.93
28	40	10	8	7.94	0.23	8.00	3.22	5	0.70
		15	5	4.57	0.22	5.00	35.32	241	9.37
		5	20	19.00	0.18	19.63	1.52	4	3.37
30	45	10	14	13.84	0.19	14.00	12.24	50	1.16
		15	10	9.80	0.25	10.00	46.67	84	2.04
		5	17	15.83	0.21	16.52	4.14	69	4.49
35	50	10	9	7.42	0.25	7.89	45.65	161	7.23
		15	5	4.31	0.19	5.00	113.56	345	16.12
		5	19	18.34	0.23	18.98	8.72	116	3.47
44	60	10	11	9.87	0.30	10.28	72.39	276	4.48
		15	8	6.43	0.24	7.10	396.80	692	11.75
		5	21	20.50	0.22	20.67	5.23	39	0.83
46	65	10	11	9.46	0.31	10.38	108.79	538	10.25
		15	6	5.20	0.24	5.44	477.28	227	5.00
		5	28	26.85	0.27	27.54	36.31	167	2.60
51	75	10	17	16.09	0.34	16.72	166.35	268	4.00
		15	11	10.94	0.27	11.00	56.94	2	0.57

In Table 3, we report the LP relaxation, denoted by $\bar{z}$, the optimal integer solution, denoted z^*, and the solution time of both methods. Furthermore, for FLO+DP we report the number of cuts generated to reach the optimal LP solution and its improvement measured as the difference between the optimality gaps, which we denote Δgap. From the results we see that for 7 out of 18 instances tested, the FLO+DP formulation reached the optimal solution. We can see that the greatest improvement occurs for larger values of r at the expense of longer computational times.

6.4 Block-Decomposition Inequalities

We now present a comparison of the TCF formulation and an equivalent formulation strengthened with the block-decomposition inequalities. Here, we use integer separation exclusively.

For the TCF formulation, we can separate minimal tree cover inequalities in polynomial time using the procedure described in Theorem 2. For the block-decomposition inequalities, we first separate a minimal tree cover T and randomly add edges to it to generate a larger subgraph. To avoid producing dense blocks, we randomly select only a few edges to add to $E(T)$. Although this procedure only considers relatively sparse subgraphs, our results show that it still provides a significant improvement over the minimal tree cover inequalities.

Table 4 compares the solution time and cuts generated of the two formulations over a set of series-parallel graphs. From these results, it is apparent that the block-decomposition inequalities, even with blocks with small edge connectivity, can significantly strengthen the tree cover inequalities. We see that the block-decomposition strengthened formulation achieves the optimal objective value faster and with much fewer constraints than TCF.

Table 4: Comparison of the TCF and block-decomposition integer programming formulations over series-parallel graphs.

				TCF		TCF+Block	
n	m	r	z^*	time	cuts	time	cuts
		5	15	0.49	785	**0.19**	336
28	40	10	8	2.04	1914	**1.65**	1,178
		15	5	0.19	808	**0.17**	399
		5	20	6.10	2,556	**4.40**	2,225
30	45	10	14	2,937	33,924	**885.05**	21,939
		15	10	**3,288**	65,679	3,812	57,285
		5	17	**0.84**	565	0.96	390
35	50	10	9	**0.79**	698	1.70	787
		15	5	0.19	491	**0.07**	149
		5	19	2.16	661	**2.13**	574
44	60	10	11	3.95	1,476	**2.50**	734
		15	8	3.92	1,462	**0.14**	102
		5	21	2.53	618	**1.98**	341
46	65	10	11	2.20	1,058	**1.90**	448
		15	6	**0.27**	443	0.29	387
		5	28	39.78	2,013	**30.76**	1,333
51	75	10	17	1,767	9,334	**3.29**	430
		15	11	654.10	10,485	**149.19**	1,831

6.5 Comparing Heuristics for Tree Cover Separation

We have discussed several algorithms for separating tree cover and knapsack tree inequalities. In this section, we will compare the performance of the fractional separation routines of a Prim's based heuristic for separating tree cover inequalities and a heuristic for computing more general knapsack tree inequalities when used to enhance the FLO formulation. In particular, we heuristically separate trees with weights approximately 1.25 and 1.50 times r and compute the corresponding inequality using Algorithm 1. We will use Watts-Strogatz graphs in order to compare these algorithms.

