Computers & Operations Research 184 (2025) 107224

Contents lists available at ScienceDirect

Computers and Operations Research

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

An exact algorithm for the Minimum Gap Graph Partitioning Problem

Maurizio Bruglieri [a,*], Gianluca Consiglio [b], Roberto Cordone [b]

[a] Department of Design, Politecnico di Milano, Italy
[b] Computer Science Department, Università degli Studi di Milano, Italy

ARTICLE INFO

Keywords:
Graph partitioning
Branch-and-bound
Lagrangian relaxation
Set covering
Reduction procedures

ABSTRACT

The *Minimum Gap Graph Partitioning Problem* requires to divide a vertex-weighted undirected graph into a given number of vertex-disjoint connected components, such that the sum of the maximum weight differences over all components is minimized. Based on an extended *Integer Linear Programming* formulation with an exponential number of binary variables, we propose two relaxations that exploit the properties of the objective function so as to restrict the search for the optimal solution to a polynomial subset of variables. We also introduce a branching scheme that maintains this nice property in all subproblems for both relaxations. This allows to replace the pricing mechanism, typically adopted to manage extended formulations, with a branching mechanism on a polynomial number of candidate variables. The experimental results show that both approaches can solve to optimality instances up to 300 vertices, unless very sparse and with a large number of subgraphs, and determine tight optimality gaps on the unsolved ones, if supported by an effective metaheuristic.

1. Introduction

Given an undirected connected graph $G = (V, E)$, with $|V| = n$ and a weight vector w_v defined on the vertices $v \in V$, the *Minimum Gap Graph Partitioning Problem* (*MGGPP*) consists in partitioning G into a given number $p < n$ of vertex-disjoint connected subgraphs $G_r = (V_r, E_r)$. Generally, we impose the constraint that the subgraphs are *non-degenerate*, i.e. each one contains at least two vertices. Each subgraph has a *gap*, defined as the difference between the maximum and minimum weights of the vertices that belong to it, and the aim of the problem is to minimize the sum of all gaps, $z = \sum_{r=1}^{p} \gamma(V_r)$, with $\gamma(V_r) = \max_{v \in V_r} w_v - \min_{v \in V_r} w_v$.

The *MGGPP* was introduced in Bruglieri and Cordone (2016) motivated by applications in the sectorization of water distribution networks and the leveling of farmlands in agriculture. Concerning the former, in fact, it is a widespread approach to partition networks into components called District Metered Areas (DMAs) (Paola et al., 2014). A hydraulic network can be naturally modeled by a graph, where edges represent pipes and vertices represent junctions, each one characterized by a ground elevation. Reducing elevation differences within the subgraphs that represent districts is a desirable property. In fact, it allows to localize leakages and burst pipes more quickly and accurately, by monitoring the input and the output discharges for each district, and to achieve a stabler management of pressure that increases asset lifespans and simplifies component maintenance (Bui et al., 2020). As for farm management, dividing a piece of land into parcels with small slope reduces the effort required to flatten each parcel by earthworks. As well, limiting the variance of a suitable soil property within each parcel allows to design site-specific crop rotation strategies (Albornoz et al., 2020). In this model, the vertices of the graph correspond to sampled locations, the weights to the relevant property, and the edges link adjacent locations. Both applications concern long-term planning, for which an exact approach can in principle provide advantages over a faster heuristic one.

Since heuristic approaches are already available, in this work, we pursue for the first time the exact solution of the problem. For instances too large to be solved to optimality, we aim at least to compute tight lower bounds, in order to assess the quality of the available heuristic approaches. Our main contribution is a branch-and-bound scheme that exploits an extended formulation of the problem, with an exponential number of variables associated with the potential subgraphs. The standard approach to solve such problems (Barnhart et al., 1988) builds a restricted master problem with a limited set of variables, relaxes the integrality constraint, solves the continuous relaxation and iteratively generates new promising variables by solving an auxiliary pricing problem. This amounts to identifying an optimal connected subgraph with respect to an objective function that depends on the optimal values of the dual variables in the restricted master problem. The process iteratively adds the new variables to the restricted master problem as long as the pricing problem provides them. In this work,

* Corresponding author.
E-mail addresses: maurizio.bruglieri@polimi.it (M. Bruglieri), gianluca.consiglio@studenti.unimi.it (G. Consiglio), roberto.cordone@unimi.it (R. Cordone).

https://doi.org/10.1016/j.cor.2025.107224
Received 13 August 2024; Received in revised form 24 July 2025; Accepted 25 July 2025
Available online 7 August 2025

we show that the particular nature of the objective function allows to relax the master problem in such a way that only $O(n^2)$ variables are actually required. The continuous relaxation, the pricing problem and the iterative process can therefore be fully avoided. On the other hand, the relaxed master problem remains an integer programming problem and its optimum provides a lower bound for the *MGGPP*. We introduce two alternative relaxations and a branching scheme that guarantees optimality for the *MGGPP* keeping the quadratic limit on the number of relevant variables in all subproblems.

The first relaxation combines a cardinality constraint, covering and packing constraints, and is solved with a general-purpose Integer Linear Programming (*ILP*) solver. Since it is quite similar to the original problem, except for the number of variables, it could likely take a long time to solve. An alternative relaxation includes only covering constraints. The cardinality constraint is here managed in a Lagrangian fashion, exploiting the concavity of the Lagrangian dual in order to determine the optimal value of the multiplier in a small number of iterations. Each iteration requires to solve a *Set Covering Problem* (*SCP*), that is $\mathcal{NP}$-hard, but can be managed with reasonable efficiency for the sizes here considered.

In both cases, we also introduce reduction procedures and logical tests that allow: (i) to prove the infeasibility of the current subproblem; (ii) to remove variables of the master problem that cannot belong to any feasible solution; (iii) to propagate the branching constraints on the subgraphs represented by the current variables.

The remainder of the paper is organized as follows. Section 2 surveys the related literature. Section 3 introduces the *ILP* extended formulation, the first relaxation and the property that severely restricts the set of relevant variables. Section 4 presents the second relaxation and discusses the computation of its Lagrangian multiplier. Section 5 provides the branch-and-bound scheme that exploits (with minor differences) the two relaxations. In particular, it describes the upper bounding heuristics, the branching mechanism and the reduction procedures. Finally, Section 6 tests two ways to compute the Lagrangian lower bound. Then, it compares this bound with the *ILP*-based one and with the currently available combinatorial bound. It reports the experimental results of the two exact approaches on the benchmark instances available from the literature, both with each other and with the state-of-the-art heuristic approaches. Moreover, it discusses the role of the more refined heuristic in tackling the hardest instances. Finally, Section 7 draws some conclusions.

2. Literature

The *MGGPP* falls within the wide field of graph partitioning problems (Wu and Hou, 2023; Çatalyürek et al., 2023; Ayall et al., 2022; Bader et al., 2013; Bichot and Siarry, 2013). It is characterized by a very particular objective function. Indeed, most graph partitioning problems, e.g., the classical *Minimum k-Cut Problem* (Goldschmidt and Hochbaum, 1988; Krauthgamer et al., 2009), optimize the cost of the edges linking different subgraphs of the partition. By contrast, the *MGGPP* does not consider edge costs but vertex weights. Therefore, we cannot use traditional techniques successfully applied to solve edge weighted graph partitioning problems, like semidefinite programming used by Goemans and Williamson (1995) to formulate a relaxation of the max-cut problem leading to an approximation algorithm. For the same reason, we cannot exploit the spectral theory of the Laplacian of the graph that has been successfully applied, for instance, to the Cheeger cut, max-cut and multi-cut problems surveyed by Chang et al. (2017). While some graph partitioning problems are also concerned with vertex weights, they usually take into account the total weight of all vertices in each subgraph, and require it to be as uniform as possible among different subgraphs. This is obtained either by minimizing the difference between the total weights of different subgraphs (Chlebikova, 1996) or by imposing bounds on each of them (Ito et al., 2012, 2007). In Ito et al. (2012), a general vertex-weighted graph must be partitioned into connected subgraphs, each one with total weight in a given range, and the first polynomial-time solution algorithm for arbitrary trees is proposed.

The *MGGPP* was introduced in Bruglieri and Cordone (2016), with a focus on its computational complexity. While in general the decision version of the problem is $\mathcal{NP}$-complete, polynomial algorithms have been designed in Bruglieri et al. (2021) for special trees and in Cordone et al. (2022) for general ones. Metaheuristic algorithms based on the *Adaptive Large Neighbourhood Search* framework and on the *Tabu Search* with a multi-level aggregation mechanism have been investigated in Bruglieri and Cordone (2021). They allow to solve instances up to several thousand vertices, but provide no lower bound, and therefore no strong estimate of the quality of these solutions. An easy and fast combinatorial bound can be computed neglecting the connectivity constraint on the subgraphs and applying a greedy algorithm proposed in Bruglieri and Cordone (2016) for the case of complete graphs. However, this bound is tight only for rather dense graphs.

3. A mathematical programming formulation and a covering-packing relaxation

Let $\mathcal{V}$ be the collection of all vertex subsets that induce non-degenerate connected subgraphs on graph G, i.e., each one containing at least two vertices. For each subset $S \in \mathcal{V}$, we denote by $\gamma(S)$ the corresponding gap. Finally, we set coefficient $a_{vS} = 1$ if vertex v belongs to subset S, $a_{vS} = 0$ otherwise. The *MGGPP* admits the following *ILP* formulation, in which binary variable $x_S = 1$ indicates that the solution includes subset S and $x_S = 0$ that it does not.

$$\min z(x) = \sum_{S \in \mathcal{V}} \gamma(S)\, x_S \tag{1a}$$

subject to:

$$\sum_{S \in \mathcal{V}} x_S = p \tag{1b}$$

$$\sum_{S \in \mathcal{V}} a_{vS}\, x_S = 1 \qquad v \in V \tag{1c}$$

$$x_S \in \{0,1\} \qquad S \in \mathcal{V} \tag{1d}$$

The objective function (1a) minimizes the total gap. The cardinality constraint (1b) guarantees to obtain the desired number of subgraphs. The partitioning constraints (1c) ensure that each vertex belongs to one and only one subgraph. Constraints (1d) define the domain of the binary variables x_S.

