European Journal of Operational Research xxx (xxxx) xxx

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Partitioning a graph into balanced connected classes: Formulations, separation and experiments

Flávio K. Miyazawa [a], Phablo F. S. Moura [b,*], Matheus J. Ota [a], Yoshiko Wakabayashi [c]

[a] Instituto de Computação, Universidade Estadual de Campinas, Brazil
[b] Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Brazil
[c] Instituto de Matemática e Estatística, Universidade de São Paulo, Brazil

ARTICLE INFO

Article history:
Received 18 July 2020
Accepted 29 December 2020
Available online xxx

Keywords:
Integer programming
Branch-and-cut
Separation algorithm
Balanced partition
Connected partition

ABSTRACT

This work addresses the *balanced connected k-partition problem* (BCP_k), which is formally defined as follows. Given a connected graph $G = (V, E)$ with nonnegative weights on the vertices, find a partition $\{V_i\}_{i=1}^k$ of V such that each class V_i induces a connected subgraph of G, and the weight of a class with the minimum weight is as large as possible. This problem, known to be NP-hard, has been largely investigated under different approaches and perspectives: exact algorithms, approximation algorithms for some values of k or special classes of graphs, and inapproximability results. On the practical side, BCP_k is used to model many applications arising in image processing, cluster analysis, operating systems and robotics. We propose three linear programming formulations for BCP_k. The first one contains only binary variables and a potentially large number of constraints that can be separated in polynomial time in the corresponding linear relaxation. We introduce new valid inequalities and design polynomial-time separation algorithms for them. The other two formulations are based on flows and have a polynomial number of constraints and variables. Our computational experiments show that the exact algorithms based on the proposed formulations outperform the other exact approaches presented in the literature.

© 2021 Elsevier B.V. All rights reserved.

1. Introduction

We adopt the standard notation for (di)graphs. For a (di)graph G, $V(G)$ denotes its vertex set; and $E(G)$ (resp. $A(G)$) denotes its edge set (resp. arc set). To simplify notation, we assume that the input graph G has vertex set V and edge set E; moreover, unless otherwise stated, $n = |V|$ and $m = |E|$. For an integer k, the symbol $[k]$ denotes the set $\{1, 2, \ldots, k\}$ if $k \geq 1$, and the empty set, otherwise. A *k-partition* of G is a collection $\{V_i\}_{i\in[k]}$ of nonempty subsets of V such that $\bigcup_{i=1}^k V_i = V$, and $V_i \cap V_j = \emptyset$ for all $i, j \in [k]$, $i \neq j$. We refer to each set V_i as a *class* of the partition. We say that a k-partition $\{V_i\}_{i\in[k]}$ of G is *connected* if $G[V_i]$, the subgraph of G induced by V_i, is connected for each $i \in [k]$.

Let $w : V \to \mathbb{Q}_{\geq}$ be a function that assigns nonnegative weights to the vertices of G. For every subset $V' \subseteq V$, we define $w(V') = \sum_{v\in V'} w(v)$. For simplicity, if H is a subgraph of G, instead of $w(V(H))$, we write $w(H)$. For a set $S \subseteq \mathbb{R}^\ell$, we denote by convS, the convex hull of S.

We are now ready to define the *balanced connected k-partition problem* (BCP_k), where k is a fixed positive integer. As we mention in what follows, studies on this problem, firstly on trees (Perl & Schach, 1981), trace back to 1981.

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

There are several applications in logistics, image processing, data base, operating systems, cluster analysis, education, robotics and metabolic networks that can be modeled as a balanced connected partition problem (Becker & Perl, 1983; Lucertini, Perl, & Simeone, 1989; 1993; Maravalle, Simeone, & Naldini, 1997; Matić & Božić, 2012; Matić & Grbić, 2020; Zhou, Wang, Ding, Hu, & Shang, 2019). These different real-world applications indicate the importance of designing algorithms for BCP_k, and reporting on the computational experiments with their implementations. Not less important are the theoretical studies of the rich and diverse mathematical formulations.

Problems on partitioning a vertex-weighted graph into a fixed number of connected subgraphs with similar weights have been

* Corresponding author.
E-mail addresses: fkm@ic.unicamp.br (F.K. Miyazawa), phablo@dcc.ufmg.br (P.F.S. Moura), matheus.ota@students.ic.unicamp.br (M.J. Ota), yw@ime.usp.br (Y. Wakabayashi).

https://doi.org/10.1016/j.ejor.2020.12.059
0377-2217/© 2021 Elsevier B.V. All rights reserved.

Please cite this article as: F.K. Miyazawa, P.F.S. Moura, M.J. Ota et al., Partitioning a graph into balanced connected classes: Formulations, separation and experiments, European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2020.12.059

largely investigated in the literature since the early eighties. Such partitions are generally called balanced, and a number of different functions have been considered to measure this feature. The balanced connected k-partition problem (BCP_k) is one of these problems. It is closely related to another one, referred to as MIN-MAX BCP_k, defined as follows. Given a pair (G, w), as in BCP_k, find a connected k-partition $\{V_i\}_{i\in[k]}$ of G that minimizes $\max_{i\in[k]}\{w(V_i)\}$.

When $k = 2$, for any instance, an optimal 2-partition for BCP_k is also an optimal solution for MIN-MAX BCP_k; but it is easy to see that when $k > 2$ the corresponding optimal k-partitions may differ (see Lucertini, Perl, & Simeone (1993)). Another possible objective function is to minimize the maximum difference of weights between the classes. All of them can be treated with the formulations we propose here (by changing only the objective function).

1.1. Some known results

The *unweighted* BCP_k (to be denoted by 1-BCP_k) is the restricted version of BCP_k in which all vertices have unit weight. This restricted problem is NP-hard on bipartite graphs for every fixed $k \geq 2$, as proven by Dyer and Frieze (1985). Chlebíková (1996) showed that 1-BCP_2 is NP-hard to approximate within an absolute error guarantee of $n^{1-\varepsilon}$, for all $\varepsilon > 0$. For the weighted case, Becker, Lari, Lucertini, and Simeone (1998) proved that BCP_2 is already NP-hard on (nontrivial) grid graphs. Chataigner, Salgado, and Wakabayashi (2007) showed that, for each $k \geq 2$, BCP_k is NP-hard in the strong sense, even on k-connected graphs, and therefore does not admit a FPTAS, unless P = NP. Wu (2012) observed that BCP_k is NP-hard on interval graphs for any fixed $k \geq 2$.

Chlebíková (1996) designed a 4/3-approximation algorithm for BCP_2. Recently, Chen et al. (2020) presented a 5/3-approximation for BCP_3 and a 3/2-approximation for MIN-MAX BCP_3 on arbitrary graphs. Approximation algorithms for BCP_4 on 4-connected graphs and for 1-BCP_k on special classes of graphs have also appeared in the literature.

Wu (2012) designed a fully polynomial-time approximation scheme (FPTAS) for BCP_2 on interval graphs. When k is part of the input, Borndörfer, Elijazyfer, and Schwartz (2019) designed Δ-approximation algorithms for both MAX-MIN and MIN-MAX versions of the balanced connected partition problem, where Δ is the maximum degree of an arbitrary spanning tree of the input graph G. Specifically for the MAX-MIN version, their Δ-approximation only holds for instances in which the largest weight is at most $w(G)/(\Delta\, k)$. For this case (k not fixed), Chataigner et al. (2007) proved that BCP_k cannot be approximated within a ratio better than 6/5.

Both BCP_k and MIN-MAX BCP_k can be solved in linear time on trees as shown by Perl and Schach (1981), Becker, Schach, and Perl (1982) and Frederickson (1991). One may easily check that 1-BCP_2 on 2-connected graphs can be solved in polynomial time. This problem also admits polynomial-time algorithms on graphs such that each block has at most two articulation vertices Alimonti and Calamoneri (1999); Chlebíková (1996). In special, when the input graph is k-connected, polynomial-time algorithms and other interesting structural results have been obtained for BCP_k by Ma and Ma (1994), Györi (1978), and Lovász (1977). Many other results on the mentioned problems and variants have appeared in the literature Andersson, Gudmundsson, Levcopoulos, and Narasimhan (2002); Nakano, Rahman, and Nishizeki (1997); Soltan, Yannakakis, and Zussman (2020).

Mixed integer linear programming formulations for BCP_2 were proposed by Matić (2014) and for MIN-MAX BCP_k by Zhou et al. (2019). Matić also presented a VNS-based heuristic for BCP_2, and Zhou et al. devised a genetic algorithm for MIN-MAX BCP_k. Both works reported on the computational results obtained with the proposed formulations and heuristics. Such results indicate that the solving method due to Zhou et al. outperforms the one showed by Matić, and, to the best of our knowledge, it is the fastest exact algorithm for BCP_k described in the literature.

1.2. Contributions

In this work, we advance the state of the art on exact algorithms for BCP_k. In Section 2, we introduce a cut-based integer linear programming (ILP) formulation which models BCP_k, and show two strong valid inequalities for this formulation. Polynomial-time separation routines for some of the proposed valid inequalities are discussed in Section 3. These separation routines are implemented in a branch-and-cut algorithm we have designed.

