On finding connected balanced partitions of trees.

Maurizio Bruglieri[1], Roberto Cordone[2], Isabella Lari[3], Federica Ricca[3], Andrea Scozzari[4]

[1] Politecnico di Milano, maurizio.bruglieri@polimi.it
[2] Università degli Studi di Milano, roberto.cordone@unimi.it
[3] Sapienza Università di Roma, {isabella.lari, federica.ricca}@uniroma1.it
[4] Università degli Studi Niccolò Cusano Roma, andrea.scozzari@unicusano.it

March 31, 2021

Abstract

Graph partitioning is a widely studied problem in the literature with several applications in real life contexts. In this paper we study the problem of partitioning a graph, with weights at its vertices, into p connected components. For each component of the partition we measure the difference between the maximum and the minimum weight of a vertex in the component. We consider two objective functions to minimize, one measuring the maximum of such differences among all the components in the partition, and the other measuring the sum of the differences between the maximum and the minimum weight of a vertex in each component. We focus our analysis on tree graphs and provide polynomial time algorithms for solving these optimization problems on such graphs. In particular, we present an $O(n^2 \log n)$ time algorithm for the min-max version of the problem on general trees and several, more efficient polynomial algorithms for some trees with a special structure, such as spiders and caterpillars. Finally, we present NP-hardness and approximation results on general graphs for both the objective functions.

Keywords: Connected partitioning of trees, min-sum gap optimization, min-max gap optimization.

1 Introduction

In clustering problems the aim is to partition a set of units into groups in order to obtain clusters that are as similar as possible within them and as dissimilar as possible to each other. There are a number of objective functions that have been provided in the literature to pursue this goal. A novel criterion in clustering objects was introduced in [13] where the author seeks clusters with limited *range* between values associated to the elements of each cluster. The range of a cluster is defined as the difference between the maximum and minimum values of two elements in the cluster. From a graph partitioning viewpoint, the problem can be formulated as partitioning a complete vertex-weighed graph where each vertex represents an object. In [13] the author presents a family of *range-based* clustering objective functions based on common clustering optimization criteria and shows that, in general, the range-based problems are easier to solve than the corresponding *total similarity* problems. Applications of range-based problems are meaningful in different contexts, for example in image segmentation problems (see [14]).

In this paper we study range-based clustering problems on special classes of graphs. We introduce an additional constraint imposing that the clusters must be connected. This condition is not guaranteed when the graph is not complete, but it is mandatory in several applications in which the graph represents a territory and its vertices correspond to territorial units. In this case, it is generally required that units in the same cluster are *contiguous* (*connected*). Some results on range-based clustering problems with connectivity constraints were presented in [5, 7]. In these works, the problem was denoted as the *Minimum Gap Graph Partitioning Problem* (MGGPP) where the *gap* of a cluster (or of a component) is the difference between the maximum and the minimum weight of a vertex in the connected component. Hence, in order to specialize and distinguish our contribution from the one in [13], we adopt the same notation used in [5, 7], and refer to the problem under study (with connectivity constraints) as MGGPP.

More generally, MGGPP is related to the broad class of uniform graph partitioning problems (see, e.g., [17] and the references therein). Typically, in this kind of problems weights are assigned to the vertices of the graph and the aim is to balance the components' weights (sum of the weights of their vertices) as much as possible. The objective function can be formulated in different ways. One possibility is to minimize the difference between the maximum and the minimum weight of a component. Alternatively, components' weights can be balanced by minimizing the maximum weight or maximizing the minimum one. This can be considered an *external* uniformity criterion aimed at balancing components' weights as much as possible. Another objective is trying to balance the weights of the vertices belonging to the same component. This can be considered as an *internal* uniformity criterion and the objective function can be precisely modelled by the difference between the maximum and the minimum weight in a component (*component's gap*). One can either minimize the maximum component gap (*min-max* MGGPP), or the sum of all of them (*min-sum* MGGPP).