The number of binary variables of this formulation is exponential, since the non-degenerate connected subgraphs are $O(2^n)$. Thus, the formulation is impractical if the graph exceeds a rather small size. However, we can derive a relaxation of the problem for which only a very limited subset of variables need to be taken into account. The following definitions characterize these variables.

Definition 1. Given a permutation of the vertices of V such that the weights are nondecreasing ($u \prec v \Rightarrow w_u \leq w_v$), we define V_{uv} the subset of vertices ranging from u to v and $G_{uv} = G[V_{uv}]$ the subgraph of G induced by V_{uv}.

Such subgraphs are non-degenerate if $u \prec v$, but in general are non-connected. Notice that the permutation adopted imposes an arbitrary order on vertices of equal weight. This might have an impact on the behavior of the algorithm, that is secondary as long as the number of vertices of equal weight is limited.

Definition 2. For each pair of vertices $u, v \in V$ with $u \prec v$ that are connected in G_{uv}, we define S_{uv} the maximal connected component of G_{uv} that contains both u and v and $\gamma_{uv} = w_v - w_u$ the corresponding gap. Finally, we define $\mathcal{V}'$ the collection of all the sets S_{uv} obtained letting u and v vary in V in all possible ways so that $u \prec v$ and the two vertices are connected in G_{uv}.

Notice that these subsets induce non-degenerate connected subgraphs by definition, and therefore $\mathcal{V}' \subseteq \mathcal{V}$. Moreover, the number of these subsets is $\leq n(n-1)/2$, because not all pairs of vertices $u, v \in V$ with $u \prec v$ might be connected in G_{uv}.

Definition 3. Given a permutation of the vertices of V such that the weights are nondecreasing, a vertex $v \in V$ and a subset $S \in \mathcal{V}$, let the binary coefficient $\hat{a}_{vS}$ be equal to 1 when v is the first or the last vertex of S according to the permutation, and equal to 0 otherwise.

Notice that $\hat{a}_{vS} \leq a_{vS}$ for all $v \in V$ and $S \in \mathcal{V}$.

Now, we relax the *MGGPP* by replacing each partitioning constraint (1c) with two weaker ones:

- a covering constraint that requires to select at least a subgraph including each vertex, with the same coefficients of the original constraint:

$$\sum_{S \in \mathcal{V}} a_{vS} x_S \geq 1, \tag{1c'}$$

- a packing constraint that requires to select at most a subgraph with a given vertex as first or last one, with reduced coefficients:

$$\sum_{S \in \mathcal{V}} \hat{a}_{vS} x_S \leq 1. \tag{1c"}$$

This leads to the following *ILP* formulation:

$$\min z(x) = \sum_{S \in \mathcal{V}} \gamma(S)\, x_S \tag{2a}$$

subject to:

$$\sum_{S \in \mathcal{V}} x_S = p \tag{2b}$$

$$\sum_{S \in \mathcal{V}} a_{vS}\, x_S \geq 1 \qquad v \in V \tag{2c}$$

$$\sum_{S \in \mathcal{V}} \hat{a}_{vS}\, x_S \leq 1 \qquad v \in V \tag{2d}$$

$$x_S \in \{0,1\} \qquad S \in \mathcal{V} \tag{2e}$$

Since all the feasible solutions of Formulation (1) are also feasible for (2), but the latter can admit additional solutions, the optimal value of (2) provides a lower bound on the optimum of (1). Formulation (2), however, has the following interesting property.

Proposition 1. *For each feasible solution of Formulation (2) there exists an equivalent feasible solution composed only by subsets in $\mathcal{V}'$ (see Definition 2).*

Proof. If a solution of Formulation (2) includes a subset $S \notin \mathcal{V}'$, we compute its vertices of minimum and maximum weight, respectively u and v, i.e, the first and the last one according to the given permutation. By definition, subset S is non-degenerate and its induced subgraph is connected. Therefore, $u \prec v$ and there certainly exists a maximal connected subset in G_{uv} that includes both u and v. This subset is S_{uv} by definition, and $S \subseteq S_{uv}$. Replacing S with S_{uv} in the solution, this remains feasible, because the cardinality does not change, the covering constraints are still satisfied and the packing constraints are unchanged ($\hat{a}_{rS_{uv}} = \hat{a}_{rS}$ for all $r \in V$). Moreover, S and S_{uv} have identical gaps: $\gamma_{S_{uv}} = \gamma_S$. Hence, the two solutions have the same value. ■

The consequence of Proposition 1 is that, unlike Formulation (1), Formulation (2) can be solved considering only maximal subsets. For the sake of simplicity, we indicate by x_{uv} the binary variable $x_{S_{uv}}$ and obtain Formulation (3), that is equivalent to (2), but has only $O(n^2)$ variables, instead of $O(2^n)$.

$$\min z(x) = \sum_{S_{uv} \in \mathcal{V}'} \gamma_{uv}\, x_{uv} \tag{3a}$$

subject to:

$$\sum_{S_{uv} \in \mathcal{V}'} x_{uv} = p \tag{3b}$$

$$\sum_{S_{uv} \in \mathcal{V}'} a_{rS_{uv}}\, x_{uv} \geq 1 \qquad r \in V \tag{3c}$$

$$\sum_{S_{uv} \in \mathcal{V}'} \hat{a}_{rS_{uv}}\, x_{uv} \leq 1 \qquad r \in V \tag{3d}$$

$$x_{uv} \in \{0,1\} \qquad u, v \in V : S_{uv} \in \mathcal{V}' \tag{3e}$$

Solving Formulation (3) to optimality provides a lower bound on Formulation (1). Of course, any lower bound on its optimum is also a lower bound for the *MGGPP*. Formulation (3) is a Set Covering problem with additional packing and cardinality constraints, and therefore $\mathcal{NP}$-hard. To the best of our knowledge, there is a single paper on the mixed set covering, packing and partitioning problem (Kuo and Leung, 2016), that introduces generalized mixed odd hole inequalities to tighten its natural formulation. A few papers have investigated approximation algorithms for its continuous relaxation, such as Boob et al. (2019). None of these papers combines packing and covering with a cardinality constraint.

4. A Lagrangian covering relaxation

In the cases in which solving Formulation (3) takes too much time, a natural idea is to investigate weaker relaxations. In particular, we removed the packing constraints (3d) and relaxed the cardinality constraint (3b) in a Lagrangian fashion. This corresponds to removing the constraint and adding its violation, multiplied by a real multiplier λ, to the objective function as a penalty term. This yields Formulation (4):

$$\min z_\lambda(x) = \sum_{S_{uv} \in \mathcal{V}'} \gamma_{uv}\, x_{uv} + \lambda \left(\sum_{S_{uv} \in \mathcal{V}'} x_{uv} - p \right) = \sum_{S_{uv} \in \mathcal{V}'} \left(\gamma_{uv} + \lambda\right) x_{uv} - \lambda p \tag{4a}$$

subject to:

$$\sum_{S_{uv} \in \mathcal{V}'} a_{rS_{uv}}\, x_{uv} \geq 1 \qquad r \in V \tag{4b}$$

$$x_{uv} \in \{0,1\} \qquad u, v \in V : S_{uv} \in \mathcal{V}' \tag{4c}$$

that is an instance of the *SCP* where the cost of each column is modified by a uniform term λ. In the following, we denote the optimum value of Formulation (4) for each given value of parameter λ as $g(\lambda) = \min_x z_\lambda(x)$, reminding that it always provides a lower bound for the original *MGGPP* instance.

Of course, we are interested in the computation of the value that produces the highest lower bound, that is, the one closest to the optimum. Section 4.1 discusses two iterative ways to do that, and shows that both require a limited number of iterations.

The choice of this specific relaxation is due to the fact that, while the *SCP* is $\mathcal{NP}$-complete in its decision version, it can be solved in a reasonable computational time in practice (Caprara et al., 2000). Once again, if the time required is excessive, a lower bound can be used instead. Moreover, we shall see that approximating the best value of multiplier λ takes only a limited number of iterations. Finally, the literature provides also dedicated algorithms for the *SCP* (Caprara et al., 1999; Ceria et al., 1998), which would allow the practical advantage to completely avoid the use of a commercial solver, in the cases in which this is not a viable option.

4.1. Computation of the optimal Lagrangian multiplier

The Lagrangian dual function $g(\lambda)$ is given by the lower envelope of a family of linear functions of λ (see Fig. 1), associated with optimal solutions of the Lagrangian subproblem (Guignard, 2003). It is piecewise linear, and its slope for a given λ is the violation of the relaxed

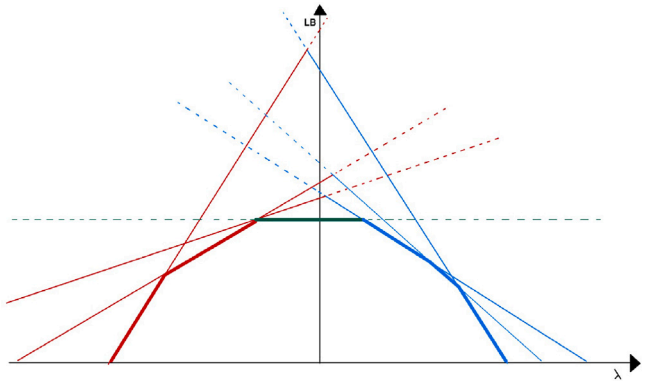

Fig. 1. Behavior of the Lagrangian dual function $g(\lambda)$: each line provides the value of the objective function for a solution of the Lagrangian problem that is optimal for a value of multiplier λ; the red lines with positive slope refer to solutions with less than p nonzero variables, the blue lines with negative slope to solutions with more than p nonzero variables, the green horizontal line to the best solution with exactly p nonzero variables.

cardinality constraint in the optimal solution of the corresponding *SCP* (4). When λ is small enough, many variables x_{uv} are equal to 1, and the slope is strictly positive. When λ is large enough, only few variables x_{uv} are equal to 1, yielding a strictly negative slope. Therefore, $g(\lambda)$ is concave, first increasing and then decreasing. Its profile is nondifferentiable, with corner points where the optimal solution is not unique. Its maximum, achieved either in a corner point or along a horizontal plateau, provides the tighest Lagrangian bound on the *MGGPP*.