In Section 4, we present a flow and a multicommodity flow based formulations for BCP_k. Both formulations are compact, that is, they have a polynomial number (on the size of the input graph) of variables and constraints. All proposed formulations for BCP_k can be used to model MIN-MAX BCP_k (and some other variants) just by slightly changing the objective function. We discuss the details of the implementations in Section 5, and describe the set of test instances in Section 6. Lastly, we report on computational experiments in Section 7. Our computational results show that the exact algorithms based on the proposed formulations outperform the previous exact methods designed by Matić (2014), and Zhou et al. (2019). Particularly, we are able to solve instances of size over 400 times larger than the size of the largest instances solved by the exact algorithms described in the literature. Moreover, our algorithms are on average 5 times faster than the previously known solving methods.

A short version of this work Miyazawa, Moura, Ota, and Wakabayashi (2020) containing preliminary experimental results was accepted for publication in the proceedings of the International Symposium on Combinatorial Optimization (ISCO 2020). This paper contains an additional formulation, more details on the implementations, and further computational experiments, including experiments on real-world instances.

2. Cut-based formulation

In this section, we consider that (G, w) is an input for BCP_k. The ILP formulation we propose for BCP_k, called $\mathcal{C}_k(G, w)$, or simply $\mathcal{C}$, is based on the following central concept. Let u and v be two non-adjacent vertices in a graph G. We say that a set $S \subseteq V \setminus \{u, v\}$ is a (u, v)-*separator* if u and v belong to different components of $G - S$. We denote by $\Gamma(u, v)$ the collection of all minimal (u, v)-separators in G. In the formulation, we use a binary variable $x_{v,i}$, for every $v \in V$ and $i \in [k]$, that is set to one if and only if v belongs to the i-th class.

$$\max \sum_{v\in V} w(v)\, x_{v,1}$$

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

$$\sum_{i\in[k]} x_{v,i} \leq 1 \qquad \forall v \in V, \tag{2}$$

$$x_{u,i} + x_{v,i} - \sum_{z\in S} x_{z,i} \leq 1 \qquad \forall uv \notin E,\ S \in \Gamma(u, v),\ i \in [k], \tag{3}$$

$$x_{v,i} \in \{0, 1\} \qquad \forall v \in V \text{ and } i \in [k]. \tag{4}$$

Inequalities (1) impose a non-decreasing weight ordering of the classes. Inequalities (2) require that every vertex is assigned to at

most one class. Inequalities (3) guarantee that every class induces a connected subgraph.

In Section 3, we show that the separation problem for inequalities (3) can be solved in polynomial time. Thus, in view of the equivalence of separation and optimization problems Grötschel, Lovász, and Schrijver (2012), the linear relaxation of $\mathcal{C}$ can be solved in polynomial time.

Because of (2), feasible solutions of formulation $\mathcal{C}_k(G,w)$ may have vertices not assigned to any of the k nonempty classes. To deal with this, we introduce the following concept. We say that a collection $\mathcal{V}=\{V_i\}_{i=1}^k$ is a *connected k-subpartition* of G, if it is a connected k-partition of a subgraph (not necessarily proper) of G, and additionally, $w(V_i)\le w(V_{i+1})$ for all $i\in[k-1]$.

Note that one can extend any optimal connected k-subpartition to a connected k-partition with the same objective value. This can be done by greedily putting unassigned vertices into the same class of one of its neighbors that belongs to a nonempty class (assign first those which are at distance 1 and repeat the process).

For any connected k-subpartition $\mathcal{V}$, we denote by $\xi(\mathcal{V})\in\mathbb{B}^{nk}$ the binary vector such that its non-null entries are precisely $\xi(\mathcal{V})_{v,i}=1$ for all $i\in[k]$ and $v\in V_i$ (that is, $\xi(\mathcal{V})$ denotes the incidence vector of $\mathcal{V}$). We now define the polytope associated with $\mathcal{C}_k(G,w)$ as

$$\mathcal{P}_k(G,w)=\text{conv}\{\mathrm{x}\in\mathbb{B}^{\mathrm{nk}}:\mathrm{x}\text{ satisfies inequalities }(1)-(3)\text{ of }\mathcal{C}_k(G,w)\}.$$

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

Proposition 1.

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

Proof. Consider first an extreme point $x\in\mathcal{P}_k(G,w)$. For each $i\in[k]$, we define the set of vertices $U_i=\{v\in V:x_{v,i}=1\}$. It follows from inequalities (1) and (2) that $\mathcal{U}:=\{U_i\}_{i=1}^k$ is a k-partition of a subgraph of G such that $w(U_i)\le w(U_{i+1})$ for all $i\in[k-1]$. To prove that $\mathcal{U}$ is a connected k-subpartition, we suppose to the contrary that there exists $i\in[k]$ such that $G[U_i]$ is not connected. Hence, there exist vertices u and v belonging to two different components of $G[U_i]$. In this case, there is a minimal (u,v)-separator S such that $S\cap U_i=\emptyset$. Thus, vector x violates inequalities (3), a contradiction.

To show the converse, consider now a connected k-subpartition $\mathcal{V}=\{V_i\}_{i=1}^k$ of G. Clearly, $\xi(\mathcal{V})$ satisfies inequalities (1) and (2). Take a fixed $i\in[k]$. For every pair u,v of non-adjacent vertices in V_i, and every (u,v)-separator S in G, it holds that $S\cap V_i\neq\emptyset$, because $G[V_i]$ is connected. Therefore, $\xi(\mathcal{V})$ satisfies inequalities (3). □

The following inequalities dominate inequalities (3) of $\mathcal{C}(G,w)$.

Proposition 2. *Let u and v be non-adjacent vertices of G, let S be a minimal (u,v)-separator, and let $i\in[k]$. Let $Z=\{z\in S:w(P_z)>w(G)/(k-i+1)\}$, where P_z is a minimum-weight (u,v)-path in G that contains z. The following inequality is valid for $\mathcal{P}_k(G,w)$:*

$$x_{u,i}+x_{v,i}-\sum_{s\in S\setminus Z}x_{s,i}\le 1. \tag{5}$$

Proof. Consider an extreme point x of $\mathcal{P}_k(G,w)$, and define $V_j=\{v\in V:x_{v,j}=1\}$ for each $j\in[k]\setminus[i-1]$. Since x satisfies inequalities (1), it holds that

$$(k-i+1)\,w(V_i)\le\sum_{j\in[k]\setminus[i-1]}w(V_j)\le w(G).$$

Thus, $w(G)/(k-i+1)$ is an upper bound for $w(V_i)$. Hence, if u and v belong to V_i, then there exists a vertex $s\in S\setminus Z$ such that s also belongs to V_i. Therefore, x satisfies inequality (5). □

The next class of inequalities was inspired by a result proposed by De Aragão and Uchoa (1999) for a connected assignment problem.

Proposition 3. *Let $q\ge 2$ be a fixed integer, and let S be a subset of vertices of G containing q distinct pairs of vertices $\{s_i,t_i\}$, $i\in[q]$, all mutually disjoint. Let $N(S)$ be the set of neighbors of S in $V\setminus S$. Moreover, let $\sigma:[q]\to[k]$ be an injective function, and let $I=\{\sigma(i):i\in[q]\}$. If there is no collection of q vertex-disjoint (s_i,t_i)-paths in $G[S]$, then the following inequality is valid for $\mathcal{P}_k(G,w)$:*

$$\sum_{i\in[q]}\left(x_{s_i,\sigma(i)}+x_{t_i,\sigma(i)}\right)+\sum_{v\in N(S)}\sum_{i\in[k]\setminus I}x_{v,i}\le 2q+|N(S)|-1. \tag{6}$$

Proof. Suppose, to the contrary, that there exists an extreme point x of $\mathcal{P}_k(G,w)$ that violates inequality (6). Let $A=\sum_{i\in[q]}\left(x_{s_i,\sigma(i)}+x_{t_i,\sigma(i)}\right)$ and $B=\sum_{v\in N(S)}\sum_{i\in[k]\setminus I}x_{v,i}$. From inequalities (2), we have that $A\le 2q$. Since x violates (6), it follows that $B>|N(S)|-1$. Thus, since x satisfies inequalities (2), it follows that $B=|N(S)|$. Hence, every vertex in $N(S)$ belongs to a class that is different from those indexed by I. This implies that every class indexed by I contains precisely one of the q distinct pairs $\{s_i,t_i\}$. Therefore, there exists a collection of q vertex-disjoint (s_i,t_i)-paths in $G[S]$, a contradiction. □

Kawarabayashi, Kobayashi, and Reed (2012) proved that, given an n-vertex graph G and a set of q pairs of terminals in G, the problem of deciding whether G contains q vertex-disjoint paths linking the given pairs of terminals can be solved in time $\mathcal{O}(n^2)$, for a fixed value of q. Hence, inequalities (6) can be separated in polynomial time when $S=V$.