An issue that often arises in branch-and-cut implementations is that the overhead required to generate cuts at every node of the branch-and-bound tree yields diminishing returns. To remedy

this, we define a parameter ρ, that dictates the probability that we attempt to separate a violated tree cover or knapsack inequality at a given node.ß In this computational study, we will compare the performance for separation probabilities $\rho \in \{1/2, 1\}$.

Tables 5 and 6 compare the runtime, number of cuts, separation time, and number of branch and bound nodes evaluated for each separation algorithm. When comparing each of the computational times, we also measure the speed-up of each formulation to that of the original FLO formulation. From these results, we can see that the tree cover inequalities with $\rho = 1/2$ produced the greatest speedup over all the enhancements. As a result, we will use this configuration as our formulation FLO+ in the next section. In general, the average time spent separating these inequalities is small compared to the total runtime. Furthermore, the tree cover inequalities separation for each value of ρ evaluated far fewer nodes in the branch-and-bound tree than the general knapsack tree inequalities.

Table 5: Comparison of the shifted geometric mean of the runtime among the FLO formulation and the FLO formulation with each separation algorithm with $\rho = 1/2, 1$

		FLO	r		$1.25r$		$1.5r$	
ρ	n	unscaled	unscaled	speed-up	unscaled	speed-up	unscaled	speed-up
	30	6.47	5.87	1.10	5.98	1.08	6.00	1.08
1	60	247.46	380.48	0.65	412.72	0.60	420.52	0.59
	100	2,694	3,072	0.88	3,199	0.84	3,341	0.81
	30	6.47	4.64	1.39	5.83	1.11	6.00	1.08
1/2	60	247.46	130.41	1.90	404.84	0.61	414.96	0.60
	100	2,694	1,845	1.46	3,255	0.83	3,345	0.81

Table 6: Comparison of the average number of cuts generated, separation time, and branch and bound nodes for each separation algorithm with $\rho = 1/2, 1$

		$w(T)$	r			$1.25r$			$1.5r$		
ρ	n	m	cuts	sep. time	nodes	cuts	sep. time	nodes	cuts	sep. time	nodes
	30	60	19	0.00	150	11	0.00	163	7	0.00	175
	30	90	43	0.01	846	13	0.01	757	10	0.01	837
	30	180	60	0.08	9,180	30	0.11	10,029	14	0.10	9,337
1	60	120	160	0.04	2,509	118	0.09	4,139	77	0.10	4,838
	60	180	243	0.34	17,271	143	0.51	22,237	84	0.60	25,820
	60	360	71	1.04	29,061	21	1.18	29,017	7	1.23	29,649
	100	200	153	0.32	5,841	130	0.54	9,674	100	0.76	13,345
	30	60	19	0.00	198	8	0.00	167	2	0.00	181
	30	90	20	0.00	556	12	0.01	743	12	0.01	1021
	30	180	60	0.05	6,414	22	0.09	9,511	10	0.09	9,555
1/2	60	120	23	0.01	338	81	0.06	3,310	54	0.08	4,319
	60	180	59	0.07	3,825	143	0.49	23,069	60	0.48	22,256
	60	360	75	0.94	28,223	15	1.08	28,782	5	1.12	29,261
	100	200	74	0.15	2,940	107	0.55	10,174	65	0.78	13,682

6.6 Performance Comparison of the Formulations

For each formulation, we compared the computation time and optimality gap, when necessary, among all instances. We average the results from each randomly generated graph with the same number of nodes and graph density. For the planar, toroidal, and series-parallel graphs, we average

the instances with the same number of nodes and edges. For real-life instances, we report the runtimes and gaps directly.