A classical way to iteratively approximate the optimal multipliers in a Lagrangian relaxation is the subgradient method (Shor, 1985). In the case of a single multiplier, however, the properties of the dual Lagrangian problem allow one to first find and then progressively refine an interval that certainly contains an optimal value λ^* of this multiplier. In this way, they provide a guarantee on the maximum number of iterations of the procedure.

Remark 1. An optimal value λ^* for the multiplier of the Lagrangian problem (4) falls between $-\gamma(V)-1$ and $\gamma(V)+1$.

Proof. Since graph G is connected by definition, the minimum weight vertex u and the maximum weight vertex v of the whole graph G are connected and the solution with $x_{uv}=1$ and all other variables set to zero is always feasible for Formulation (4). If $\lambda=\gamma(V)+1$, this solution is optimal: neglecting the constant factor $-\lambda p$ for the sake of simplicity, its cost is $\gamma_{uv}+\lambda=2\gamma(V)+1$, while any solution with more than one positive variable costs at least $2\gamma(V)+2$; the solution with all variables set to zero is infeasible. If $\lambda=-\gamma(V)-1$, the optimal solution to Formulation (4) is $x_{uv}=1$ for all variables. In fact, its cost is $\sum_{S_{uv}\in\mathcal{V}'}(\gamma_{uv}-\gamma(V)-1)$, whereas setting any variable x_{uv} to zero would increase the cost by $-\gamma_{uv}+\gamma(V)+1>0$, making it larger. ■

In the following, we discuss two iterative ways to refine the interval in order to approximate λ^*. The former approach provides a maximum number of iterations to reduce the interval width under a given threshold, the latter a maximum number of iterations to reach an optimal value.

Binary search. The first approach we consider to estimate an optimal value λ^* is simply a binary search in $[-\gamma(V)-1,\gamma(V)+1]$. We start solving the Lagrangian relaxation (4) with $\lambda=0$. This corresponds to the *SCP* without the cardinality constraint. We obtain a relaxed solution with value $g(0)$, that is a lower bound on the optimum of (3). If the number of nonzero variables in this solution is strictly greater than p, the current value of λ is too small; then, we remove the first half of the range. Vice versa, if the cardinality is strictly smaller than p, we remove the second half. In both cases, we set λ to the middle point of the remaining interval. If the nonzero variables are exactly p, the solution is optimal for (3), and we can terminate. The process stops, as well, when the width of the interval falls below a given threshold ϵ. Consequently, the number of iterations is at most $\lceil\log_2(2(\gamma(V)+1)/\epsilon)\rceil$.

Two-line approximation. The second approach takes into account the specific value of the slope, instead of just its sign. Given two values of λ that determine lines of opposite slopes, we generate the next value, $\bar{\lambda}$, by intersecting the two lines. Solving the Lagrangian relaxation for $\bar{\lambda}$ yields two possible cases. The most frequent one is the generation of a third line with a slope flatter than the current line with the same sign. Then, the new line replaces the current one and the procedure is repeated. The second case generates again one of the two current lines or possibly a third one, corresponding to the fact that $\bar{\lambda}$ is the optimal break-point and the relaxed problem has two or more optimal solutions. Then, the process terminates. Since each column covers at least two rows (due to the non-degeneracy constraint), the optimal Lagrangian solution can include from 1 to $\lfloor n/2\rfloor$ columns and the slope is between $1-p$ and $\lfloor n/2\rfloor-p$. Hence, as at least one of the slopes strictly decreases in absolute value, the number of iterations of the process is at most $|1-p|+|\lfloor n/2\rfloor-p|-1=\lfloor n/2\rfloor-2$.

Notice that there are instances where the Lagrangian lower bound can be arbitrarily bad. The following remark gives an example.

Remark 2. The lower bound provided by the optimal value of the Lagrangian dual problem associated with Formulation (4) may be arbitrarily bad with respect to the optimum of the *MGGPP*.

Proof. Consider any positive integer value for p and set $n=3p-1$. Let graph G consist of a path of $2(p-1)$ vertices, $(v_1,\dots,v_{2p-2})$, and a star with root v_{2p-1} and $p-1$ leaves, $\{v_{2p},\dots,v_{3p-1}\}$. The path and the root of the star are connected by edge (v_{2p-2},v_{2p-1}). The vertices of the path with even index have weight $w_{v_{2r}}=1$, with $r\in\{1,\dots,p-1\}$; all the other vertices have zero weight.

Since each subgraph must contain at least two vertices, the star must be fully included in a single subgraph. This implies that the only feasible solution consists of $p-1$ subgraphs consisting of pairs of adjacent vertices along the path and a subgraph coinciding with the star: $V_r=\{v_{2i-1},v_{2i}\}$ for $i\in\{1,\dots,p-1\}$ and $V_p=\{v_{2p-1},\dots,v_{3p-1}\}$. Thus, the optimal solution costs $(p-1)$.

On the other hand, the Lagrangian relaxation admits for any value of the multiplier λ a feasible solution consisting in a subgraph that includes all the vertices of the path and $p-1$ subgraphs obtained selecting the root of the star and one of its leaves. The cost of this solution is $(1+\lambda)+(p-1)(0+\lambda)-\lambda p=1$. Since this solution can be nonoptimal, the Lagrangian lower bound is $g(\lambda)\le 1$ and, therefore, is at least $p-1$ times weaker than the optimum. ■

5. Exact branch-and-bound approaches

The two relaxations presented above can be embedded in a branch-and-bound scheme. Since the two approaches are rather similar, we here describe them together, briefly discussing their minor differences.

The computation of the lower bound at each node of the branching tree has already been described in Sections 3 and 4, respectively. We adopt the best-first visit strategy, that is we always consider the open node with the minimum lower bound. In the following, we describe the heuristic procedures applied to compute upper bounds during the computation, the branching mechanism and the logical tests used to prune the current node or to strengthen its branching constraints.

5.1. Upper bounding procedures

Both relaxations have an optimal solution that provides a collection of subgraphs covering all vertices of the graph. This can be used as a

starting point to build heuristic solutions to the *MGGPP*. In general, the relaxed solutions are not feasible for the following two reasons: (i) the number of subgraphs can be different from p (only for the Lagrangian relaxation), (ii) some vertices can belong to more than one subgraph (for both relaxations, but more frequently for the Lagrangian one).

Repair procedure. At each branching node, we aim to repair the solution provided by the relaxation. The procedure (see Algorithm 1) first identifies a core assignment of vertices to subgraphs that respects the packing, cardinality and non-degeneracy constraints, then applies a greedy mechanism that completes the assignment of the other vertices. The first phase removes from the solution all vertices that are assigned to more than one subgraph. After the removal, some subgraphs could be disconnected. For each of them, we keep only the connected component of maximum cardinality, removing the other vertices. If also the largest connected component is an isolated vertex, we delete the whole subgraph. Finally, if the number of subgraphs obtained with this procedure is strictly larger than p, we remove the smaller ones until we obtain exactly p. The second phase of the repair procedure reassigns all removed vertices. If the number of subgraphs is strictly smaller than p, it iteratively finds the edge induced by the unassigned vertices, $(u, v) \in E(U)$, with minimum gap $|w_u - w_v|$ and assigns its extreme vertices to a new subgraph, until the required number is obtained. This step fails when the unassigned vertex pairs are insufficient (e.g., the Lagrangian solution could be a partition with fewer than p components). In that case, the heuristic terminates. Otherwise, we obtain exactly p subgraphs. Then, we iteratively select one of the unassigned vertices and add it to one of its adjacent subgraphs so as to minimize the overall gap. When all vertices have been assigned, the heuristic terminates providing a feasible solution.

Local search procedure. If a feasible solution is obtained, the algorithm tries to improve it by applying a simple standard local search approach. The neighborhood is defined as the set of all feasible solutions obtained moving a single vertex from the current subgraph to an adjacent one. In order to guarantee feasibility, the original subgraph must include at least three vertices and must not be disconnected by the removal. The procedure considers all feasible moves and performs the best one, but only if it strictly improves the value of the objective function. When this is no longer possible, it terminates.

Destroy and repair procedure. As discussed in Section 6.4, the repair procedure, while reasonably effective for most of the instances, is unable to find good quality solutions for the sparsest ones, even if powered by local search. We have therefore introduced an additional more refined procedure, which will be run only occasionally for the sake of efficiency.