3. Separation algorithms

We implemented a branch-and-cut algorithm to solve the cut-based formulation that we introduced in Section 2. In this section, we describe the separation routines for inequalities (5) and for a subclass of inequalities (6) that are embedded in this algorithm. In Section 7, we report on the computational results obtained with our implementation.

3.1. Connectivity inequalities

We focus first on the class of inequalities (3) of Section 2, henceforth called *connectivity inequalities*. We address here its corresponding separation problem: given a vector $\widetilde{x}\in\mathbb{R}^{nk}$, find connectivity inequalities that are violated by $\widetilde{x}$ or prove that this vector satisfies all such inequalities.

To tackle this problem, given the input graph $G=(V,E)$, for each $i\in[k]$, we define a digraph D_i with capacities $c_i:A(D_i)\to\mathbb{Q}_{\ge}\cup\{\infty\}$ assigned to its arcs, in the following manner. We set $V(D_i)=\{v_1,v_2:v\in V\}$ and $A(D_i)=A_1\cup A_2$, where $A_1=\{(u_2,v_1),(v_2,u_1):\{u,v\}\in E\}$ and $A_2=\{(v_1,v_2):v\in V\}$. We define $c_i(a)=\widetilde{x}_{v,i}$ if $a=(v_1,v_2)\in A_2$; and $c_i(a)=\infty$, otherwise. Note that each arc in D_i with a finite capacity is associated with a vertex of G. Now, for every pair of non-adjacent vertices $u,v\in V$ such that $\widetilde{x}_{u,i}+\widetilde{x}_{v,i}>1$, we find in D_i a minimum (u_1,v_2)-separating cut. If the weight of such a cut is smaller than $\widetilde{x}_{u,i}+\widetilde{x}_{v,i}-1$, then it is finite and the vertices of G associated with the arcs in this cut defines a (u,v)-separator S in G that violates the connectivity inequality $\widetilde{x}_{u,i}+\widetilde{x}_{v,i}-\sum_{z\in S}\widetilde{x}_{z,i}\le 1$.

Given a (u,v)-separator S, let H_u (resp. H_v) be the connected component of $G-S$ containing u (resp. v). We now describe a procedure to perform a lifting of the connectivity inequalities by removing iteratively unnecessary vertices from S. First, we remove

every vertex z from S such that the neighborhood of z does not intersect with H_u and H_v. Since removing a vertex from S changes the components of $G - S$, we use a Union-Find data structure to update the components. Then, we use Dijkstra's algorithm to remove from S the set Z, as defined in Proposition 2.

The time complexity to separate the connectivity inequalities depends on the algorithm used to find a minimum cut. We use Goldberg's preflow algorithm for maximum flow Goldberg and Tarjan (1988), whose time complexity is $\mathcal{O}(\tilde{n}^2\sqrt{\tilde{m}})$, for a digraph with $\tilde{n}$ vertices and $\tilde{m}$ arcs. Thus, in the worst-case, checking for every $i \in [k]$, and candidate pairs u, v in D_i, the time complexity of this separation algorithm is $\mathcal{O}(kn^4\sqrt{n+m})$.

Despite the high time complexity, we noted in the computational experiments that the number of fractional vertices is relatively small. Thus, we perform arc contractions on all arcs of D_i such that both of its endpoints correspond to vertices associated with variables of integer value. More precisely, an arc $(u_2, v_1) \in A_1$ is contracted if $\widetilde{x}_{u,i} = \widetilde{x}_{v,i} = 1$, and an arc $(u_1, u_2) \in A_2$ is contracted if $\widetilde{x}_{u,i} = 1$. After such contractions, the resulting graphs usually have a small number of vertices and arcs, and so the proposed separation algorithm runs very quickly in practice.

3.2. Cross inequalities

Now we turn to the separation of inequalities (6) on planar graphs $G = (V, E)$, restricted to the case $S = V$. Consider a plane embedding of G, and let F be the boundary of a face with at least 4 vertices and with no repeated vertices. Take four different vertices in F, say s_1, s_2, t_1, t_2, in clockwise order. Since G is planar, it does not contain vertex-disjoint paths P_1 and P_2, with endpoints s_1, t_1 and s_2, t_2, respectively. For $S = V$, inequalities (6) simplifies to $x_{s_1,\sigma(1)} + x_{s_2,\sigma(2)} + x_{t_1,\sigma(1)} + x_{t_2,\sigma(2)} \leq 3$. We refer to these inequalities as *cross inequalities*.

For the separation problem of the cross inequalities induced by the vertices in F, where $|V(F)| = f$, we implemented a $\mathcal{O}(fk^2)$ time complexity algorithm (the same complexity mentioned by Barboza (1997), without providing much detail; the algorithms may possibly be different). Next, we give more details on this separation algorithm.

Let $\widetilde{x} \in \mathbb{R}^{nk}$ be a fractional solution of formulation $\mathcal{C}$. Consider a linear ordering of the vertices in F which is obtained by traversing its vertices in clockwise order from an arbitrary fixed vertex. For every $j \in [f]$, we denote by $F(j)$ the jth vertex of F in such ordering. Furthermore, we define matrices L and R such that, for each $j \in [f]$ and each $i \in [k]$, $L(j,i) = \max_{j' \in [j]}\{\widetilde{x}_{F(j'),i}\}$ and $R(j,i) = \max_{j' \in [f] \setminus [j-1]}\{\widetilde{x}_{F(j'),i}\}$. In other words, $L(j,i)$ (resp. $R(j,i)$) corresponds to the maximum value in an entry of $\widetilde{x}$ indexed by i and by a vertex that appears before (resp. after) $F(j)$ in the ordering. Clearly, the construction of L and R takes time $\mathcal{O}(fk)$.

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

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

Note that, given $j \geq 2$ and $i_1, i_2 \in [k]$, $M(j, i_1, i_2)$ is the maximum value of $\widetilde{x}_{F(j_1),i_1} + \widetilde{x}_{F(j_2),i_2}$ for all $j_1, j_2 \in [j]$ with $j_1 < j_2$. Our algorithm works as follows: for every $j \in \{3, \ldots, f-1\}$ and every $i_1, i_2 \in [k]$ with $i_1 \neq i_2$, it checks whether $M(j-1, i_1, i_2) + \widetilde{x}_{F(j),i_1} + R(j+1, i_2) > 3$, that is, whether there is a violated cross inequality (w.r.t. F) such that $\sigma(1) = i_1$, $\sigma(2) = i_2$ and $t_1 = F(j)$. Clearly, one may also keep track of the violated inequalities.

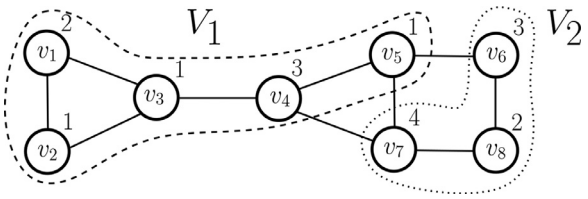

Fig. 1. An instance for BCP_2 and its unique optimal solution $\{V_1, V_2\}$ of value 8, where $V_1 = \{v_1, \ldots, v_5\}$ and $V_2 = \{v_6, v_7, v_8\}$.

4. Flow-based formulations

We present in this section a mixed integer linear programming formulation for BCP_k based on flows in a digraph. Given an input (G, w) for BCP_k, with $G = (V, E)$, we construct a digraph D as follows. First, we replace every edge of G with two arcs with the same endpoints and opposite directions; then we add a set $S = \{s_1, \ldots, s_k\}$ of k new vertices (sources), and add an arc from each vertex in S to each vertex of G. More formally, the vertex set of D is $V(D) = V \cup S$ and its arc set is $A(D) = \{(u, v), (v, u) : \{u, v\} \in E\} \cup \{(s_i, v) : i \in [k], v \in V\}$.

In Fig. 2(a) we illustrate the construction of the digraph D for the instance (G, w) of BCP_2 shown in Fig. 1. The idea behind the formulation is the following. In the digraph D we impose that, altogether, the k sources in S distribute a total of $w(G)$ amount of flow to the other vertices. Moreover, we impose that every non-source vertex v receives flow only from a single vertex of D and consumes $w(v)$ of the received flow. In this way, we have that each source s_i sends a positive flow to a single non-source vertex, which will in turn spread to other vertices defining precisely the vertices assigned to the ith class of the partition. (See Fig. 2(b).)