Table 7 and 8 indicates the performance of each formulation for different graph sizes, densities, and structures. We have decided to omit the results for $n = 100$ with $\frac{m}{n} \in \{4, 6\}$ as only a few instances were solved within the time limit. Overall, the formulation FLO+ had the best performance except for the smallest instances (PAT) and thdensestse instances (TRI). On average it appears the formulation FLO+ outperforms TRI by 550% and FLO by 50%. Furthermore, FLO+ performed especially well on small-world graphs. It is worth noting in these experiments there were no additional cuts for tree covers or knapsack trees generated in PAT+, so we omit this formulation for the other instances.

Table 7 and 8 indicate the performance of each formulation for different graph sizes, densities, and structures. We have decided to omit the results for $n = 100$ with $\frac{m}{n} \in \{4, 6\}$, as only a few instances were solved within the time limit. Overall, the formulation FLO+ had the best performance except for the smallest instances, where PAT did better, and the densest instances, where TRI did better. On average it appears the formulation FLO+ outperforms TRI by 550% and FLO by 50%. Furthermore, FLO+ performed especially well on small-world graphs. It is worth noting in these experiments there were no additional cuts for tree covers or knapsack trees generated in PAT+, so we omit this formulation for the other instances.

Table 10 contains the results of each formulation for real unweighted instances. We can see that FLO and PAT are more competitive than the previous instances and the addition of the tree cover inequalities to FLO is less effective than for weighted instances. For instances where FLO and FLO+ had the best performance, it appears that as r increased, the strength of the tree cover inequalities become less effective.

Table 7: Comparison of the geometric mean for solution time among all formulations on random weighted instances with different graph sizes

		TRI		FLO		PAT		FLO+		PAT+	
type	n	unscaled	scaled	unscaled	scaled	unscaled	scaled	unscaled	scaled	unscaled	scaled
watts	30	7.76	2.40	4.94	1.53	3.56	1.10	**3.23**	**1.00**	5.53	1.71
barabasi	30	6.43	2.53	4.34	1.71	**2.54**	**1.00**	2.97	1.17	3.64	1.43
erdos	30	9.77	2.47	7.01	1.77	**3.95**	**1.00**	4.93	1.25	6.25	1.58
all	30	7.85	2.39	5.30	1.61	**3.29**	**1.00**	3.60	1.10	4.99	1.52
watts	60	5,044	42.02	236.45	1.97	580.26	4.83	**120.04**	**1.00**	624.77	5.20
barabasi	60	4,704	12.32	514.61	1.35	873.19	2.29	**381.74**	**1.00**	895.91	2.35
erdos	60	6,995	4.68	2,024	1.35	2,419	1.62	**1,494**	**1.00**	2,360	1.58
all	60	5,479	13.61	617.67	1.53	1,059	2.63	**402.53**	**1.00**	1,087	2.70
all	-	207.38	5.45	57.21	1.50	59.02	1.55	**38.08**	**1.00**	73.68	1.93

7 Conclusions

This paper concerns integer programming formulations for solving graph partitioning problems on sparse graphs that impose an upper limit on the weight of the partition clusters. We develop two extended formulations based on maximum flow interdiction and path connectivity from critical element detection problems. For each extended formulation, we studied the projection onto the space of the edges which revealed that the LP relaxation of the flow and path-based extended formulations are of equal strength to the triangle formulation, yet provide a significant speed up in solution times across almost all instances we test.