This procedure is a simplified version of the Adaptive Large Neighbourhood Search approach proposed in Bruglieri and Cordone (2021). It follows the *destroy-and-repair* framework: first, remove some vertices from the current solution applying a destroy procedure randomly chosen in a given set of removal heuristics with a uniform distribution; then, assign the removed vertices to the subgraphs applying a randomized greedy heuristic. The destroy procedures adopted are: (i) a *random* procedure that removes a fraction q of the n vertices, recomputes the connected components for each subgraph affected by the removal and keeps only the component of maximum cardinality (and minimum gap, in case of ties), if it has at least two vertices; (ii) a *worst cluster* procedure that completely removes the subgraph with the maximum gap; (iii) a *nearly worst cluster* procedure that completely removes either the cluster with the second maximum gap or that with the third maximum gap, with 50% probability; (iv) a *Shaw removal* procedure (Shaw, 1998), that removes a fraction q of the n vertices, selected in order to be somehow related, as follows. For any two vertices u and v, let $R(u, v)$ be a convex combination of their weight difference, $|w_u - w_v|$, and of the number of edges, l_{uv}, of the minimum cardinality path between them:

$$R(u,v) = \omega \frac{|w_u - w_v|}{\Gamma} + (1-\omega)\frac{l_{uv}}{L},$$

where $\omega \in [0,1]$ tunes the relative strength of the two components and the normalizing coefficients $\Gamma = \max_{u,v\in V} |w_u - w_v|$, and L, diameter of the graph (i.e., $L = \max_{u,v\in V} l_{uv}$), guarantee that they have comparable

Algorithm 1 Pseudocode for the basic repair procedure

procedure REPAIR($x, G, w, \mathcal{V}', a$)
  $c := 0$
  $\mathcal{C} := \emptyset$
  for $S_{uv} \in \mathcal{V}' : x_{uv} = 1$ **do** ▷ Find the subsets used by relaxed solution x
    $c := c + 1$
    $C_c := S_{uv}$
    $\mathcal{C} := \mathcal{C} \cup \{C_c\}$
  end for
  for $C_c \in \mathcal{C}$ **do** ▷ Remove the vertices belonging to more than one subset
    $C_c := C_c \setminus \{w \in V : \sum_{S_{uv}\in\mathcal{V}'} a_{wS_{uv}} x_{uv} > 1\}$
    $C_c :=$ LargestConnComponent($G[C_c]$) ▷ Reduce C_c to its largest component
  end for
  while $|\mathcal{C}| > p$ **do** ▷ Keep only the largest p components
    $\mathcal{C} := \mathcal{C} \setminus \arg\min_c |C_c|$
    $c := c - 1$
  end while
  $U := V \setminus \bigcup_c C_c$ ▷ Find the unassigned vertices
  while $|\mathcal{C}| < p$ **and** $E(U) \neq \emptyset$ **do** ▷ Build the missing components
    $(u^*, v^*) := \arg\min_{u,v\in E(U)} |w_u - w_v|$ ▷ with unassigned adjacent vertex pairs
    $c := c + 1$
    $C_c := \{u^*, v^*\}$
    $\mathcal{C} := \mathcal{C} \cup \{C_c\}$
    $U := U \setminus \{u^*, v^*\}$
  end while
  if $E(U) = \emptyset$ **then**
    return FAIL
  end if
  while $U \subset V$ **do**
    $(u^*, c^*) :=$ FindBestAssignment($U, \mathcal{C}, G, w$) ▷ Find the best vertex assignment
    $C_{c^*} := C_{c^*} \cup \{u^*\}$
  end while
  return $\mathcal{C}$
end procedure

Algorithm 2 Pseudocode for the local search procedure

procedure LOCALSEARCH($\mathcal{C}$)
  stop := false
  while not stop **do**
    $z^* := +\infty$
    for $u \in V$ and $C_c \in \mathcal{C} : u \notin C_c$ **do**
      $\mathcal{C}' :=$ Move($u, c, \mathcal{C}$)
      if $z(\mathcal{C}') < z^*$ **then**
        $u := u^*$
        $c := c^*$
        $z^* := z(\mathcal{C}')$
      end if
    end for
    if $z^* \geq z(\mathcal{C})$ **then**
      stop := true
    else
      $\mathcal{C} :=$ Move($u^*, c^*, \mathcal{C}$)
    end if
  end while
  return $\mathcal{C}$
end procedure

values. We consider three versions of this procedure, setting $\omega = 0$, $\omega = 0.5$, and $\omega = 1$, respectively. They start by removing a random vertex $\bar{v}$ and proceed iteratively. Each time, they randomly choose one of the previously removed vertices, v, sort the n' remaining vertices by nondecreasing values of $R(u, v)$ and remove the vertex in position $\lfloor y^{\rho} n' \rfloor$, where y is a random number extracted from $[0, 1)$ with a uniform distribution. This results in a biased random extraction tuned by coefficient $\rho \geq 1$: if $\rho = 1$, the selection is uniform; as ρ increases, y^{ρ} tends to 0 and the selection focuses more and more on the vertex u with minimum $R(u, v)$. The basic idea is that vertices with similar weights and connected by short paths are easier to rearrange differently; by contrast, when the vertices removed are widely dispersed in the graph or have very different weights, the repair procedure will probably reinsert them in their original subgraphs and return the starting solution.

The repair procedure computes for each of the h removed vertices that can be feasibly added to one of the current subgraphs the resulting total gap. Then, it randomly selects (with a uniform distribution) one of the k best vertices, where k is set to a random value in $\{1, \lceil h/3 \rceil, \lceil 2h/3 \rceil, h\}$.

The algorithm repeatedly applies one of the destroy procedures and the repair procedure to the current best known solution, updating it whenever a better one is found. It terminates after a given time T_{LNS} and feeds the result to the local search heuristic for a possible further improvement.

This algorithm simplifies the one proposed in Bruglieri and Cordone (2021) because it only accepts improving solutions and it does not adapt the probabilities of its various options. In fact, it will be applied only for short runs as an intensifying procedure, so that neither learning nor diversification will play a significant role. On the other hand, the destroy-and-repair heuristic is more complicated than the repair procedure applied to the relaxed solutions and its randomized structure requires several iterations to provide useful results. Therefore, this upper bounding procedure is not run at each branching node, but only periodically, after a time T_{ref} from the beginning or from the previous call.

Algorithm 3 Pseudocode for the *LNS* procedure

```
procedure ALNS(C, RH, IH)
    C* := C;
    while the termination condition is not satisfied do
        Select RH ∈ RH                ▷ Choose a random removal heuristic
        C := RH(C*);
        Select IH ∈ IH                ▷ Choose a random insertion heuristic
        C := IH(C*);
        if z(C) < z(C*) then
            C* := C;
        end if
    end while
    return C*
end procedure
```

5.2. The branching mechanism

The branching constraints must guarantee that every feasible solution can be obtained by suitably fixing the branching variables. In the present case, then, they must allow the introduction of nonmaximal connected components, that are necessary to generate feasible solutions for the original problem. At the same time, however, they should also keep the fundamental property of Proposition 1, that is, reducing the number of relevant variables to $O(n^2)$. A simple binary branching on the x_{uv} variables does not satisfy this requirement. In fact, while setting $x_{uv} = 1$ has a very simple and natural impact on the formulation of the problem, setting $x_{uv} = 0$ implies that several nonmaximal connected components should replace S_{uv}, violating the property.

To face this problem, we adopt a more refined branching rule, that considers an auxiliary family of $O(n^2)$ binary variables y_{uv} associated with vertex pairs (u, v), where $u \preceq v$. In detail, $y_{uv} = 1$ imposes that u is the minimum weight vertex in the subgraph including vertex v (where v can coincide with u), while $y_{uv} = 0$ forbids it, that is, either imposes that u is not the minimum weight vertex for any subgraph, or that this subgraph does not include v. Notice that any feasible solution can be fully described by suitably fixing all variables y_{uv}.

Moreover, Proposition 1 extends to the case in which some y variables are set, suitably redefining the collection $\mathcal{V}'$ to account for the branching constraints. In particular, each subset S_{uv} becomes the maximal connected component of graph $G[V_{uv}]$ such that:

- it includes all vertices w with $y_{uw} = 1$;
- it excludes all vertices w with $y_{uw} = 0$.

Excluding some vertices reduces the connected component, but also possibly forbids to satisfy the definition. As well, including some vertices can be impossible. In these cases, set S_{uv} does not exist and collection $\mathcal{V}'$ must be reduced. Correspondingly, variable x_{uv} is not defined in the original problem (1), and consequently also in the covering-packing relaxation (2) and in the Lagrangian covering relaxation (4). This corresponds to implicitly imposing constraints on the y variables, even if they do not occur explicitly in these formulations. As a result, the values of the lower bounds provided can increase. The relaxations can even become infeasible, in which case, of course, also the original problem is infeasible and the branching node must be pruned.

5.3. Logical tests

Combining the constraints of the problem with the branching constraints sometimes allows to fix the value of other variables through logical implications. In the following, we list the logical tests that we have detected, labeling them with the constraint on which they are hinged. As the tests can trigger each other, we apply them cyclically until they all fail. Every time an implication leads to violating a previous variable assignment, this reveals an inconsistency in the constraints imposed and the node is pruned.