To model this idea, with each arc $a \in A(D)$, we associate a non-negative real variable f_a that represents the amount of flow passing through a, and a binary variable y_a (such that $y_a = 1$ if $f_a > 0$) that allow us to impose that flows from different sources do not mix. We denote by $\mathcal{F}_k(G, w)$, or simply $\mathcal{F}$, the corresponding formulation. In this formulation, the notation $y(A')$ (resp. $f(A')$) for arc sets $A' \subseteq A(D)$ stands for $\sum_{a \in A'} y_a$ (resp. $\sum_{a \in A'} f_a$).

$$\max f(\delta^+(s_1))$$

$$\text{s.t. } f(\delta^+(s_i)) \leq f(\delta^+(s_{i+1})) \qquad \forall i \in [k-1], \tag{7}$$

$$f(\delta^-(v)) - f(\delta^+(v)) = w(v) \qquad \forall v \in V, \tag{8}$$

$$f_a \leq w(G)\, y_a \qquad \forall a \in A(D), \tag{9}$$

$$y(\delta^+(s_i)) \leq 1 \qquad \forall i \in [k], \tag{10}$$

$$y(\delta^-(v)) \leq 1 \qquad \forall v \in V, \tag{11}$$

$$y_a \in \{0, 1\} \qquad \forall a \in A(D), \tag{12}$$

$$f_a \in \mathbb{R}_{\geq} \qquad \forall a \in A(D). \tag{13}$$

Inequalities (7) impose that the flows sent by sources $s_1, s_2, \ldots, s_k$ are in a non-decreasing order. This explains the objective function. Inequalities (8) guarantee that each vertex $v \in V$ consumes $w(v)$ of the flow that it receives. By (9), a positive flow can only pass through arcs that are chosen (arcs a for which $y(a) = 1$). Inequalities (10) impose that from every

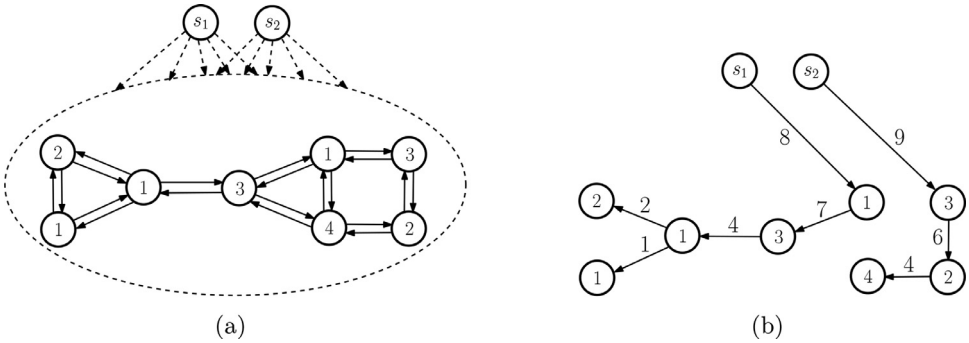

Fig. 2. (a) Digraph D obtained from the input graph (G, w) shown in Fig. 1, now with weights at the vertices, $w(G) = 17$. (b) An optimal solution for formulation $\mathcal{F}_2(G, w)$ in which only arcs with non-zero flow (shown on their side) are indicated.

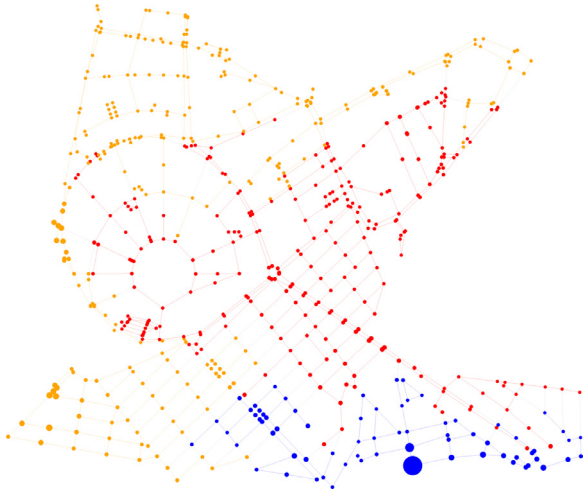

Fig. 3. A police patrolling instance, based on the map of the University of Campinas campus, and an optimal solution for $k = 3$. The radius of each vertex is proportional to its weight; the vertices in the lightest class are colored red.

source s_i at most one arc leaving it transports a positive flow to a single vertex in V. Inequalities (11) require that every non-source vertex receives a positive flow from at most one vertex of D.

Because of inequalities (11), the flows sent by any two distinct sources do not pass through a same vertex. That is, if a source s_i sends an amount of flow, say w_i, this amount w_i is distributed to a subset of vertices, say V_i (with total weight w_i); and all subsets V_i, for $i \in [k]$, are mutually disjoint (each one being defined by an arborescence rooted at s_i). Moreover, w_i is exactly the sum of the weights of the vertices that receive flow from s_i, and $G[V_i]$ is a connected subgraph of G. By summing up the inequalities (8) we can see that the k sources, altogether, distribute a total of $w(G)$ amount of flow.

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

The proposed formulation has a total of $2nk + 4m$ variables (half of them binary), and only $\mathcal{O}(n + m + k)$ constraints, where $n = |V|$ and $m = |E|$. Possibly, some of its drawbacks are the large amount of symmetric solutions and the dependency of inequalities (9) on the weights assigned to the vertices. To overcome such disadvantages, we introduce in the next section another model based on flows that considers a total order of the vertices to avoid symmetries and uncouples the weights assigned to the vertices from the flow circulating in the digraph. As one may expect, such model uses additional variables and constraints. However, its size is still bounded by a polynomial function on the size of the input graph and k.

4.1. Asymmetric flow-based formulation

Our second compact formulation for BCP$_k$ is also based on flows in a digraph D, this time used in a different way, independent from the vertex-weights. Given an input (G, w), where $G = (V, E)$, we construct a digraph D with a single source s, as follows. We have $V(D) = V \cup \{s\}$ and

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

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

In this formulation, we consider flows of type i, for each $i \in [k]$, corresponding to the classes $i \in [k]$ (that they will define). We use real variables $f_{a,i}$ for $a \in A(D)$ and $i \in [k]$, where $f_{a,i}$ is the amount of flow of type i that passes through arc a: if non-zero and both ends of arc a are distinct from s, then these ends belong to class i (of the subpartition). Basically, a class i with vertex set, say V_i, will consist of an arborescence $\overrightarrow{T_i}$ rooted at a vertex, which will receive a flow of value $|V_i|$ from s. Then, this root consumes one unit of flow and sends $|V_i| - 1$ amount of flow to its neighbors; then each of the vertices that receives a non-null flow consumes one unit of flow and sends the remaining flow to its neighbors, until vertices that receive one unit of flow are reached (they become leaves of the arborescence). The ordering of the vertices is used to impose that, among the vertices of each arborescence, the root is the smallest one (this helps breaking symmetries).

To control the spreading of the flows, we also use binary variables $y_{a,i}$, for $a \in A(D)$ and $i \in [k]$, such that $y_{a,i} = 1$ if and only if in arc a a non-null flow of type i passes through it. Using these variables, we are able to write restrictions to impose an ordering in the classes, according to their weights, being class 1 the lightest one (which explains the objective function). In the next formulation, the notation $y(A', i)$ (resp. $f(A', i)$) for arc sets $A' \subseteq A(D)$ stands for $\sum_{a \in A'} y_{a,i}$ (resp. $\sum_{a \in A'} f_{a,i}$). We denote by $\mathcal{F}'_k(G, w)$, or simply, $\mathcal{F}'$, the corresponding formulation.

$$\max \sum_{v \in V(D)} w(v)\, y(\delta^-(v), 1)$$

$$\text{s.t.} \sum_{v \in V} w(v)\, y(\delta^-(v), i) \le \sum_{v \in V} w(v)\, y(\delta^-(v), i+1) \quad \forall i \in [k-1], \tag{14}$$

$$y(\delta^+(s), i) \leq 1 \qquad \forall i \in [k], \tag{15}$$

$$\sum_{i\in[k]} y(\delta^-(v), i) \leq 1 \qquad \forall v \in V, \tag{16}$$

$$y_{sv,i} + y(\delta^-(u), i) \leq 1 \qquad \forall u, v \in V,\ v \succ u,\ i \in [k], \tag{17}$$

$$f_{a,i} \leq n\, y_{a,i} \qquad \forall a \in A(D),\ i \in [k], \tag{18}$$

$$f(\delta^+(v), i) \leq f(\delta^-(v), i) \qquad \forall v \in V,\ i \in [k], \tag{19}$$

$$\sum_{i\in[k]} f(\delta^-(v), i) - \sum_{i\in[k]} f(\delta^+(v), i) = 1 \qquad \forall v \in V, \tag{20}$$

$$y_{a,i} \in \{0, 1\} \qquad \forall a \in A(D),\ i \in [k], \tag{21}$$

$$f_{a,i} \geq 0 \qquad \forall a \in A(D),\ i \in [k]. \tag{22}$$

To show that $\mathcal{F}'_k(G, w)$ indeed models BCP_k correctly, let us consider the following polytope.

$$\mathcal{Q}_k(G, w) = \text{conv}\{(y, f) \in \mathbb{B}^{(n+2m)k} \times \mathbb{R}^{(n+2m)k} : (y, f) \text{ satisfies } (14) - (22)\}.$$

Let $\mathcal{V}$ be a connected k-subpartition of G such that $w(V_i) \leq w(V_{i+1})$ for all $i \in [k-1]$. Then, for each integer $i \in [k]$, there exists in D an arborescence $\overrightarrow{T_i}$ rooted at r_i such that $V(\overrightarrow{T_i}) = V_i$ and $v \succ r_i$ for all $v \in V_i \setminus \{r_i\}$. Now, let g_i be the function $g_i : A(\overrightarrow{T_i}) \cup \{(s, r_i)\} \to \mathbb{R}_{\geq}$ defined as follows: $g_i((u, v)) = 1$ if v is a leaf of $\overrightarrow{T_i}$, and $g_i((u, v)) = 1 + \sum_{(v,z)\in A(\overrightarrow{T_i})} g_i((v, z))$, otherwise. It follows from this definition that $g_i((s, r_i)) = |V_i|$.