Table 8: Comparison of the geometric mean among all formulations on random weighted instances with different graph densities

		TRI		FLO		PAT		FLO+		PAT+	
type	$\frac{m}{n}$	unscaled	scaled	unscaled	scaled	unscaled	scaled	unscaled	scaled	unscaled	scaled
watts	2	386.12	60.83	12.84	2.02	41.49	6.54	**6.35**	**1.00**	44.59	7.02
barabasi	2	302.89	22.56	23.73	1.77	35.46	2.64	**13.42**	**1.00**	37.20	2.77
erdos	2	490.25	6.27	120.89	1.55	89.33	1.14	**78.24**	**1.00**	96.72	1.24
total	2	386.77	20.46	33.42	1.77	51.07	2.70	**18.90**	**1.00**	54.60	2.89
watts	4	197.17	11.64	39.30	2.32	33.03	1.95	**16.94**	**1.00**	47.87	2.83
barabasi	4	226.91	5.56	65.32	1.60	72.59	1.78	**40.84**	**1.00**	88.44	2.17
erdos	4	251.72	2.84	150.27	1.70	132.60	1.50	**88.57**	**1.00**	166.59	1.88
total	4	224.10	5.69	72.95	1.85	68.17	1.73	**39.40**	**1.00**	89.02	2.26
watts	6	439.03	1.27	439.11	1.27	575.74	1.66	**346.15**	**1.00**	788.84	2.28
barabasi	6	**377.63**	**1.00**	494.37	1.31	383.73	1.02	473.63	1.25	509.16	1.35
erdos	6	**737.69**	**1.00**	969.26	1.31	922.80	1.25	1,147	1.55	1,238	1.68
total	6	**498.94**	**1.00**	596.86	1.20	593.32	1.19	574.90	1.15	799.05	1.60
total	2,4,6	355.98	5.77	94.91	1.54	111.54	1.81	**61.66**	**1.00**	134.82	2.19

Table 9: Comparison of the geometric mean of solution time among planar gird, toroidal grid and series-parallel graphs

			TRI		FLO		PAT		FLO+	
type	n	m	unscaled	scaled	unscaled	scaled	unscaled	scaled	unscaled	scaled
	40	80	36.41	14.63	3.29	1.32	3.77	1.52	**2.49**	**1.00**
	50	100	647.85	69.72	14.40	1.55	24.24	2.61	**9.29**	**1.00**
toroidal	60	120	4,586	154.26	47.10	1.58	136.08	4.58	**29.73**	**1.00**
grid	70	140	7,200	63.22	186.76	1.64	1,018	8.94	**113.89**	**1.00**
	80	160	7,200	20.70	522.08	1.50	3,032	8.72	**347.86**	**1.00**
	28	40	1.80	1.63	1.14	1.03	**1.11**	**1.00**	1.15	1.04
	30	45	2.71	2.30	1.20	1.02	**1.18**	**1.00**	1.19	1.01
series	35	50	3.01	2.40	1.29	1.03	**1.25**	**1.00**	1.27	1.01
parallel	44	60	21.62	16.07	1.44	1.07	1.35	1.00	1.35	1.00
	46	65	15.49	11.33	**1.37**	**1.00**	1.45	1.06	1.38	1.01
	51	75	220.71	118.27	2.18	1.17	2.65	1.42	**1.87**	**1.00**
	40	66	26.03	33.19	1.17	1.49	1.06	1.36	**0.78**	**1.00**
	50	85	447.83	153.94	3.72	1.28	4.87	1.68	**2.91**	**1.00**
	60	104	3,031	740.33	5.88	1.43	9.31	2.27	**4.09**	**1.00**
planar	70	123	6,958	777.56	12.02	1.34	59.90	6.69	**8.95**	**1.00**
grid	80	142	7,200	213.23	44.46	1.32	367.09	10.87	**33.77**	**1.00**
	90	161	7,200	95.31	116.35	1.54	1,552	20.54	**75.54**	**1.00**
	100	180	7,200	51.74	**139.15**	**1.00**	1,669	11.99	160.15	1.15
	110	199	7,200	15.96	**451.20**	**1.00**	4,452	9.87	499.58	1.11

Table 10: Solution time and gap for real unweighted instances. $*$ - Denotes the best bound for any instance where the optimal solution was not obtained.