Minimum weight vertex tests. If vertex v is assigned to a subgraph whose minimum vertex weight is u, then also u must be assigned to the same subgraph:

$$y_{uv} = 1 \Rightarrow y_{uu} = 1.$$

Vice versa, if vertex u is not the minimum weight one for any subgraph, then no other vertex v can be assigned to it:

$$y_{uu} = 0 \Rightarrow y_{uv} = 0 \quad \forall v \in V.$$

Partition tests. Since each vertex must belong to exactly one subgraph, in the end $\sum_{u \preceq v} y_{uv} = 1$ for all $v \in V$. Hence:

$$y_{uv} = 1 \Rightarrow y_{u'v} = 0 \quad \forall u' \in V \setminus \{u\}$$

and

$$y_{u'v} = 0 \quad \forall u' \in V \setminus \{u\} \Rightarrow y_{uv} = 1.$$

Cardinality test. Since the solution must consist of exactly p subgraphs, as soon as p vertices are fixed as minimum weight vertices, no other vertex can be a minimum weight vertex for any subgraph. Therefore:

$$\exists U \subseteq V : |U| = p \text{ and } y_{uu} = 1 \ \forall u \in U \Rightarrow y_{uu} = 0 \ \forall u \in V \setminus U.$$

Vice versa, as soon as $n - p$ vertices are fixed as nonminimum weight vertices, every other vertex must be a minimum weight vertex for some subgraph. Therefore:

$$\exists U \subseteq V : |U| = n - p \text{ and } y_{uu} = 0 \ \forall u \in U \Rightarrow y_{uu} = 1 \ \forall u \in V \setminus U.$$

Non-degeneracy tests. Since every subgraph must include at least two vertices, if all vertices adjacent to u (that is, those in Δ_u) cannot be assigned to it, then u cannot be a minimum weight vertex for any subgraph.

$$y_{uv} = 0 \ \forall v \in \Delta_u \Rightarrow y_{uu} = 0.$$

If a vertex u is a minimum weight vertex for a subgraph and exactly one adjacent vertex v can be assigned to it, then we enforce the assignment:

$$y_{uu} = 1 \text{ and } \Delta_u \setminus \{w \in V : y_{uw} = 0\} = \{v\} \Rightarrow y_{uv} = 1.$$

Leaf test. If a vertex is a leaf ($\Delta_f = \{r\}$), it must belong to the same subgraph as its only adjacent vertex r. Therefore, the current branching constraints on y_{uf} must be applied to y_{ur} and vice versa for all $u \in V$:

$$\Delta_f = \{r\} \Rightarrow y_{uf} = y_{ur} \ \forall u \in V.$$

6. Experimental results

The experimental results here reported have been obtained coding the branch-and-bound algorithm in C language, compiling it with `gcc 7.3.1` and running it on an Intel Xeon E5-2620 2.1 GHz server with 16 GB of RAM.

6.1. Benchmark instances

We draw from the literature on the *MGGPP* (Bruglieri and Cordone, 2021) a benchmark of instances composed by three different classes: the *dense* random graphs have $m = 2n(n-1)/3$ edges, the *sparse* random graphs have $m = n(n-1)/3$ edges, and the *planar* graphs are generated by computing a greedy triangulation of n points with a uniform random distribution in a square. The weights of the vertices are integer numbers drawn from a uniform distribution in $\{1, \dots, n\}$, allowing repetitions. We consider three sizes: $n = 100$, 200, and 300. We also consider two benchmarks of real-world instances. The former includes 11 hydraulic networks provided in Wang et al. (2014), that range from 7 to 447 vertices, whose weights represent elevations. The latter includes 3 agricultural networks. These represent rectangular grids of, respectively, 616, 741 and 862 vertices, corresponding to sampled locations on fields. Their weights also represent the elevations of the corresponding points. In all these instances, the number of subgraphs p is set to three different values: $\lfloor \ln(n) \rceil$, $\left\lfloor \sqrt{n} \right\rceil$ or $\lfloor n/\ln(n) \rceil$, where symbol $\lfloor\rceil$ represents rounding a real number to the closest integer. Since in the first benchmark each combination of topology, size and number of subgraphs corresponds to 5 instances, the overall benchmark set consists of $3 \cdot 3 \cdot 3 \cdot 5 + 11 \cdot 3 + 3 \cdot 3 = 177$ instances, which are all available at https://homes.di.unimi.it/cordone/research/research.html. Table 1 provides the sources, names and features of the 14 real-world graphs, namely the number of vertices n and edges m, and the three possible required cardinalities p.

6.2. Size of the restricted formulation

The first round of experiments aims to evaluate the actual size of the restricted Formulation (3) with respect to its theoretical quadratic limit. We consider the root node, where the branching constraints do not introduce further reductions. We remind that the number of variables x_{uv} coincides with the cardinality of the collection $\mathcal{V}'$, so that the ratio

$$\rho_{\text{var}} = \frac{|\mathcal{V}'|}{n(n-1)/2}$$

measures the reduction. However, each variable also corresponds to a subset S_{uv} of connected vertices that is smaller than the theoretical subset V_{uv} of vertices with intermediate weights between u and v. We therefore introduce a second reduction index:

$$\rho_{\text{vert}} = \frac{\sum_{S_{uv} \in \mathcal{V}'} \frac{|S_{uv}|}{|V_{uv}|}}{|\mathcal{V}'|},$$

which is the average ratio of the restricted subsets with respect to the theoretical ones.

Table 2 reports the values of these indices for the benchmark instances. The first two columns provide the class of instances and the subclass of 5 instances (for the first benchmark) or the name of the single instance. The last two columns report the values of the two indices: for the first benchmark, they are averaged over the instances of the subclass; for the other ones, they are directly reported.

As expected, the two indices strongly depend on the density of the graph. The planar and real-world graphs, which are sparser, have smaller values, whereas the denser graphs often exhibit nearly no reduction with respect to the theoretical values. Index ρ_{vert} is less affected than ρ_{var}, suggesting that it is more likely for a subset S_{uv} to be completely ruled out than to be shrinked in size. This could be related to the fact that the weights are randomly generated, and adjacent vertices have uncorrelated weights, but it holds also for the

Table 1
Main features of the real-world benchmark instances.

Source	Instance	n	m	p		
				$\ln n$	$\sqrt{n}$	$n/\ln n$
Hydraulic	Two-loop	7	8	2	3	3
	Two-reservoir	12	17	2	3	5
	GoYang	23	31	3	5	7
	Blacksburg	31	35	3	6	9
	BakRyan	36	57	4	6	10
	Fossoli	37	58	4	6	10
	Richmond	48	51	4	7	12
	Pescara	71	99	4	8	17
	Modena	272	317	6	16	49
	D-town	407	459	6	20	68
	Balerma	447	454	6	21	73
Agriculture	cellplot3	616	1172	6	25	96
	cellplot2	741	1422	7	27	112
	cellplot1	862	1664	7	29	128

Table 2
Size of the restricted Formulation (3) at the root node: average ratio of the number of variables to the theoretical limit (ρ_{var}) and average ratio of the number of vertices in each subgraph to the theoretical number (ρ_{vert}).

Type	Subclass/instance	ρ_{var}	ρ_{vert}
Planar	100	34.6%	76.6%
	200	31.1%	79.2%
	300	28.8%	80.2%
Sparse	100	94.4%	99.2%
	200	96.9%	99.5%
	300	97.9%	99.7%
Dense	100	98.7%	99.9%
	200	99.3%	99.9%
	300	99.5%	100.0%
Hydraulic	Two-loop	33.3%	96.4%
	Two-reservoir	53.0%	95.5%
	GoYang	45.1%	83.0%
	Blacksburg	29.9%	79.3%
	BakRyan	41.7%	79.5%
	Fossoli	69.2%	93.5%
	Richmond	15.1%	72.3%
	Pescara	16.7%	56.0%
	Modena	40.0%	88.5%
	D-town	7.7%	42.6%
	Balerma	4.1%	55.6%
Agriculture	cellplot3	40.2%	73.9%
	cellplot2	65.4%	90.2%
	cellplot1	54.0%	85.6%

Table 3
Comparison between the binary search and the two-line approach to compute the optimal Lagrangian multiplier for the cardinality constraint relaxation on the random instances.

Type	p	n	Binary search		Two-line	
			Iter	CPU	Iter	CPU
Planar	$\ln(n)$	100	8.2	0.1	7.4	0.1
		200	5.0	0.9	1.8	0.4
		300	11.6	8.4	13.0	7.1
	$\sqrt{n}$	100	8.8	0.1	10.8	0.1
		200	10.6	1.6	12.2	1.4
		300	12.0	8.8	13.0	7.0
	$n/\ln(n)$	100	10.0	0.1	10.4	0.1
		200	10.0	1.1	11.6	1.0
		300	12.0	7.3	13.0	5.9
Sparse	$\ln(n)$	100	9.6	1.3	6.2	0.9
		200	9.8	27.6	5.4	17.3
		300	11.6	236.1	7.0	152.1
	$\sqrt{n}$	100	9.0	1.1	5.4	0.7
		200	11.0	33.3	6.4	20.3
		300	12.0	255.6	7.0	161.8
	$n/\ln(n)$	100	10.0	1.3	5.8	0.6
		200	11.0	29.9	10.6	24.2
		300	12.0	205.2	8.4	124.1
Dense	$\ln(n)$	100	9.6	1.0	7.0	0.8
		200	10.0	28.1	6.4	18.1
		300	11.2	232.5	7.6	167.0
	$\sqrt{n}$	100	10.0	1.1	6.6	0.8
		200	11.0	31.6	7.2	21.4
		300	11.2	232.9	6.6	144.4
	$n/\ln(n)$	100	10.0	1.1	5.8	0.7
		200	11.0	33.5	6.4	19.3
		300	12.0	261.7	6.2	140.9

hydraulic networks, where the vertices represent junctions and the weights physical elevations, so that adjacent vertices are more likely to have similar weights.

6.3. Computation of the Lagrangian multiplier

In this section, we compare the two approaches to compute the optimal Lagrangian multiplier described in Section 4.1: the binary search approach and the two-line approach. Table 3 reports their results at the root node, that is, on the original problem, for the random instances. Each row provides average results on the 5 instances with the same size, topology and number of subgraphs. The first three columns of the table contain this information. The following ones give the average number of iterations (Iter) and the average computational time required in seconds (CPU).

In our experiments, we have set $\epsilon = 0.2$ to make the computational times of the two methods comparable. We remind the reader that the number of iterations has a theoretical limit in both cases: $\lceil \log_2 (2(\gamma(V)+1)/\epsilon) \rceil$ for the binary search and $\lfloor n/2 \rfloor - 2$ for the two-line approach, respectively. For the binary search, the theoretical number of iterations ranges from 10 to 12, and the actual one is the same most of the time (in 114 instances out of 135). The theoretical limit for the two-line approximation is much larger (from 48 to 148), but also much looser in practice. In fact, the actual number of iterations is comparable to that of the binary search: slightly larger on the planar instances, but lower on the sparse and dense ones. On average, it is 23% smaller.