MGGPP on special classes of graphs is motivated by applications in different fields. First, the standard approach to manage large Water Distribution Networks (WDN) is to sectorize them into subnetworks called District Metered Areas (*DMA*s) [8]. This allows localizing leakages more accurately, by monitoring the input and the output discharges for each district. Moreover, it also achieves a better management of pressure through valves and turbines that produce energy, reduce the amount of existing losses and limit the occurrence of new damages. The optimal design of *DMA*s takes into account, among other objectives, the minimization of the difference between the required heads within the *DMA*s. The purpose is to establish a unique target pressure value in each *DMA*, and consequently, to achieve an efficient pressure regulation also in networks with strong variations in ground elevation [11]. If the network is represented as a general graph where the edges correspond to pipes, the vertices to their intersections and the weight of a vertex is the ground elevation of an intersection, the MGGPP models the search for an optimal sectorization of the WDN.

Another application of our problem on general graphs is in the agriculture field where the levelling of farmlands is an important foundation of modern ground irrigation systems, as it improves the uniformity of irrigation and soil salt distribution, thus controlling weeds, saving water and increasing yield [3]. In general, it might be impractical or too expensive to flatten a sloping land as a whole. In this case, it is necessary to divide the land into parcels, and build a flat terrace on each parcel by suitable earthworks. Selecting parcels in which the difference between their highest and lowest elevation is small, helps to reduce the amount of ground to be moved and, therefore, the cost associated with the following earthworks. The MGGPP models the division of a land into parcels with a limited difference in height (or in other soil properties [4]): the vertices of the graph correspond to sampled locations in the land, the weights to the their heights, while the edges are the links between adjacent locations.

MGGPP was studied as a general graph partitioning problem in [5] and [6] In particular, a first computational complexity and approximability analysis is studied in [5], while a Tabu Search metaheuristic and a Mixed Integer Linear Programming (MILP) formulation for the *min-sum* version are proposed in [6]. Moreover, some polynomial cases of the MGGPP concerning special graphs such as paths, spiders, stars, caterpillars and complete graphs are investigated in [7]. The difficulty in solving MGGPP derives precisely from the connectivity constraints on the partition's components. We note that the *min-max MGGPP* can be seen as a special case of the following more general graph partitioning problem. Given a graph G and an $n \times n$ matrix of *dissimilarities* between any pair of vertices, find a partition of G into p connected components that minimizes the maximum dissimilarity of a pair of vertices in the same component. This problem is $\mathcal{NP}$-complete even on star graphs [15, 19]. However, this negative result does not necessarily affect the complexity of the min-max MGGPP. In fact, this problem might be easier than the dissimilarity one due to the fact that the dissimilarity for any given pair of vertices u and v is computed by $|w_u - w_v|$, where w_u and w_v are the weights of u and v, respectively.

In this paper we focus on the partition of tree graphs and study both the min-max and min-sum MGGPP. In particular, we provide an $O(n^2 \log n)$ time algorithm for minimizing the maximum gap on a general tree. Then, we further investigate MGGPP on special classes of trees, such as spiders and caterpillars, for which we obtain algorithms with very low time complexities. Finally, the computational complexity and the approximability of MGGPP for general graphs is investigated.

In the remainder of the paper we first provide some notation and definitions in Section 2. In Section 3, we illustrate the polynomial time algorithm for the min-max MGGPP on a general tree. The complexity status of the min-sum MGGPP on trees in the general case is still open. In Section 4 we, first, recall some known results of MGGPP on special classes of trees, that are used in the sequel, and then we study the problem on spiders and caterpillars considering both the min-max and the min-sum objective functions. For these two cases we provide efficient polynomial time algorithms. In Section 5, we introduce a variant of MGGPP with an additional constraint requiring that any component of the partition has at least two vertices. We call this problem the *Minimum Gap Graph Partitioning Problem with cardinality at least 2* (MGGPP2). This version of the problem is introduced to model situations in which components corresponding to singletons are not meaningful. In Section 6, we provide some $\mathcal{NP}$-completeness and approximation results for general graphs. In Section 7, some ideas for future research on the topic are depicted. Finally, two appendices report the technical details of some of the procedures presented in the paper.