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

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

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

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

Proposition 4. *The polytope $\mathcal{Q}_k(G, w)$ is precisely the polytope*

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

Proof. Let (y, f) be an extreme point of $\mathcal{Q}_k(G, w)$; and for every $i \in [k]$, let $U_i = \{v \in V : y(\delta^-(v), i) = 1\}$. Inequalities (15) guarantee that, for each type i, at most one arc leaving s is chosen. Therefore, we have that $\{U_i\}_{i\in[k]}$ is a connected k-partition of G.

Inequalities (16) impose that, for every vertex $v \in V$, at most one of the arcs entering it is chosen. Observe that inequalities (18) force that a flow of type i can only pass through an arc of type i if this arc is chosen. Hence, every vertex receives at most one type of flow from its in-neighbors. Furthermore, inequalities (19) and (20) guarantee that the flow that enters a vertex and leaves it are of the same type, and that each vertex consumes exactly one unit of such flow.

To prove the converse, let $\mathcal{V} = \{V_i\}_{i\in[k]}$ be a connected k-partition of G. We assume without loss of generality that $w(V_i) \leq w(V_{i+1})$ for all $i \in [k-1]$. Let (y, f) be a vector such that $y = \rho(\mathcal{V})$ and $f = \tau(\mathcal{V})$. For each $i \in [k]$, every vertex in $\overrightarrow{T_i}$ has in-degree at most one, and r_i is the smallest vertex in $V(\overrightarrow{T_i})$ with respect to the order $\succ$. Thus, inequalities (16) and (17) hold for y. From the definition of $\rho(\mathcal{V})$, the entry of y indexed by (s, r_i) and i equals one, for all $i \in [k]$. Consequently, y also satisfies inequalities (15). Recall that $g_i((s, r_i)) = |V_i|$ for every $i \in [k]$. This clearly implies that inequalities (18) are satisfied by (y, f).

Note that, for every $i \in [k]$, the function g_i assigns to each arc $(u, v) \in A(\overrightarrow{T_i}) \cup \{(s, r_i)\}$ the value one plus the sum of the sizes of the sub-arborescences of $\overrightarrow{T_i}$ rooted at the out-neighbors of v in $\overrightarrow{T_i}$. Hence, inequalities (19) and (20) hold for y. Finally, inequalities (14) are satisfied, as we assumed that the elements of partition $\mathcal{V}$ are in a non-decreasing order of weights. Therefore, we conclude that (y, f) belongs to $\mathcal{Q}_k(G, w)$. □

5. Implementation details

The computational experiments were carried out on a PC with Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz, 40 cores, 64 GB RAM and Ubuntu 18.04.2 LTS. The code was written in C++ using the graph library Lemon (Dezső, Jüttner, & Kovács, 2011).

We implemented a branch-and-cut algorithm to solve the cut-based formulation $\mathcal{C}$ using SCIP Optimization Suite 6.0 (Gleixner et al., 2018) and Gurobi 9.0 as the LP solver. Unlike Gurobi, SCIP allows for multiple rounds of cut generation in non-root nodes of the branch-and-bound tree. Moreover, it has built-in routines for separating the lifted minimal cover inequalities, and supports customized domain propagation routines. We give more details about these features later in this section.

We implemented branch-and-bound algorithms (using only Gurobi 9.0) to solve our flow-based formulations $\mathcal{F}$ and $\mathcal{F}'$. Since Matić (2014) and Zhou et al. (2019) implementations were not publicly available, we also implemented their models using Gurobi. This way, we executed all the experiments in the same computational environment.

Due to small improvements in the preliminary experiments, we replace inequalities (1), (7) and (14) with equalities. Furthermore, to evaluate the performance of the mentioned formulations, all standard cuts used by SCIP and Gurobi are deactivated, except for the lifted minimal cover inequalities.

5.1. Cover inequalities

Consider an input instance (G, w) of BCP_k. As we have previously mentioned, the following inequalities are valid for $\mathcal{P}_k(G, w)$ and may be easily derived from constraints (1) of formulation $\mathcal{C}_k(G, w)$.

$$\sum_{v\in V} w(v)\, x_{v,i} \leq \frac{w(G)}{k - i + 1}, \qquad \forall i \in [k-1]. \tag{23}$$

Note that, for each $i \in [k-1]$, the corresponding inequality (23) defines a knapsack problem of budget $w(G)/(k-i+1)$. Hence, we can take advantage of the extensive work regarding strong inequalities for the 0/1 knapsack polytope as follows. For each inequality of class (23), we use the heuristics mentioned by Wolter (2006) to separate lifted minimal cover inequalities and extended weight inequalities. Such valid inequalities are obtained from a sequence of up- and down-lifting operations on inequalities that are valid for projections of the initial 0/1 knapsack polytope. For more details about this procedure, the reader is referred to Wolter's thesis.

5.2. Domain propagation

Suppose that our algorithm is currently exploring a node in the branch-and-bound tree, *domain propagation* refers to the technique that tries to tighten the variable bounds based on the domain of

the variables in the current node. When a complete description of the problem formulation is available to the solver, the solver itself can reduce the domain of the variables. However, when implementing a branch-and-cut algorithm, for example, the solver may not be able to effectively reduce the domain of the variables, since only a small part of the inequalities might be available to it. Thus, based on the work of Hojny, Joormann, Lüthen, and Schmidt (2020), we implemented an algorithm to effectively reduce the domains of the variables of formulation $\mathcal{C}$.

For completeness, we now briefly describe the domain propagation technique (see Hojny et al. (2020) for more details). For every $j \in [k]$, let $F_j \subseteq V$ be the set of vertices fixed on class j in the current node of the branch-and-bound tree; in other words, $F_j = \{v \in V : x_{v,j} = 1\}$. Let us fix $i \in [k]$ and assume $F_i \neq \emptyset$. Consider the graph $G_i = G\left[V \setminus \left(\bigcup_{i' \in [k]\setminus\{i\}} F_{i'}\right)\right]$, that is, G_i is the graph obtained from G by removing the vertices that were fixed in classes distinct from i. Let $\{C_j\}_{j\in[t]}$ be the connected components of G_i, and suppose, without loss of generality, that $F_i \cap V(C_t) \neq \emptyset$. Now for every vertex $u \in V(C_j)$, with $j \in [t-1]$, we either set $x_{u,i} = 0$; or, if variable $x_{u,i}$ was already fixed to one, we consider that the current branch-and-bound node is infeasible and therefore it can be pruned.

6. Benchmark instances

Computational experiments on instances consisting of grid graphs and random connected graphs are reported in Matić (2014); Zhou et al. (2019). In this work, besides considering instances previously proposed in the literature, we also considered larger graphs, different weight distributions and real world instances.

The grid instances are named in the format *gg_height_width_*[a|b]. The random instances are named in the format *rnd_n_m_*[a|b], where n is the number of vertices and m is the number of edges in the graph. In both cases, a and b indicate the intervals in which the weights (all integers) are uniformly distributed: a $= [1, 100]$ and b $= [1, 500]$.

To generate a random connected graph with n vertices and m edges (with $m > n-1$), we first use Wilson's algorithm Wilson (1996) to generate a random spanning tree T on n vertices, and then add $m-n+1$ distinct new edges randomly selected from $E(K_n) \setminus E(T)$ with uniform probability. Wilson's algorithm returns a spanning tree T sampled from the set τ_n — of all possible spanning trees of K_n — with probability $1/|\tau_n|$.

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

Finally, we also considered instances for BCP_k coming from a real-world application, namely demarcation of preventive police patrol areas (Assunção & Furtado, 2008). This problem consists in subdividing a given map into k contiguous regions with roughly the same crime rate (in order to balance the work load of k police patrol teams).

Using OSMnx (Boeing, 2017) library, we transformed maps from OpenStreetMap (https://www.openstreetmap.org) into undirected graphs. The edges in the graphs generated by OSMnx corresponds to sections of the streets. As some edges may be too "long" (over 200meters), we subdivide these long edges into smaller edges so that the length of each edge is limited to 200 meters.

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

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

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

The formula we have used to assign weights to the vertices expresses the idea that the influence of a crime over a region is a Gaussian distribution on the distance to the crime.

The generated instances are available at the website of the Laboratory of Optimization and Combinatorics at the University of Campinas[1].