The computational time mainly derives from the effort spent at each iteration to solve the Lagrangian subproblem, that is, a *SCP* instance. The results confirm the slightly stronger efficiency of the two-line approach with respect to the binary search, with an average reduction in time of 27% that holds for all classes of instances. Of course, tuning the ϵ coefficient might improve the computational time of binary search. However, while the two-line approach terminates with the optimal multiplier, enlarging the tolerance margin for the binary search could yield a suboptimal value, weakening the lower bound. With $\epsilon = 0.2$ the two approaches most of the time provide the same bound, also because we round it up to the next integer value, exploiting the integrality of the objective function. Moreover, the fact that the actual number of iterations is close to the theoretical one suggests that decreasing the number of iterations by 1 would require to double the margin of tolerance. So, the range available for fine tuning appears to be narrow.

Another possible advantage of the two-line approximation is that the repair procedure of Section 5.1, when applied to the relaxed Lagrangian solution, retrieves a feasible solution for 81 instances out of 135, whereas in the case of binary search this happens only for 60 instances. However, we have no formal proof that repairing the solution associated with the slightly suboptimal multiplier provided by binary search necessarily yields worse results than repairing the one associated with the optimal multiplier provided by the two-line approach.

6.4. Comparison of the available relaxations

The next round of experiments aims to assess the performance of the three available relaxations (covering-packing, Lagrangian covering, and combinatorial). In particular, we want to determine whether the stronger tightness of the covering-packing relaxation counterbalances the probably longer time required to solve it with respect to the simpler Lagrangian covering relaxation, and how much they both improve on the combinatorial bound proposed in Bruglieri and Cordone (2016).

This bound is obtained assuming that the graph is complete, so that every subgraph is by definition connected. The optimum of such a relaxation can be computed sorting the vertices by nondecreasing weights and subtracting from the total gap of the graph, $\gamma(V)$, the largest $p-1$ differences between consecutive weights. In fact, the p uninterrupted subsequences of vertices thus obtained correspond to the subgraphs of an optimal solution.

Table 4 compares the lower bounds obtained on the random benchmark using the three procedures at the root node. The first three columns identify the group of 5 instances with the same features. The following column (labeled GapLB) provides the average gap $(z^* - LB)/z^*$ between the lower bound obtained by the combinatorial relaxation and the optimum. When this is unknown, we overestimate it as $(UB^* - LB)/LB^*$, where LB^* and UB^* are the best known lower and upper bound, respectively. We do not report the computational time because it is always smaller than 0.0005 s. The following pair of columns provides the gap and the average computational time in seconds (CPU) for the Lagrangian relaxation with the two-line method. The results for the binary search method would be nearly the same. The last pair of columns provide the same information for the covering-packing relaxation.

The combinatorial bound is pretty tight on the sparse and dense instances with a small number of subgraphs. In 32 cases out of 135, it is even equal to the optimum (which implies not necessarily that the relaxed solution is feasible, but that an equivalent optimal solution exists). However, in the planar instances, in particular when p is large, the quality of the lower bound quickly degrades. Moreover, combining this bound with a branching mechanism is far from trivial, as any meaningful branching constraint soon destroys the completeness assumption. The Lagrangian bound is much tighter, even on the planar instances, with 91 cases in which it coincides with the optimum. However, it takes a much longer time, in particular for the larger and denser instances, where it goes up to a few minutes. This derives from the size of the auxiliary *SCP*, that has a linear number of rows and a quadratic number of columns with respect to the size of the graph. The covering-packing relaxation shows a similar dependence on the topology, number of subgraphs and vertices, but the increase in computational time is less

Table 4
Comparison between the combinatorial, the Lagrangian covering and the covering-packing relaxation of the *MGGPP* on the random instances.

Type	p	n	Comb.	Lagr.		Cover.-pack.	
			GapLB	GapLB	CPU	GapLB	CPU
Planar	$\ln(n)$	100	17.9%	0.0%	0.1	0.0%	0.3
		200	10.4%	0.4%	0.4	0.1%	2.1
		300	8.8%	0.0%	7.1	0.0%	10.2
	$\sqrt{n}$	100	35.2%	1.3%	0.1	0.0%	0.3
		200	30.4%	1.0%	1.4	0.1%	2.1
		300	26.3%	0.3%	7.0	0.2%	10.2
	$n/\ln(n)$	100	69.8%	12.6%	0.1	1.9%	0.3
		200	73.3%	10.4%	1.0	2.3%	2.2
		300	77.5%	14.6%	5.9	5.1%	10.8
Sparse	$\ln(n)$	100	0.9%	0.0%	0.9	0.0%	0.6
		200	0.2%	0.3%	17.3	0.0%	5.0
		300	0.1%	0.1%	152.1	0.0%	15.1
	$\sqrt{n}$	100	3.8%	0.3%	0.7	0.0%	0.4
		200	2.0%	0.3%	20.3	0.0%	5.0
		300	1.8%	0.2%	161.8	0.0%	14.6
	$n/\ln(n)$	100	27.6%	0.0%	0.6	0.0%	0.4
		200	24.3%	1.2%	24.2	0.0%	5.1
		300	20.9%	0.4%	124.1	0.1%	14.9
Dense	$\ln(n)$	100	0.5%	0.0%	0.8	0.0%	0.5
		200	0.0%	0.0%	18.1	0.0%	6.1
		300	0.1%	0.0%	167.0	0.0%	23.7
	$\sqrt{n}$	100	2.0%	0.0%	0.8	0.0%	0.5
		200	0.3%	0.1%	21.4	0.0%	6.1
		300	0.3%	0.0%	144.4	0.0%	23.8
	$n/\ln(n)$	100	10.9%	0.0%	0.7	0.0%	0.5
		200	4.3%	0.4%	19.3	0.0%	6.2
		300	3.9%	0.0%	140.9	0.0%	23.7

sharp: while it takes longer than the Lagrangian relaxation on the planar instances, it is much faster on the dense ones. This was unexpected, given that the structure of the problem is more complicated. Possibly, the additional packing constraints allow to manage more efficiently the decision variables. Moreover, the quality of the bound is also stronger, with 116 optimal values out of 135.

6.5. Results of the basic branch-and-bound algorithm

In this section, we combine the lower bound procedures based on mathematical programming with the branch-and-bound mechanism, in order to assess its overall performance. In these experiments, we have applied only the basic repair heuristic at each branching node, with no local search to improve the possibly found feasible solution and no destroy-and-repair metaheuristic.

Table 5 reports the results obtained after one hour of computation by the two competing approaches. The structure is similar to that of the previous tables. Three columns describe the features of the random instances. Two groups of four columns provide the results for the branch-and-bound algorithms based, respectively, on the Lagrangian and on the covering-packing relaxations. The first column of each group (labeled GapUB) reports the average gap, for each class of 5 instances with the same features, of the upper bound UB returned by the algorithm with respect to the optimum z^*. This is defined as $(UB - z^*)/z^*$ and expressed as a percentage. When the optimum is unknown, we overestimate the gap by $(UB - LB^*)/LB^*$. The second column of each group, with the label GapLB, contains the percentage gap of the lower bound returned by the algorithm, as already defined above. The third and fourth column report the average number of branching nodes (under the label BN) and the computational time in seconds (CPU).

The Lagrangian relaxation allows the branch-and-bound algorithm to solve 98 instances out of 135 to optimality, usually in a few seconds or minutes and with a few dozens of branching nodes. When the lower bound at the root node coincides with the optimum and the repair procedure finds an equivalent feasible solution, the problem is even solved directly without branching. This holds in particular when the number of subgraphs is small and the graph is dense. By contrast, the planar instances with a large number of subgraphs exhibit very large gaps even after one hour of computation and a large number of branching nodes. The covering-packing relaxation proves even better: nearly all dense and sparse instances are solved at the root node in a few seconds, and the same occurs for the planar ones, provided that the number of subgraphs p is limited. Overall, 119 instances are solved exactly. However, the planar graphs with large p remain hard, albeit with smaller final gaps.

6.6. Improvements with the refined heuristic

The metaheuristics proposed in Bruglieri and Cordone (2021) allow to prove that the worse performance of the branch-and-bound algorithm on the planar instances with large p mainly concerns the computation of the upper bound. A natural idea would be to apply the local search procedure to improve the result obtained at each branching node. This unfortunately does not work: while the results are indeed better, they still remain far from the best known solutions. This depends on the fact that the objective function often does not change when a vertex moves from a subgraph to another, because the minimum and maximum weight of all subgraphs remain the same. Another natural idea would be to apply the available metaheuristics at the beginning of the computation, in order to initialize the search. However, these algorithms take a nonnegligible time ($n/100$ s, that is, from 1 to 3 min) to get close to the best known results. For most instances, the branch-and-bound solves the problem to optimality in a much shorter time. Therefore, it would not make much sense to impose such an initialization step in general.

Hence, we introduced a more sophisticated management of the interaction between metaheuristic and exact approach, that allows to run the metaheuristic only on hard instances, without defining them *a priori*, and keeps a controlled balance between the time dedicated to the metaheuristic and to the exact algorithm. In practice, we run the exact method for $T_{ref} = 9$ s, suspend it, run the metaheuristic for $T_{LNS} = 1$ s starting from the best known solution and resume the branch-and-bound, alternating them periodically until the exact algorithm terminates. Consequently, for the hardest instances the metaheuristic takes approximately one tenth of the total computational time, but in half of the cases it is not applied and in most of the other cases it is applied only once. As for the specific parameters of the metaheuristic, we applied the values tuned in Bruglieri and Cordone (2021): $q = 0.15$ and $\rho = 50$.