In order to highlight our results we provide the following table where, for the sake of completeness, we report in bold the new results on the specific classes of graphs considered in the present paper. The already known results for the special trees derive from [7], while those for the general graphs are just sketched in [5], but developed here in detail. In the table we denote by n the total number of vertices of the graph and by p the number of the partition's components. We denote by "?" the open problems.

	min-max		*min-sum*	
Topology	MGGPP	*MGGPP2*	MGGPP	*MGGPP2*
Trees	$\mathbf{O(n^2 \log n)}$	?	?	?
Paths	$O(n \log n)$	$\mathbf{O(n^2 \log n)}$	$O(n^2 p)$	$O(n^2 p)$
Spiders	$O(n\, log^2\, n)$	$\mathbf{O(n^4 \log n)}$	$\mathbf{O(n^4 p)}$	$\mathbf{O(n^4 p)}$
Caterpillars	$\mathbf{O(n^2 \log n)}$	$O(n^2 p)$	$\mathbf{O(n^3 logn + n^2 p^2)}$	$O(n^2 p)$
General graphs	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard	?	$\mathcal{NP}$-hard

Table 1: Summary of the computational complexity results

2 Notation, definitions and properties

Let $G = (V, E)$, with $|V| = n$, be a connected undirected graph, w_v an integer *weight* defined for each vertex $v \in V$, and p a positive integer such that $1 < p < n$. Given a vertex subset $U \subseteq V$, we denote by $m_U = \min_{u \in U} w_u$ and $M_U = \max_{u \in U} w_u$ the minimum and maximum vertex weight in U, respectively. The *range* of a set $U \subseteq V$ is the minimal interval containing all weights of the vertices in U. We define the *gap* of U as $\gamma_U = M_U - m_U$ (if U is a singleton, $\gamma_U = 0$). The MGGPP can be stated as follows: partition G into p vertex-disjoint connected subgraphs $G_r = (V_r, E_r)$, with $r = 1, \ldots, p$, such that a function of the gaps γ_{V_r} is minimized. In the *min-max* version of the problem the objective function is $f^{MM} = \max_{r=1,\ldots,p} \gamma_{V_r}$, while the *min-sum* version minimizes the sum of all gaps $f^{MS} = \sum_{r=1}^{p} \gamma_{V_r}$.

In the following we provide a general property of MGGPP.

Remark 1 *Given a partition π in p components with maximum gap equal to γ, another partition π' in p' components with maximum gap less than or equal to γ exists for any p' such that $p < p' < n$.*

In fact, it is always possible to obtain a partition into p' components with maximum gap $\leq \gamma$ by moving $p' - p$ times a suitable vertex from one of the current components into a new singleton component, without disconnecting any components.

Remark 1 will be exploited to solve the min-max MGGPP by the solution of a polynomial number of instances of the following auxiliary problem, MGGPP(γ), which is defined by fixing a value for the integer parameter γ.

Problem MGGPP(γ): find a connected partition of G having the minimum number $q(\gamma)$ of components and such that each component has gap at most γ.

In the following, a *γ-best partition* of a graph G is an optimal solution to MGGPP(γ) on G.

Given an instance of the min-max MGGPP, we repeatedly solve MGGPP(γ) so as to find the minimum value of γ for which a partition into p connected components with gap at most γ exists. This can be done efficiently by a binary search over all the $O(n^2)$ possible values of γ, i.e., the value of all the possible differences between any two vertex weights in G. For any γ, if $q(\gamma) > p$, it is impossible to find a solution to min-max MGGPP (which requires exactly p components) having gap at most γ. Therefore, in this case, the value γ must be increased. On the other hand, if $q(\gamma) < p$, we can decrease γ and solve again MGGPP(γ) to check whether a solution for the min-max MGGPP with an improved objective function value can be found. At the end of the binary search over the γ values, in case $q(\gamma) < p$, Remark 1 guarantees that a partition in exactly p components with the same maximum gap can be found by further dividing some components of the partition until p components are obtained. The binary search requires a pre-sorting of the $O(n^2)$ possible values of γ, implying an overall time complexity of $O(n^2 \log n)$. This complexity can be reduced as follows. First sort the weights of the vertices, in $O(n \log n)$ time; then consider implicitly the ordered matrix of the differences between weights and apply the $O(n)$ time procedure for finding the k^{th} smallest value in a $n \times n$ matrix with sorted rows and columns [20]. With this approach the following general complexity result holds.