7. Computational results

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

Henceforth, when we refer to any of the formulations, it should be understood that we are referring to the corresponding exact algorithms that we have implemented for them. Thus, CUT refers to the branch-and-cut algorithm for the cut-based formulation $\mathcal{C}$, while FLOW and FLOW2 refer to the corresponding branch-and-bound algorithms for the flow-based formulations $\mathcal{F}$ and $\mathcal{F}'$.

We omit the results for FLOW2 because, according to our experiments, it is (on average) over 10 times slower than FLOW. We also omit the results obtained with the branch-and-bound algorithm for Matić's formulation since it was, on average, four times slower than Zhou et al. Furthermore, it solved fewer instances than the other implemented algorithms.

In Table 1 we show the impact of separating cross inequalities. CUT-CROSS refers to CUT with the cross inequalities. Conn Cuts (resp. Cross Cuts) show the number of connectivity (resp. cross) inequalities separated by the algorithm. We note that, on average, CUT-CROSS was much faster than CUT on all grid instances. Moreover, only CUT-CROSS was able to solve all instances with more than 100 vertices within the time limit.

However, as it can be seen in Table 2, FLOW had better execution times than CUT-CROSS on most of the grid instances. Moreover, on grids with over 100 vertices, Matić and Zhou formulations were not able to solve the majority of the instances, while CUT-CROSS and FLOW solved all of them. Examining carefully the experimental results, we noted that, when the optimal solution costs deviates reasonably from the trivial upper bound of $w(G)/k$, FLOW frequently fails to find a provably optimal solution within the time limit. A more in-depth analysis of this behavior is provided in Section 7.1.

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

Table 1
Computational results for BCP_2 on grid graphs showing the efficiency of the cross inequalities.

Instance	CUT			CUT-CROSS			
	Sol	Conn Cuts	Time	Sol	Conn Cuts	Cross Cuts	Time
gg_05_05_a	10	3504	0.50	10	449	478	**0.15**
gg_05_05_b	10	24,252	3.32	10	2817	3841	**0.84**
gg_05_06_a	10	6184	0.80	10	1056	1286	**0.32**
gg_05_06_b	10	21,125	2.76	10	3311	4536	**1.02**
gg_05_10_a	10	44,635	5.63	10	2187	2147	**0.55**
gg_05_10_b	10	67,926	9.37	10	3776	3884	**1.10**
gg_05_20_a	6	2,821,710	521.39	10	9578	8768	**2.43**
gg_05_20_b	10	1,836,157	325.73	10	23,832	23,613	**6.47**
gg_07_07_a	10	41,279	6.84	10	2128	2220	**0.78**
gg_07_07_b	10	184,730	30.99	10	3035	3647	**1.29**
gg_07_10_a	10	301,289	57.34	10	2154	1942	**0.81**
gg_07_10_b	10	427,441	87.24	10	5954	6243	**2.69**
gg_10_10_a	10	1,373,429	389.42	10	2987	2432	**1.20**
gg_10_10_b	9	1,289,465	370.06	10	3652	3343	**2.12**
gg_15_15_a	0	-	-	10	13,032	7371	**8.97**
gg_15_15_b	0	-	-	10	15,411	10,182	**13.88**

Table 2
Computational results for BCP_2 on grid graphs.

Instance	CUT-CROSS			FLOW			Zhou et al.		
	Sol	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time
gg_05_05_a	10	56	0.15	10	919	**0.09**	10	2564	0.45
gg_05_05_b	10	913	**0.84**	10	320,162	19.17	10	7341	1.22
gg_05_06_a	10	189	0.32	10	460	**0.07**	10	1843	0.43
gg_05_06_b	10	844	1.02	10	1592	**0.13**	10	4454	0.91
gg_05_10_a	10	151	0.55	10	500	**0.15**	10	12,542	3.27
gg_05_10_b	10	468	1.10	10	806	**0.17**	10	18,258	5.13
gg_05_20_a	10	246	2.43	10	454	**0.25**	2	116,374	42.32
gg_05_20_b	10	950	6.47	10	1146	**0.34**	4	2,221,308	522.95
gg_07_07_a	10	200	0.78	10	645	**0.14**	10	10,346	2.66
gg_07_07_b	10	497	1.29	10	784	**0.18**	10	15,657	3.60
gg_07_10_a	10	136	0.81	10	366	**0.16**	10	529,336	116.87
gg_07_10_b	10	650	2.69	10	1304	**0.29**	9	400,859	66.10
gg_10_10_a	10	100	1.20	10	186	**0.20**	6	1,472,514	396.10
gg_10_10_b	10	301	2.12	10	905	**0.36**	5	544,515	141.69
gg_15_15_a	10	125	8.97	10	184	**0.40**	0	-	-
gg_15_15_b	10	458	13.88	10	696	**0.59**	0	-	-

Table 3 shows that both of our formulations are also faster (on average) on random graphs instances. Moreover, the execution time for CUT was better than FLOW on some instances. A closer look at the number of vertices and edges of the instances, revealed that the density of edges of the input graphs was a crucial factor for the performance of both algorithms. In order to further explore this observation, we generated random graphs with a larger number of vertices ($n = 500$ and $n = 1000$) and different values of density ($m = n^{\alpha}$, where $\alpha \in \{1.1, 1.2, 1.3, 1.4, 1.5\}$). The experiments on these instances are shown in Table 4 and indicate that CUT performs better than FLOW when $m \geq n^{1.2}$. One possible explanation for these results is that, in such dense instances, as many pairs of vertices are adjacent, the algorithm might spend less time in the separation routines since the connectivity inequalities are only for non-adjacent pairs of vertices (see inequalities (3)). Another possibility is that the flow-based formulation $\mathcal{F}$ has variables associated with the edges of the input graph, which means that this formulation has additional symmetries for all the different spanning trees of the classes.

In our experiments, FLOW was the only algorithm able to solve the police patrolling instances within the time limit. As Table 5 indicates, the problem becomes much harder to solve when $k > 2$.

The experiments that we carried out for $k > 2$ on grid graphs and random graphs are shown in Table 6. To compare with the formulation due to Zhou et al. (2019), designed for MIN-MAX BCP_k, we considered our cut-based and flow-based formulations with min-max objective (i.e. minimizing the weight of the kth class), denoted by CUT (MIN-MAX) and FLOW (MIN-MAX). Since CUT (MIN-MAX) was not able to solve most of the tested instances within the time limit, its running times are omitted. We remark that the unique instance (out of 10) that Zhou et al.'s branch-and-bound solved in 508.93 seconds, FLOW solved in 388.67 seconds (and additionally solved other 5 instances).

Table 7 shows some instances consisting of grid graphs with large number of vertices, and different weight distributions (a and b), that could only be solved by FLOW.

Our computational experiments show that the algorithms based on the formulations we presented in this work substantially outperform the algorithms based on the previous models in the literature. On most of the tested instances, FLOW had the best average running time. On the other hand, CUT explored (on average) a smaller number of nodes in the branch-and-bound tree, and it is the fastest on instances consisting of dense graphs.

7.1. Integrality gap

To evaluate the strength of ILP and MILP formulations it is common to compare the cost of optimal (integral) solutions with the cost of optimal solutions of the corresponding linear relaxations. We now briefly discuss about the quality of the linear relaxation of formulations $\mathcal{C}$ and $\mathcal{F}$.

Table 3
Computational results for BCP_2 on random graphs.

Instance	CUT			FLOW			Zhou et al.		
	Sol	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time
rnd_50_70_a	10	201	0.57	10	256	**0.08**	10	958	0.46
rnd_50_70_b	10	847	1.89	10	928	**0.12**	10	1872	0.65
rnd_50_100_a	10	46	0.16	10	203	**0.10**	10	399	0.43
rnd_50_100_b	10	251	0.51	10	683	**0.16**	10	648	0.45
rnd_50_400_a	10	15	**0.05**	10	11	0.18	10	58	1.07
rnd_50_400_b	10	129	0.24	10	11	**0.19**	10	177	1.52
rnd_70_100_a	10	121	0.69	10	119	**0.09**	10	1185	0.78
rnd_70_100_b	10	389	1.56	10	876	**0.21**	10	1710	0.79
rnd_70_200_a	10	13	**0.07**	10	23	0.14	10	156	0.97
rnd_70_200_b	10	168	0.40	10	66	**0.17**	10	357	0.85
rnd_70_600_a	10	10	**0.06**	10	1	0.21	10	19	1.90
rnd_70_600_b	10	71	**0.20**	10	12	0.27	10	160	2.28
rnd_100_150_a	10	244	1.84	10	206	**0.20**	10	1918	2.06
rnd_100_150_b	10	1614	9.46	10	531	**0.25**	10	1711	1.77
rnd_100_300_a	10	30	**0.20**	10	69	0.21	10	247	1.44
rnd_100_300_b	10	91	0.39	10	57	**0.23**	10	565	2.32
rnd_100_800_a	10	3	**0.04**	10	1	0.30	10	29	3.29
rnd_100_800_b	10	41	**0.18**	10	83	0.35	10	71	3.01
rnd_200_300_a	10	80	2.35	10	519	**0.41**	10	1951	7.04
rnd_200_300_b	10	397	8.96	10	849	**0.49**	10	2857	10.71
rnd_200_600_a	10	14	**0.35**	10	356	0.56	10	728	9.65
rnd_200_600_b	10	107	1.02	10	515	**0.62**	10	995	9.81
rnd_200_1500_a	10	1	**0.06**	10	1	0.64	10	3	7.34
rnd_200_1500_b	10	8	**0.13**	10	303	1.05	10	241	14.85
rnd_300_500_a	10	62	5.08	10	732	**0.72**	10	2811	28.61
rnd_300_500_b	10	303	8.88	10	1029	**0.84**	10	3629	21.84
rnd_300_1000_a	10	8	**0.47**	10	614	1.07	10	982	20.34
rnd_300_1000_b	10	57	**1.08**	10	1113	1.39	10	1030	19.75
rnd_300_2000_a	10	1	**0.10**	10	1	1.05	10	35	25.69
rnd_300_2000_b	10	28	**0.60**	10	754	1.90	10	54	80.63

Table 4
Computational results for BCP_2 on dense random graphs.