Table 6 reports the results of the branch-and-bound without and with the destroy-and-repair metaheuristic. Its structure is the same as that of Table 5: three columns describe the features of the instances and two groups of columns provide, for the two competitors, the percentage gaps of the upper and the lower bound, the number of branching nodes and the computational time in seconds, respectively.

The results show that the introduction of the destroy-and-repair metaheuristic strongly reduces the percentage gap of the upper bound (the maximum gap decreases from 54% to 12%). Indeed, one additional instance is solved to optimality within the time limit of one hour. Moreover, the best known result found in Bruglieri and Cordone (2021) improves for 12 instances out of 135, while the simple branch-and-bound improves only one.

On the other hand, the metaheuristic subtracts time from the processing of the branching nodes. In spite of that, the number of branching nodes and the computational time are almost the same and the quality of the lower bound is unaffected, except for three instances in which it decreases by one.

Table 5
Performance of the branch-and-bound based on the Lagrangian and covering-packing relaxation on the random instances.

Type	p	n	Lagrangian				Covering-packing			
			GapUB	GapLB	BN	CPU	GapUB	GapLB	BN	CPU
Planar	$\ln(n)$	100	0.0%	0.0%	23.8	0.6	0.0%	0.0%	1.0	0.3
		200	0.0%	0.0%	13.0	7.9	0.0%	0.0%	1.4	2.4
		300	0.0%	0.0%	196.6	50.6	0.0%	0.0%	1.4	14.0
	$\sqrt{n}$	100	0.0%	0.0%	862.2	6.5	0.0%	0.0%	1.4	0.3
		200	0.0%	0.1%	24 549.0	1105.0	0.0%	0.0%	133.0	56.0
		300	0.0%	0.0%	12 313.4	722.2	0.0%	0.1%	136.6	742.8
	$n/\ln(n)$	100	3.4%	3.8%	733 515.8	3600.0	0.5%	0.1%	6150.2	792.0
		200	19.9%	7.7%	96 789.0	3600.0	8.3%	1.0%	3842.6	3600.0
		300	83.6%	13.0%	8 591.0	3600.0	39.2%	4.0%	722.6	3600.0
Sparse	$\ln(n)$	100	0.0%	0.0%	34.2	16.4	0.0%	0.0%	1.0	0.6
		200	0.0%	0.0%	12.6	145.8	0.0%	0.0%	1.0	5.0
		300	0.3%	0.1%	24.2	2080.6	0.0%	0.0%	1.0	15.2
	$\sqrt{n}$	100	0.0%	0.0%	2.6	1.7	0.0%	0.0%	1.0	0.4
		200	0.0%	0.3%	230.2	1112.2	0.0%	0.0%	1.0	5.0
		300	0.6%	0.2%	19.0	1987.7	0.0%	0.0%	1.0	14.6
	$n/\ln(n)$	100	0.0%	0.0%	412.6	63.8	0.0%	0.0%	6.6	2.4
		200	0.5%	1.6%	287.0	2924.1	0.0%	0.0%	10.6	51.1
		300	1.5%	0.6%	27.4	3112.6	0.2%	0.1%	36.2	732.1
Dense	$\ln(n)$	100	0.0%	0.0%	1.4	1.0	0.0%	0.0%	1.0	0.5
		200	0.0%	0.0%	13.8	167.0	0.0%	0.0%	1.0	6.0
		300	0.2%	0.0%	24.2	2251.1	0.0%	0.0%	1.0	23.7
	$\sqrt{n}$	100	0.3%	0.0%	70 451.0	724.6	0.0%	0.0%	1.0	0.5
		200	0.0%	0.0%	41.0	608.0	0.0%	0.0%	1.0	6.1
		300	0.3%	0.0%	22.6	2242.3	0.0%	0.0%	1.0	23.9
	$n/\ln(n)$	100	0.0%	0.0%	5 776.6	612.0	0.0%	0.0%	1.0	0.5
		200	0.0%	0.0%	53.0	744.4	0.0%	0.0%	1.0	6.0
		300	1.9%	0.1%	19.4	2774.5	0.0%	0.0%	1.0	23.5

Table 6
Performance of the branch-and-bound with the covering-packing relaxation without or with the destroy-and-repair metaheuristic on the random instances.

Type	p	n	Branch-and-bound				Branch-and-bound + metaheuristic			
			GapUB	GapLB	BN	CPU	GapUB	GapLB	BN	CPU
Planar	$\ln(n)$	100	0.0%	0.0%	1.0	0.3	0.0%	0.0%	1.0	0.3
		200	0.0%	0.0%	1.4	2.4	0.0%	0.0%	1.4	2.4
		300	0.0%	0.0%	1.4	14.0	0.0%	0.0%	1.4	14.2
	$\sqrt{n}$	100	0.0%	0.0%	1.4	0.3	0.0%	0.0%	1.4	0.3
		200	0.0%	0.0%	133.0	56.0	0.0%	0.0%	133.0	60.6
		300	0.0%	0.1%	136.6	742.8	0.0%	0.1%	129.4	733.8
	$n/\ln(n)$	100	0.5%	0.1%	6150.2	792.0	0.3%	0.1%	5723.8	787.2
		200	8.3%	1.0%	3842.6	3600.0	1.1%	1.1%	3242.6	3172.4
		300	39.2%	4.0%	722.6	3600.0	6.1%	4.1%	679.8	3600.0
Sparse	$\ln(n)$	100	0.0%	0.0%	1.0	0.6	0.0%	0.0%	1.0	0.6
		200	0.0%	0.0%	1.0	5.0	0.0%	0.0%	1.0	5.0
		300	0.0%	0.0%	1.0	15.2	0.0%	0.0%	1.0	15.2
	$\sqrt{n}$	100	0.0%	0.0%	1.0	0.4	0.0%	0.0%	1.0	0.4
		200	0.0%	0.0%	1.0	5.0	0.0%	0.0%	1.0	5.0
		300	0.0%	0.0%	1.0	14.6	0.0%	0.0%	1.0	14.6
	$n/\ln(n)$	100	0.0%	0.0%	6.6	2.4	0.0%	0.0%	6.6	2.4
		200	0.0%	0.0%	10.6	51.1	0.0%	0.0%	6.6	35.1
		300	0.2%	0.1%	36.2	732.1	0.1%	0.1%	35.0	740.0
Dense	$\ln(n)$	100	0.0%	0.0%	1.0	0.5	0.0%	0.0%	1.0	0.5
		200	0.0%	0.0%	1.0	6.0	0.0%	0.0%	1.0	6.1
		300	0.0%	0.0%	1.0	23.7	0.0%	0.0%	1.0	23.7
	$\sqrt{n}$	100	0.0%	0.0%	1.0	0.5	0.0%	0.0%	1.0	0.5
		200	0.0%	0.0%	1.0	6.1	0.0%	0.0%	1.0	6.1
		300	0.0%	0.0%	1.0	23.9	0.0%	0.0%	1.0	23.9
	$n/\ln(n)$	100	0.0%	0.0%	1.0	0.5	0.0%	0.0%	1.0	0.5
		200	0.0%	0.0%	1.0	6.0	0.0%	0.0%	1.0	6.1
		300	0.0%	0.0%	1.0	23.5	0.0%	0.0%	1.0	23.7

Tables 7 and 8 compare the branch-and-bound without and with the destroy-and-repair metaheuristic on the two real-world benchmarks. This time, each row corresponds to a single instance. The first two columns identify the name of the instance and the number of subgraphs. The following two groups of columns provide the usual information for the two competitors: percentage gaps of the upper and the lower bound, number of branching nodes and computational time in seconds (0.0 stands for a time smaller than 0.05 s).

Table 7
Performance of the branch-and-bound with the covering-packing relaxation without or with the destroy-and-repair metaheuristic on the smaller real-world instances.

Instance	p	Branch-and-bound				Branch-and-bound + metaheuristic			
		GapUB	GapLB	BN	CPU	GapUB	GapLB	BN	CPU
Two-loop	$\ln(n)$	0.0%	0.0%	3	0.0	0.0%	0.0%	1	0.0
	$\sqrt{n}$	0.0%	0.0%	1	0.0	0.0%	0.0%	1	0.0
	$n/\ln(n)$	0.0%	0.0%	1	0.0	0.0%	0.0%	1	0.0
Two-reservoir	$\ln(n)$	0.0%	0.0%	1	0.0	0.0%	0.0%	1	0.0
	$\sqrt{n}$	0.0%	0.0%	1	0.0	0.0%	0.0%	1	0.0
	$n/\ln(n)$	0.0%	0.0%	5	0.0	0.0%	0.0%	5	0.0
GoYang	$\ln(n)$	0.0%	0.0%	1	0.0	0.0%	0.0%	1	0.0
	$\sqrt{n}$	0.0%	0.0%	1	0.0	0.0%	0.0%	1	0.0
	$n/\ln(n)$	0.0%	0.0%	1	0.0	0.0%	0.0%	1	0.0
Blacksburg	$\ln(n)$	0.0%	0.0%	1	0.0	0.0%	0.0%	1	0.0
	$\sqrt{n}$	0.0%	0.0%	29	0.2	0.0%	0.0%	25	0.2
	$n/\ln(n)$	0.0%	0.0%	125	0.8	0.0%	0.0%	123	0.8
BakRyan	$\ln(n)$	0.0%	0.0%	5	0.1	0.0%	0.0%	5	0.1
	$\sqrt{n}$	0.0%	0.0%	159	2.3	0.0%	0.0%	159	2.3
	$n/\ln(n)$	0.0%	0.0%	4 807	43.8	0.0%	0.0%	4 807	45.0
Fossoli	$\ln(n)$	0.0%	0.0%	1	0.0	0.0%	0.0%	1	0.0
	$\sqrt{n}$	0.0%	0.0%	1	0.0	0.0%	0.0%	1	0.0
	$n/\ln(n)$	0.0%	0.0%	21	0.4	0.0%	0.0%	21	0.4
Pescara	$\ln(n)$	0.0%	0.0%	1	0.0	0.0%	0.0%	1	0.1
	$\sqrt{n}$	0.0%	0.0%	11	0.4	0.0%	0.0%	11	0.4
	$n/\ln(n)$	0.0%	0.0%	99 973	1690.2	0.0%	0.0%	99 973	1765.7
Modena	$\ln(n)$	0.0%	0.0%	1	5.2	0.0%	0.0%	1	5.2
	$\sqrt{n}$	0.0%	0.0%	113	242.4	0.0%	0.0%	113	254.5
	$n/\ln(n)$	187.3%	1.5%	2 033	3600.0	3.3%	1.5%	1 899	3600.0

Table 8
Performance of the branch-and-bound with the covering-packing relaxation without or with the destroy-and-repair metaheuristic on the larger real-world instances.