Proposition 1 *A solution procedure for the min-max MGGPP, which applies a binary search over all the possible values of γ and finds a γ-best partition, has an overall time complexity $O(n \log n + T_\gamma(G) \log n)$, where $T_\gamma(G)$ is the time required for solving MGGPP(γ).*

The above results will be fully exploited in Section 3 for solving min-max MGGPP on trees.

3 The gap partitioning problem on trees

In this section we propose a polynomial time algorithm for solving the min-max version of MGGPP on a tree $T = (V, E)$. The algorithm follows the approach illustrated in Section 2 which basically solves a sequence of problems MGGPP(γ) for a logarithmic number of different values of γ. For a fixed positive integer γ, MGGPP(γ) corresponds to finding a connected partition of T having the minimum number of components and such that each component has gap at most γ.

In order to solve the problems MGGPP(γ), we root T at an arbitrary vertex r and denote by T_r the resulting rooted tree. For each $v \in V$, let T_v be the subtree of T_r rooted at v and $V(T_v)$ the corresponding set of vertices. For each $v \in V$, $Succ(v)$ is the set of children of v in T_r, and, for each $v \in V - \{r\}$, $p(v)$ is the parent of v in T_r.

For a given γ, a dynamic programming algorithm is applied for finding a γ-best partition of T_r. The algorithm visits the tree bottom up, level by level, from the leaves to the root, and, for each vertex v, it finds a γ-best partition of T_v by solving for a polynomial number of times a network optimization problem known in the literature as the *Minimum Weight Closure Problem* (MWCP) [1, 12]. Since MWCP can be solved in polynomial time on trees, it follows that MGGPP(γ) can be solved in polynomial time on a trees, as well.

Definition 1 *Consider a tree T_v rooted at a vertex v and a real weight associated to each vertex of T_v. A* closure *R of T_v is a subset of vertices of T_v such that if $u \in R - \{v\}$, then $p(u) \in R$. A* Minimum Weight Closure *of T_v is a closure such that the sum of the weights of its vertices is minimum.*

We denote by n_v the number of components of a γ-best partition of T_v. When v is a leaf, we have $n_v = 1$. We notice that in any partition of T_v, the minimum weight of the component containing v is one of the $O(n)$ weights of the vertices of T_v. This suggests a way for obtaining n_v and a γ-best partition of T_v efficiently. Consider the weights of the vertices u in $V(T_v)$, such that $w_v - \gamma \leq w_u \leq w_v$. For each of them, set $a = w_u$ and define a *γ_a-best partition* of T_v as a partition minimizing the number of components among all the connected partitions of T_v having gap at most γ and such that the range of the component containing v is included in $[a, a + \gamma]$. Let $n_v(a)$ denote the number of components of a γ_a-best partition of T_v. Among all such γ_a-best partitions, the γ-best partition of T_v is the one with the minimum $n_v(a)$. Note that both n_v and $n_v(a)$ depend on the currently fixed value of γ. We will show that, for a given minimum weight a, it is possible to formulate the problem of finding a γ_a-best partition of T_v as a Minimum Weight Closure Problem of T_v. A linear time algorithm for solving this problem on trees is described in [12].

Given a nonempty closure R of T_v, we denote by $\sigma(R)$ the set of nodes in $V(T_v) - R$ whose parent belongs to R, i.e.,

$$\sigma(R) = \{z \in V(T_v) - R : p(z) \in R\}.$$

Let $R_v(a)$ be the (unique) closure of T_v whose elements have weight in $[a, a+\gamma]$, and which is not contained in any other closure of T_v having the same property. We denote by $T_v(a)$ the tree rooted at v induced by $R_v(a)$. Notice that if $w_v \notin [a, a+\gamma]$ then $R_v(a)$ is empty and $T_v(a)$ is the null graph.