					TRI		FLO		PAT		FLO+	
instance	n	m	r	z^*	time	gap	time	gap	time	gap	time	gap
karate	34	78	3	56	0.23	0	0.12	0	**0.06**	**0**	0.12	0
			7	35	2.49	0	0.84	0	**0.36**	**0**	0.64	0
			10	24	1.87	0	0.30	0	**0.17**	**0**	0.29	0
dolphins	62	159	6	78	1,299	0	**19.63**	**0**	58.61	0	67.60	0
			12	48	7,200	6	16.46	0	25.33	0	**11.47**	**0**
			18	34	7,200	12	41.90	0	**26.81**	**0**	36.67	0
lesmis	77	254	8	110	7,200	9	80.09	0	58.68	0	**33.92**	**0**
			16	56	7,200	75	5.60	0	9.21	0	**5.43**	**0**
			24	44	7,200	28	**14.67**	**0**	39.09	0	22.50	0
albany	90	149	10	8.51	7,200	57	21.29	0	23.53	0	**16.17**	**0**
			20	4.57	7,200	8	26.39	0	27.13	0	**16.17**	**0**
			30	2.92	7,200	22	**19.38**	**0**	32.57	0	56.81	0
buffalo	90	149	10	47.37	7,200	78	56.26	0	59.34	0	**40.68**	**0**
			20	26.85	7,200	89	25.09	0	57.06	0	**23.43**	**0**
			30	19.8	7,200	52	**57.39**	**0**	121.81	0	64.91	0
polbooks	105	441	10	216*	7,200	21	**7,200**	**1**	7,200	4	7,200	3
			21	130	7,200	24	7,154	0	**2,187**	**0**	7,200	1
			31	84	7,200	41	5,849	0	**1,899**	**0**	5,643	0
adjnoun	112	425	11	259*	7,200	27	**7,200**	**12**	7,200	18	7,200	15
			22	252*	7,200	36	7,200	31	7,200	47	7,200	31
			33	212*	7,200	39	7,200	31	7,200	64	7,200	30
football	115	613	11	214*	7,200	14	7,200	8	**7,200**	**5**	7,200	8
			23	148*	7,200	22	**7,200**	**7**	7,200	13	7,200	38
			35	117*	7,200	25	**7,200**	**9**	7,200	87	7,200	35
powerbus	494	586	46	51*	-	-	**7,200**	**29**	7,200	94	7,200	58
			92	287*	-	-	7,200	95	7,200	97	7,200	93
			139	20*	-	-	**7,200**	**20**	7,200	98	7,200	24

Since each formulation produces inequalities that aim to partition the trees of the graph, yet none of them project into the convex hull on trees, we developed a dynamic programming algorithm that provides a recursive structure that can be used to built a linear programming formulation that is integral for trees. With this LP, we also developed a cut generation subroutine to generate strong cuts for general graphs. As part of the technical developments of the paper, we also expanded the polyhedral characterization of the problem's solution space, introducing new families of inequalities and provide empirical evidence of their efficacy to improve the quality of the proposed formulations.

We provided an extensive computational study testing the following: (1) the LP bound of all the formulations; (2) the impact of the tree dynamic programming inequalities on the LP bounds; (3) the improvement given by the block-decomposition inequalities; (4) the effect of several heuristics for generating tree cover inequalities on the flow-based formulation; and (5) a comparison of each formulation on randomly generated, grid, and real graphs. We show that the flow-based formulation which includes the heuristically separated minimal tree cover inequalities provides the best performance overall on weighted graphs. For unweighted graphs, this formulation becomes less effective as the value of r increases.

References

[1] A. Abusorrah, A. Alabdulwahab, Z. Li, and M. Shahidehpour. Minimax-regret robust defensive strategy against false data injection attacks. *IEEE Transactions on Smart Grid*, 10(2):2068–2079, 2017.