Instance	p	Branch-and-bound				Branch-and-bound + metaheuristic			
		GapUB	GapLB	BN	CPU	GapUB	GapLB	BN	CPU
Richmond	$\ln(n)$	0.0%	0.0%	3	0.4	0.0%	0.0%	3	0.0
	$\sqrt{n}$	0.0%	0.0%	87	0.8	0.0%	0.0%	87	0.8
	$n/\ln(n)$	0.0%	0.0%	715	6.2	0.0%	0.0%	715	6.2
D-town	$\ln(n)$	0.0%	0.0%	573	389.3	0.0%	0.0%	573	412.0
	$\sqrt{n}$	21.0%	1.3%	9 953	3600.0	21.0%	1.3%	9 165	3600.0
	$n/\ln(n)$	415.7%	121.7%	3 301	3600.0	202.4%	121.7%	3 117	3600.0
Balerma	$\ln(n)$	0.0%	0.0%	11	19.3	0.0%	0.0%	11	9.8
	$\sqrt{n}$	0.6%	0.6%	14 059	3600.0	0.6%	0.6%	13 655	3600.0
	$n/\ln(n)$	186.2%	34.1%	8 943	3600.0	58.0%	34.1%	8 445	3600.0
cellplot3	$\ln(n)$	0.0%	0.0%	1	79.2	0.0%	0.0%	1	45.0
	$\sqrt{n}$	0.0%	0.0%	1	57.3	0.0%	0.0%	1	46.7
	$n/\ln(n)$	27.7%	3.6%	75	3600.0	8.2%	3.6%	73	3600.0
cellplot2	$\ln(n)$	0.0%	0.0%	1	318.0	0.0%	0.0%	1	308.5
	$\sqrt{n}$	0.0%	0.0%	1	318.5	0.0%	0.0%	1	311.2
	$n/\ln(n)$	137.6%	16.9%	13	3600.0	64.2%	16.9%	13	3600.0
cellplot1	$\ln(n)$	0.0%	0.0%	1	999.2	0.0%	0.0%	1	508.4
	$\sqrt{n}$	0.0%	0.3%	3	1568.0	0.0%	0.3%	3	1970.3
	$n/\ln(n)$	140.4%	1.6%	7	3600.0	2.4%	1.6%	7	3600.0

The real-world instances have a structure similar to the planar ones, that is, they are very sparse, and therefore pose a similar challenge to the algorithms. Correspondingly, the results are similar. When the number of subgraphs is small, the problem can be quickly solved, as the lower bound is very tight and only few branching nodes (if any) are necessary. When p is large, the lower bound is still tight, but usually not zero, and finding a good upper bound can be hard. In fact, only the branch-and-bound with the destroy-and-repair metaheuristic is consistently able to do it.

7. Conclusions

This paper proposes a mathematical programming approach to the *Minimum Gap Graph Partitioning Problem*, which aims to partition a graph minimizing the sum of all maximum weight differences between the vertices in each generated subgraph. Based on an extended formulation of exponential size, we build two relaxations that allow to reduce the relevant variables to a quadratic number, relying on a branching mechanism to close the gap with respect to the original problem. Both relaxations provide tight bounds for most instances, except for those with very sparse graphs and a large number of subgraphs. For these instances, the covering-packing relaxation is still able to yield tight lower bounds, but it requires a more refined heuristic to tighten also the upper bounds. Overall, 157 instances out of a benchmark of 177 are solved to optimality, most of the time in a few seconds, improving a number of best known results previously obtained by metaheuristic approaches.

CRediT authorship contribution statement

Maurizio Bruglieri: Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Gianluca Consiglio:** Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Formal analysis. **Roberto Cordone:** Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization.

Data availability

Data will be made available on request.

References

Albornoz, V., Véliz, M., Ortega, R., Ortíz-Araya, V., 2020. Integrated versus hierarchical approach for zone delineation and crop planning under uncertainty. Ann. Oper. Res. 286, 617—-634.

Ayall, T.A., Liu, H., Zhou, C., Seid, A.M., Gereme, F.B., Abishu, H.N., Yacob, Y.H., 2022. Graph computing systems and partitioning techniques: A survey. IEEE Access 10, 118523–118550.

Bader, D.A., Meyerhenke, H., Sanders, P., Wagner, D. (Eds.), 2013. Graph partitioning and graph clustering. In: Contemporary Mathematics, vol. 588, American Mathematical Society, Providence, Rhode Island.

Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P., Vance, P.H., 1988. Branch-and-price: Column generation for solving huge integer programs. Oper. Res. 46 (3), 316–329.

Bichot, C.-E., Siarry, P. (Eds.), 2013. Graph Partitioning. Wiley-ISTE, London, United Kingdom.

Boob, D., Sawlani, S., Wang, D., 2019. Faster width-dependent algorithm for mixed packing and covering LPs. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E., Garnett, R. (Eds.), Advances in Neural Information Processing Systems. Vol. 32, Curran Associates, Inc..

Bruglieri, M., Cordone, R., 2016. Partitioning a graph into minimum gap components. Electron. Notes Discret. Math. 55, 33–36.

Bruglieri, M., Cordone, R., 2021. Metaheuristics for the minimum gap graph partitioning problem. Comput. Oper. Res. 132, 105301.

Bruglieri, M., Cordone, R., Lari, I., Ricca, F., Scozzari, A., 2021. On finding connected balanced partitions of trees. Discrete Appl. Math. 299, 1–16.

Bui, X.K., Marlim, M.S., Kang, D., 2020. Water network partitioning into District Metered Areas: A state-of-the-art review. Water 2 (4), 1002.

Caprara, A., Fischetti, M., Toth, P., 1999. A heuristic method for the set covering problem. Oper. Res. 47 (5), 730–743.

Caprara, A., Toth, P., Fischetti, M., 2000. Algorithms for the set covering problem. Ann. Oper. Res. 98 (1), 353–371.

Çatalyürek, Ü., Devine, K., Faraj, M., Gottesbüren, L., Heuer, T., Meyerhenke, H., Sanders, P., Schlag, S., Schulz, C., Seemaier, D., Wagner, D., 2023. More recent advances in (hyper)graph partitioning. ACM Comput. Surv. 55 (12), http://dx.doi.org/10.1145/3571808.

Ceria, S., Nobili, P., Sassano, A., 1998. A Lagrangian-based heuristic for large-scale set covering problems. Math. Program. 81, 215–228.

Chang, K.C., Shao, S.H., Zhang, D., 2017. Cheeger's cut, maxcut and the spectral theory of 1-Laplacian on graphs. Sci. China Math. 60, 1963–1980.

Chlebikova, J., 1996. Approximability of the maximally balanced connected partition problem in graphs. Inform. Process. Lett. 60, 225–230.

Cordone, R., Franchi, D., Scozzari, A., 2022. Cardinality constrained connected balanced partitions of trees under different criteria. Discrete Optim. 46, 100742.

Goemans, M.X., Williamson, D.P., 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM 42 (6), 1115–1145. http://dx.doi.org/10.1145/227683.227684.

Goldschmidt, O., Hochbaum, D.S., 1988. Polynomial algorithms for the k-cut problem. In: Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Sciences. FOCS, pp. 444–451.

Guignard, M., 2003. Lagrangean relaxation. TOP: Off. J. Span. Soc. Stat. Oper. Res. 11 (2), 151–200.

Ito, T., Nishizeki, T., Schröder, M., Uno, T., Zhou, X., 2012. Partitioning a weighted tree into subtrees with weights in a given range. Algorithmica 62, 823–841.

Ito, T., Zhou, X., Nishizeki, T., 2007. Partitioning a weighted graph to connected subgraphs of almost uniform size. IEICE Trans. Inf. Syst. E90-D (2), 449–456.

Krauthgamer, R., Naor, J., Schwartz, R., 2009. Partitioning graphs into balanced components. In: Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms. SODA '09, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 942–949.

Kuo, Y.-H., Leung, J.M., 2016. On the mixed set covering, packing and partitioning polytope. Discrete Optim. 22 (A), 162–182.

Paola, F.D., Fontana, N., Galdiero, E., Giugni, M., degli Uberti, G.S., Vitaletti, M., 2014. Optimal design of district metered areas in water distribution networks. Procedia Eng. 70, 449–457.

Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M., Puget, J.-F. (Eds.), 4th International Conference on Principles and Practice of Constraint Programming. CP98, Vol. 1520, Springer Verlag, Pisa, Italy, pp. 417–431.

Shor, N., 1985. Minimization methods for non-differentiable functions. In: Springer Series in Computational Mathematics, vol. 3, Springer-Verlag New York, Inc..

Wang, Q., Guidolin, M., Savic, D., Kapelan, Z., 2014. Two-objective design of benchmark problems of a water distribution system via MOEAs: Towards the best-known approximation of the true Pareto front. J. Water Resour. Plan. Manag. 141 (3), 04014060–1–14.

Wu, S., Hou, J., 2023. Graph partitioning: An updated survey. AKCE Int. J. Graphs Comb. 20 (1), 9–19.