Instance		CUT			FLOW		
	m	Sol	Nodes	Time	Sol	Nodes	Time
rnd_500_931_a	$n^{1.1}$	10	35	12.41	10	1397	**2.01**
rnd_500_931_b	$n^{1.1}$	10	309	26.86	10	1100	**1.67**
rnd_500_1733_a	$n^{1.2}$	10	7	**1.16**	10	1243	2.98
rnd_500_1733_b	$n^{1.2}$	10	42	**2.58**	10	2101	3.45
rnd_500_3226_a	$n^{1.3}$	10	2	**0.31**	10	99	2.17
rnd_500_3226_b	$n^{1.3}$	10	22	**0.79**	10	495	3.72
rnd_500_6005_a	$n^{1.4}$	10	1	**0.30**	10	42	4.74
rnd_500_6005_b	$n^{1.4}$	10	9	**0.77**	10	1257	13.55
rnd_500_11180_a	$n^{1.5}$	10	1	**0.53**	10	40	3.08
rnd_500_11180_b	$n^{1.5}$	10	4	**0.86**	10	1666	23.59
rnd_1000_1995_a	$n^{1.1}$	10	31	54.04	10	2163	**6.48**
rnd_1000_1995_b	$n^{1.1}$	10	64	77.14	10	2726	**6.27**
rnd_1000_3981_a	$n^{1.2}$	10	12	**4.56**	10	1942	14.19
rnd_1000_3981_b	$n^{1.2}$	10	12	**5.29**	10	3545	20.95
rnd_1000_7943_a	$n^{1.3}$	10	1	**0.91**	10	800	24.32
rnd_1000_7943_b	$n^{1.3}$	10	1	**0.88**	10	2891	48.37
rnd_1000_15849_a	$n^{1.4}$	10	1	**1.13**	10	148	27.59
rnd_1000_15849_b	$n^{1.4}$	10	1	**1.08**	10	10,155	134.55
rnd_1000_31623_a	$n^{1.5}$	10	1	**1.94**	10	523	58.47
rnd_1000_31623_b	$n^{1.5}$	10	1	**1.98**	10	2343	144.46

Table 5
Performance of FLOW to solve BCP_k on police patrolling instances when $k \in \{2, 3, 4, 5, 6\}$.

Instance	2	3	4	5	6
barao_1913_2752	7.31	429.72	-	-	-
campinas_centro_579_942	6.60	159.88	-	-	-
chicago_englewood_1560_2579	8.18	118.24	-	-	-
chicago_lakeview_1004_1563	3.43	34.73	251.97	-	-
chicago_loop_624_971	1.93	31.13	459.44	-	-
la_hollywood_1368_2030	19.75	102.91	739.39	587.44	-
la_skidrow_1667_2459	41.59	46.84	1314.81	-	-
nyc_chelsea_822_1228	2.28	72.75	109.59	861.67	-
nyc_hellskitchen_498_746	1.30	8.82	3.13	-	-
unicamp_624_901	1.17	83.52	-	-	710.68

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

Instance		FLOW (MIN-MAX)			Zhou et al.		
	k	Sol	Nodes	Time	Sol	Nodes	Time
gg_07_10_a	3	10	4,177	**0.93**	2	1,072,107	603.97
gg_07_10_a	4	10	25,242	**3.86**	1	99,880	68.95
gg_07_10_a	5	10	1,578,669	**247.61**	0	-	-
gg_07_10_a	6	3	3,077,826	**482.45**	0	-	-
rnd_100_150_a	3	10	1,933	**0.74**	9	109,799	70.04
rnd_100_150_a	4	10	13,404	**3.14**	4	1,019,886	641.36
rnd_100_150_a	5	10	627,636	**149.27**	0	-	-
rnd_100_150_a	6	6	2,268,233	682.23	1	205,247	508.93

Table 7
Performance of FLOW to solve BCP_2 on large grids.

Instance	a			b		
	Sol	Nodes	Time	Sol	Nodes	Time
gg_30_30	10	466	2.71	10	731	3.14
gg_60_60	10	420	20.06	10	751	19.49
gg_90_90	10	727	89.21	10	874	88.78
gg_120_120	10	575	148.93	10	736	207.57
gg_150_150	10	546	302.97	10	1087	334.19
gg_180_180	10	305	571.58	10	451	599.53
gg_210_210	10	400	1100.32	9	325	1064.07
gg_240_240	3	467	1709.85	6	138	1422.05

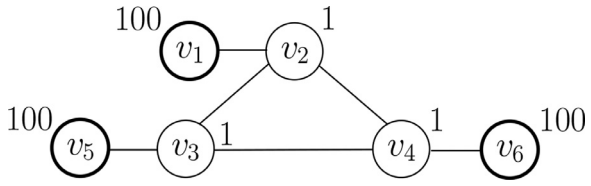

Fig. 4. Instance (G, w) of BCP_2, with $w(v_1) = w(v_5) = w(v_6) = 100$ and $w(v_2) = w(v_3) = w(v_4) = 1$.

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

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

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

This linear relaxation may be strengthened by adding the cover inequalities mentioned in Section 5.1. As an example, consider the instance (G, w) of BCP_2 illustrated in Fig. 4. Note that $\{v_1, v_5\}$ is a cover since $w(v_1) + w(v_5) > w(G)/2$, and thus the lifted cover inequality $x_{v_1,1} + x_{v_5,1} + x_{v_6,1} \leq 1$ is valid for the polytope associated with the linear relaxation of $\mathcal{C}_2(G, w)$. Such inequality cuts off any optimal solution for the linear relaxation of $\mathcal{C}_2(G, w)$ whose cost is $w(G)/2 = 151.5$, as we observed before.

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

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

8. Concluding remarks

We proposed an ILP and two MILP formulations for the Balanced Connected k-Partition Problem. The first of our formulations, denoted by $\mathcal{C}$, has an exponential number of inequalities to guarantee connectivity. We presented a polynomial-time separation algorithm and a lifting procedure for these inequalities. Moreover, we introduced a new class of valid inequalities for this formulation, and showed how to separate a special case of them (namely, cross inequalities) on planar graphs in polynomial time. The experiments showed that the addition of these inequalities improved greatly the performance of the corresponding branch-and-cut algorithm.

Then, we introduced two compact MILP formulations based on flows in a digraph constructed from the input graph. The first of them, $\mathcal{F}$, has a polynomial number of variables and constraints, and is based on flows whose values depend on the weights of the vertices. To overcome the apparent disadvantages of this formulation, like symmetries and vertex-weight dependent flows, we designed $\mathcal{F}'$. Although more complex than the former, this formulation avoids some symmetries and uses flows of small values just to control connectedness of the classes. However, in our computational experiments, the performance of the branch-and-bound algorithm for formulation $\mathcal{F}$ was significantly superior to the one for $\mathcal{F}'$.

Our formulations impose a non-decreasing weight ordering of the classes of a balanced connected k-partition $\{V_i\}_{i \in [k]}$, that is, $w(V_i) \leq w(V_{i+1})$ for all $i \in [k-1]$. Therefore, one may easily modify the objective function to capture other concepts of "balance" such as minimize the heaviest class or the maximum difference of weights between the classes. Recall that all these problems are equivalent for $k = 2$, but they are not when $k > 2$. Furthermore, the addition of the constraints that order the classes by weights reduces the symmetry of our formulations. In the case of the asymmetric flow-based formulation $\mathcal{F}'$, tests showed that removing symmetrical solutions does not necessarily imply better results in practice. Preliminary experiments with other formulations corroborate with the idea that allowing symmetric solutions may yield faster algorithms on some classes of instances. Further investigation is still needed to fully understand the effectiveness (or not) of breaking symmetries.

Table 8 summarizes the number of variables and constraints in the formulations that we have considered here. In comparison with

Table 8
Comparison of formulations for BCP_k in terms of number of variables and constraints on an input graph with n vertices and m edges.