[2] E. Agasi, R. I. Becker, and Y. Perl. A shifting algorithm for constrained min-max partition on trees. *Discrete Applied Mathematics*, 45(1):1–28, 1993.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows*. Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[4] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[5] Z. Ales and A. Knippel. The k-partitioning problem: Formulations and branch-and-cut. *Networks*, 2020.

[6] K. Andreev and H. Racke. Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006.

[7] R. Aringhieri, A. Grosso, P. Hosteins, and R. Scatamacchia. Polynomial and pseudo-polynomial time algorithms for different classes of the distance critical node problem. *Discrete Applied Mathematics*, 253:103–121, 2019.

[8] A. Arulselvan, C. W. Commander, L. Elefteriadou, and P. M. Pardalos. Detecting critical nodes in sparse graphs. *Computers and Operations Research*, 36(7):2193–2200, 2009.

[9] E. Balas and S. M. Ng. On the set covering polytope: I. all the facets with coefficients in {0, 1, 2}. *Mathematical Programming*, 43(1-3):57–69, 1989.

[10] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear programming and network flows*. John Wiley & Sons, 2008.

[11] P. Bertolazzi, M. Lucertini, and A. M. Spaccamela. Analysis of a class of graph partitioning problems. *RAIRO. Informatique théorique*, 16(3):255–261, 1982.

[12] M. Buchin and L. Selbach. Partitioning algorithms for weighted cactus graphs. *arXiv preprint arXiv:2001.00204*, 2020.

[13] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. Recent advances in graph partitioning. In *Algorithm Engineering*, pages 117–158. Springer, 2016.

[14] R. D. Carr and G. Lancia. Compact vs. exponential-size lp relaxations. *Operations Research Letters*, 30(1):57 – 65, 2002.

[15] J. Chen and B. Yuan. Detecting functional modules in the yeast protein–protein interaction network. *Bioinformatics*, 22(18):2283–2290, 2006.

[16] S. Chopra. The graph partitioning polytope on series-parallel and 4-wheel free graphs. *SIAM journal on discrete mathematics*, 7(1):16–31, 1994.

[17] S. Chopra and M. R. Rao. The partition problem. *Mathematical Programming*, 59(1-3):87–115, 1993.

[18] V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4(4):305 – 337, 1973.

[19] V. Chvátal and W. Cook. The discipline number of a graph. *Discrete mathematics*, 86(1-3):191–198, 1990.

[20] M. Conforti, G. Cornuéjols, and G. Zambelli. Extended formulations in combinatorial optimization. *4OR*, 8(1):1–48, 2010.

[21] M. Conforti, M. Rao, and A. Sassano. The equipartition polytope. I: formulations, dimension and basic facets. *Mathematical Programming*, 49(1-3):49–70, 1990.

[22] S. J. D'Amico, S.-J. Wang, R. Batta, and C. M. Rump. A simulated annealing approach to police district design. *Computers & Operations Research*, 29(6):667–684, 2002.

[23] N. Deo. *Graph theory with applications to engineering and computer science*. Courier Dover Publications, 2017.

[24] M. Di Summa, A. Grosso, and M. Locatelli. Branch and cut algorithms for detecting critical nodes in undirected graphs. *Computational Optimization and Applications*, 53(3):649–680, 2012.

[25] P. Erdős and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[26] U. Faigle, R. Schrader, and R. Suletzki. A cutting plane algorithm for optimal graph partitioning. *Methods of operations research*, 57:109–116, 1987.

[27] C. E. Ferreira, A. Martin, C. C. de Souza, R. Weismantel, and L. A. Wolsey. Formulations and valid inequalities for the node capacitated graph partitioning problem. *Mathematical Programming*, 74(3):247–266, 1996.

[28] C. E. Ferreira, A. Martin, C. C. de Souza, R. Weismantel, and L. A. Wolsey. The node capacitated graph partitioning problem: a computational study. *Mathematical programming*, 81(2):229–256, 1998.

[29] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[30] A. Frangioni, A. Lodi, and G. Rinaldi. Optimizing over semimetric polytopes. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 431–443. Springer, 2004.

[31] A. Frangioni, A. Lodi, and G. Rinaldi. New approaches for optimizing over the semimetric polytope. *Mathematical programming*, 104(2-3):375–388, 2005.

[32] R. E. Gomory. An algorithm for integer solutions to linear programs. *Recent advances in mathematical programming*, 64(260-302):14, 1963.

[33] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[34] M. Grötschel and Y. Wakabayashi. A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 45(1-3):59–96, 1989.

[35] M. Grötschel and Y. Wakabayashi. Facets of the clique partitioning polytope. *Mathematical Programming*, 47(1-3):367–387, 1990.

[36] C. Hojny, I. Joormann, H. Lüthen, and M. Schmidt. Mixed-integer programming techniques for the connected max-k-cut problem. *Mathematical Programming Computation*, 13(1):75–132, 2021.

[37] J. Hopcroft and R. Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, 16:372–378, 1973.

[38] E. Israeli and R. K. Wood. Shortest-path network interdiction. *Networks*, 40(2):97–111, 2002.

[39] T. Ito, T. Nishizeki, M. Schröder, T. Uno, and X. Zhou. Partitioning a weighted tree into subtrees with weights in a given range. *Algorithmica*, 62(3):823–841, 2012.

[40] E. L. Johnson, A. Mehrotra, and G. L. Nemhauser. Min-cut clustering. *Mathematical programming*, 62(1-3):133–151, 1993.

[41] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu. *VLSI physical design: from graph partitioning to timing closure*. Springer Science & Business Media, 2011.

[42] S. Kundu and J. Misra. A linear tree partitioning algorithm. *SIAM Journal on Computing*, 6(1):151–154, 1977.

[43] M. Labbé and F. A. Özsoy. Size-constrained graph partitioning polytopes. *Discrete mathematics*, 310(24):3473–3493, 2010.

[44] M. Laurent, M. Deza, and M. Grötschel. Complete descriptions of small multicut polytopes. In *Applied geometry and discrete mathematics: The Victor Klee Festschrift*, pages 221–252. American Mathematical Society, 1991.

[45] X. Liu, Z. Li, and Z. Li. Optimal protection strategy against false data injection attacks in power systems. *IEEE Transactions on Smart Grid*, 8(4):1802–1810, 2016.

[46] D. Lozovanu and A. Zelikovsky. Minimal and bounded tree problems. *Tezele Congresului XVIII al Academiei Romano-Americane, Kishniev*, page 25, 1993.

[47] J. A. Lukes. Efficient algorithm for the partitioning of trees. *IBM Journal of Research and Development*, 18(3):217–224, 1974.

[48] F. Mahdavi Pajouh, V. Boginski, and E. L. Pasiliao. Minimum vertex blocker clique problem. *Networks*, 64(1):48–64, 2014.

[49] F. Mahdavi Pajouh, J. L. Walteros, V. Boginski, and E. L. Pasiliao. Minimum edge blocker dominating set problem. *European Journal of Operational Research*, 247(1):16–26, 2015.

[50] A. Mehrotra and M. A. Trick. Cliques and clustering: A combinatorial approach. *Operations Research Letters*, 22(1):1–12, 1998.

[51] H. Mittelmann. Benchmarks for optimization software, November 2018. `http://plato.asu.edu/bench.html`. Last accessed: May, 2021.

[52] Y.-S. Myung and H.-J. Kim. A cutting plane algorithm for computing k-edge survivability of a network. *European Journal of Operational Research*, 156(3):579–589, 2004.

[53] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY, USA, 1988.