Formulation	# Binary Variables	# Real Variables	# Constraints
Cut $\mathcal{C}$	kn	0	$\mathcal{O}(2^n)$
Flow $\mathcal{F}$	$kn+2m$	$kn+2m$	$\mathcal{O}(n+m+k)$
Asym. flow $\mathcal{F}'$	$k(n+2m)$	$kn+2km$	$\mathcal{O}(k(n^2+m))$
Zhou et al. (2019)	$2k(n+m)$	$n+2m+1$	$\mathcal{O}(km)$

Zhou et al. formulation, our Flow formulation has a smaller number of binary variables and constraints.

The computational experiments show that CUT and FLOW have a considerably better performance than all previous exact algorithms in the literature. For the instances consisting of grid graphs, FLOW was able to solve (within the time limit) instances of size over 400 times larger than the size of the instances that could be solved by the previous exact methods in the literature. On average, CUT was approximately 8 and 2 times faster than the algorithms based on the formulations proposed by Matić and Zhou et al., respectively. Moreover, FLOW was 22 times faster than Matić's algorithm and 5 times faster than Zhou et al's algorithm.

Acknowledgments

Research partially supported by grant 2015/11937-9, São Paulo Research Foundation (FAPESP). Miyazawa is supported by CNPq (314366/2018-0 and 425340/2016-3) and FAPESP (2016/01860-1). Moura is supported by FAPESP (2016/21250-3 and 2017/22611-2), CAPES, and Pró-Reitoria de Pesquisa da Universidade Federal de Minas Gerais. Ota is supported by CNPq (167242/2018-0). Wakabayashi is supported by CNPq (306464/2016-0 and 423833/2018-9).

References

Alimonti, P., & Calamoneri, T. (1999). On the complexity of the max balance problem. In *Proceedings of the Argentinian workshop on theoretical computational science (WAIT'99)* (pp. 133–138).
Andersson, M., Gudmundsson, J., Levcopoulos, C., & Narasimhan, G. (2002). Balanced partition of minimum spanning trees. In P. M. A. Sloot, A. G. Hoekstra, C. J. K. Tan, & J. J. Dongarra (Eds.), *Proceedings of the computational science (ICCS 2002)* (pp. 26–35). Springer Berlin Heidelberg.
Assunção, T., & Furtado, V. (2008). A heuristic method for balanced graph partitioning: An application for the demarcation of preventive police patrol areas. In H. Geffner, R. Prada, I. M. Alexandre, & N. David (Eds.), *Proceedings of the advances in artificial intelligence (IBERAMIA 2008)* (pp. 62–72).
Barboza, E. U. (1997). *Problemas de classificação com restrições de conexidade flexibilizadas*. Master's thesis Universidade Estadual de Campinas.
Becker, R. I., Lari, I., Lucertini, M., & Simeone, B. (1998). Max-min partitioning of grid graphs into connected components. *Networks, 32*, 115–125.
Becker, R. I., & Perl, Y. (1983). Shifting algorithms for tree partitioning with general weighting functions. *Journal of Algorithms, 4*, 101–120.
Becker, R. I., Schach, S. R., & Perl, Y. (1982). A shifting algorithm for min-max tree partitioning. *Journal of ACM, 29*, 58–67.
Boeing, G. (2017). OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems, 65*, 126–139.
Borndörfer, R., Elijazyfer, Z., & Schwartz, S. (2019). *Approximating balanced graph partitions. technical report*. ZIB. http://nbn-resolving.de/urn:nbn:de:0297-zib-73675.
Chataigner, F., Salgado, L. R. B., & Wakabayashi, Y. (2007). Approximation and inapproximability results on balanced connected partitions of graphs. *Discrete Mathematics & Theoretical Computer Science, 9*.
Chen, G., Chen, Y., Chen, Z.-Z., Lin, G., Liu, T., & Zhang, A. (2020). Approximation algorithms for the maximally balanced connected graph tripartition problem. *Journal of Combinatorial Optimization*, 1–21. https://doi.org/10.1007/s10878-020-00544-w.
Chlebíková, J. (1996). Approximating the maximally balanced connected partition problem in graphs. *Information Processing Letters, 60*, 225–230.
De Aragão, M. P., & Uchoa, E. (1999). The γ-connected assignment problem. *European Journal of Operational Research, 118*, 127–138.
Dezső, B., Jüttner, A., & Kovács, P. (2011). Lemon – an open source c++ graph template library. *Electronic Notes in Theoretical Computer Science, 264*, 23–45.
Dyer, M., & Frieze, A. (1985). On the complexity of partitioning graphs into connected subgraphs. *Discrete Applied Mathematics, 10*, 139–153.
Frederickson, G. N. (1991). Optimal algorithms for tree partitioning. In *Proceedings of the ACM-SIAM symposium on discrete algorithms SODA '91* (pp. 168–177).
Gleixner, A., Bastubbe, M., Eifler, L., Gally, T., Gamrath, G., Gottwald, R. L., & Witzig, J. (2018). *The SCIP optimization suite 6.0. ZIB-Report*. Zuse Institute Berlin. http://nbn-resolving.de/urn:nbn:de:0297-zib-69361.
Goldberg, A. V., & Tarjan, R. E. (1988). A new approach to the maximum-flow problem. *Journal of the ACM, 35*, 921–940.
Grötschel, M., Lovász, L., & Schrijver, A. (2012). *Geometric algorithms and combinatorial optimization*: 2 (1st). Springer.
Györi, E. (1978). On division of graph to connected subgraphs. In *Combinatorics (proc. fifth hungarian colloq., koszthely, 1976)*. In *Colloq. Math. Soc. János Bolyai: vol. 18* (pp. 485–494).
Hojny, C., Joormann, I., Lüthen, H., & Schmidt, M. (2020). Mixed-integer programming techniques for the connected max-k-cut problem. *Mathematical Programming Computation*, 1–58.
Kawarabayashi, K., Kobayashi, Y., & Reed, B. (2012). The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B, 102*, 424–435.
Lovász, L. (1977). A homology theory for spanning tress of a graph. *Acta Mathematica Academiae Scientiarum Hungarica, 30*, 241–251.
Lucertini, M., Perl, Y., & Simeone, B. (1989). Image enhancement by path partitioning. In V. Cantoni, R. Creutzburg, S. Levialdi, & G. Wolf (Eds.), *Recent issues in pattern analysis and recognition*. In *Lecture Notes in Computer Science: vol. 399* (pp. 12–22).
Lucertini, M., Perl, Y., & Simeone, B. (1993). Most uniform path partitioning and its use in image processing. *Discrete Applied Mathematics, 42*, 227–256.
Ma, J., & Ma, S. (1994). An $o(k^2n^2)$ algorithm to find a k-partition in a k-connected graph. *Journal of computer science and technology, 9*, 86–91.
Maravalle, M., Simeone, B., & Naldini, R. (1997). Clustering on trees. *Computational Statistics & Data Analysis, 24*, 217–234.
Matić, D. (2014). A mixed integer linear programming model and variable neighborhood search for maximally balanced connected partition problem. *Applied Mathematics and Computation, 237*, 85–97.
Matić, D., & Božić, M. (2012). Maximally balanced connected partition problem in graphs: Application in education. *The Teaching of Mathematics, XV*, 121–132.
Matić, D., & Grbić, M. (2020). Partitioning weighted metabolic networks into maximally balanced connected partitions. In *Proceedings of the 19th international symposium INFOTEH-JAHORINA (INFOTEH)* (pp. 1–6).
Miyazawa, F. K., Moura, P. F. S., Ota, M. J., & Wakabayashi, Y. (2020). Cut and flow formulations for the balanced connected k-partition problem. In M. Baïou, B. Gendron, O. Günlük, & A. R. Mahjoub (Eds.), *Combinatorial optimization*. In *Lecture Notes in Computer Science: vol. 12176* (pp. 128–139).
Nakano, S., Rahman, M., & Nishizeki, T. (1997). A linear-time algorithm for four–partitioning four-connected planar graphs. *Information Processing Letters, 62*, 315–322.
Perl, Y., & Schach, S. R. (1981). Max-min tree partitioning. *Journal of the ACM, 28*, 5–15.
Soltan, S., Yannakakis, M., & Zussman, G. (2020). Doubly balanced connected graph partitioning. *ACM Transactions on Algorithms, 16*.
Wilson, D. B. (1996). Generating random spanning trees more quickly than the cover time. In *Proceedings of the twenty-eighth annual ACM symposium on theory of computing STOC '96* (pp. 296–303).
Wolter, K. (2006). *Implementation of cutting plane separators for mixed integer programs*. Master's thesis Technische Universität Berlin.
Wu, B. Y. (2012). Fully polynomial-time approximation schemes for the max-min connected partition problem on interval graphs. *Discrete Mathematics, Algorithms and Applications, 04*, 1250005.
Zhou, X., Wang, H., Ding, B., Hu, T., & Shang, S. (2019). Balanced connected task allocations for multi-robot systems: An exact flow-based integer program and an approximate tree-based genetic algorithm. *Expert Systems with Applications, 116*, 10–20.