[54] M. E. Newman. Community detection and graph partitioning. *EPL (Europhysics Letters)*, 103(2):28003, 2013.

[55] D. P. Nguyen, M. Minoux, V. H. Nguyen, T. H. Nguyen, and R. Sirdey. Improved compact formulations for a wide class of graph partitioning problems in sparse graphs. *Discrete Optimization*, 25:175–188, 2017.

[56] M. Oosten, J. H. Rutten, and F. C. Spieksma. The clique partitioning problem: facets and patching facets. *Networks: An International Journal*, 38(4):209–226, 2001.

[57] M. Oosten, J. H. G. C. Rutten, and F. C. R. Spieksma. Disconnecting graphs by removing vertices: A polyhedral approach. *Statistica Neerlandica*, 61(1):35–60, 2007.

[58] R. A. Rossi and N. K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015.

[59] H. Salemi and A. Buchanan. Solving the distance-based critical node problem. 2021. http://www.optimization-online.org/DB_HTML/2020/04/7751.html.

[60] S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27 – 64, 2007.

[61] S. Shen and J. C. Smith. Polynomial-time algorithms for solving a class of critical node problems on trees and series-parallel graphs. *Networks*, 60(2):103–119, 2012.

[62] S. Shen, J. C. Smith, and R. Goli. Exact interdiction models and algorithms for disconnecting networks via node deletions. *Discrete Optimization*, 9(3):172 – 188, 2012.

[63] H. D. Sherali and B. J. Lunday. On generating maximal nondominated benders cuts. *Annals of Operations Research*, 210(1):57–72, 2013.

[64] J. Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

[65] J. C. Smith and Y. Song. A survey of network interdiction models and algorithms. *European Journal of Operational Research*, 283(3):797–811, 2020.

[66] M. M. Sørensen. *A Polyhedral Approach to Graph Partitioning*. PhD thesis, Aarhus School of Business, 1995.

[67] M. M. Sørensen. *b*-tree facets for the simple graph partitioning polytope. *Journal of Combinatorial Optimization*, 8(2):151–170, 2004.

[68] M. M. Sørensen. Facet-defining inequalities for the simple graph partitioning polytope. *Discrete Optimization*, 4(2):221–231, 2007.

[69] M. M. Sørensen. Facets for node-capacitated multicut polytopes from path-block cycles with two common nodes. *Discrete Optimization*, 25:120–140, 2017.

[70] M. M. Sørensen et al. Polyhedral computations for the simple graph partitioning problem. *Aarhus School of Business, Department of Accounting, Finance and Logistics*, page 7, 2005.

[71] M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM*, 44(4):585–591, 1997.

[72] STOM-Group. Hazmat network data, 2021. data retrieved from, https://github.com/STOM-Group/Hazmat-Network-Data. Last accessed: June 2021.

[73] H. Validi and A. Buchanan. Political districting to minimize cut edges. 2021. http://www.optimization-online.org/DB_HTML/2021/04/8349.html.

[74] H. Validi, A. Buchanan, and E. Lykhovyd. Imposing contiguity constraints in political districting models. *To appear in Operations Research*, 2020.

[75] C. Vogiatzis and J. L. Walteros. Integer programming models for detecting graph bipartitions with structural requirements. *Networks*, 71(4):432–450, 2018.

[76] J. L. Walteros and A. Buchanan. Why is maximum clique often easy in practice? *Operations Research*, 68(6):1866–1895, 2020.

[77] J. L. Walteros, A. Veremyev, P. M. Pardalos, and E. L. Pasiliao. Detecting critical node structures on graphs: A mathematical programming approach. *Networks*, 73(1):48–88, 2019.

[78] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *nature*, 393(6684):440–442, 1998.

[79] N. Wei, J. L. Walteros, and F. M. Pajouh. Integer programming formulations for minimum spanning tree interdiction. *INFORMS Journal on Computing*, 2021.