Let π^* be a γ_a-best partition of T_v. The component of π^* containing v is a closure of $T_v(a)$, which we denote by R^*, and, by the optimality of π^*, the number of components of a γ_a-best partition of T_v is given by

$$n_v(a) = 1 + \sum_{z \in \sigma(R^*)} n_z \tag{1}$$

If n_z has been already computed for all $z \in V(T_v) - \{v\}$, in order to find a γ_a-best partition of T_v, one has to find a closure R^* of $T_v(a)$ which minimizes $\sum_{z\in\sigma(R)} n_z$ among all closures R of $T_v(a)$. We will show that such a problem corresponds to a Minimum Weight Closure problem with the following weights defined on the nodes of $V - \{r\}$.

$$\rho(u) = \sum_{z \in Succ(u)} n_z - n_u \tag{2}$$

Assume that a partition π of T_v is given and let R be the component containing v. For each $u \in \sigma(R)$, let $\pi(u)$ be the partition obtained from π by adding u to the component R. If π is formed by R and by a γ-best partition of T_u, for each $u \in \sigma(R)$, then the weights defined in (2) have the following meaning: for each $u \in \sigma(R)$, $\rho(u)$ is the difference between the number of components in π and in $\pi(u)$.

The following lemma establishes a relation between the number of components in a γ-best partition of T_v and the new weights $\rho(u)$ defined by (2). On the basis of this result, we will show how the problem of finding a γ_a-best partition of T_v can be reformulated as finding a minimum weight nonempty closure of T_v.

Lemma 1 *For a given nonempty closure R_v of T_v one has:*

$$\sum_{z \in \sigma(R_v)} n_z = \sum_{u \in R_v - \{v\}} \rho(u) + K_v \tag{3}$$

where $K_v = \sum_{z\in Succ(v)} n_z$. We observe that if v is a leaf we set $K_v = 0$.

Proof

We have:

$$\textstyle\sum_{z\in\sigma(R_v)} n_z =$$

$$\textstyle\sum_{z\in V(T_v)-R_v:p(z)\in R_v} n_z + \sum_{z\in R_v-\{v\}} n_z - \sum_{u\in R_v-\{v\}} n_u =$$

$$\textstyle\sum_{z\in V(T_v):p(z)\in R_v} n_z - \sum_{u\in R_v-\{v\}} n_u =$$

$$\textstyle\sum_{u\in R_v} \sum_{z\in Succ(u)} n_z - \sum_{u\in R_v-\{v\}} n_u =$$

$$\textstyle\sum_{u\in R_v-\{v\}} \sum_{z\in Succ(u)} n_z + \sum_{z\in Succ(v)} n_z - \sum_{u\in R_v-\{v\}} n_u =$$

$$\textstyle\sum_{u\in R_v-\{v\}} (\sum_{z\in Succ(u)} n_z - n_u) + \sum_{z\in Succ(v)} n_z =$$

$$\textstyle\sum_{u\in R_v-\{v\}} \rho(u) + \sum_{z\in Succ(v)} n_z =$$

$$\textstyle\sum_{u\in R_v-\{v\}} \rho(u) + K_v.$$

□

By Lemma 1, equation (1) can be written as follows:

$$n_v(a) = 1 + K_v + \sum_{u\in R_v(a)-\{v\}} \rho(u) \tag{4}$$

We observe that since K_v depends on v but not on R_v, it can be considered as a constant when searching for a partition of T_v. Therefore, the problem of finding a γ_a-best partition of T_v can be solved by considering the weights $\rho(\cdot)$ and searching for a minimum weight nonempty closure of T_v with range in $[a, a+\gamma]$.

In order to find a γ-best partition of T, we solve a minimum weight closure problem on $T_v(a)$ for each possible weight a of a vertex, and for each vertex $v \in V$. Since the algorithm for computing a minimum weight closure problem on rooted trees requires a time which is linear in the number of vertices, the resulting time complexity for finding a γ-best partition of T_r would be $O(n^3)$. In the following we describe how it is possible to reduce this time complexity to $O(n^2)$.

First of all we briefly describe the $O(n)$ time algorithm presented in [12] for computing a minimum weight closure of a tree T_r rooted at vertex r, having n vertices and weights c_v, for all $v \in V$. This algorithm visits the tree bottom up and, for each visited vertex $v \neq r$, computes a cumulative weight $\overline{c}_v$ that is the minimum contribution to the closure weight when v and possibly other vertices in the subtree rooted at v are added to the closure. If this contribution is nonnegative, v is not added to the closure, otherwise v is added and the cumulative weight of the parent of v is updated.

In view of the following remark, for each value a, the computation of all the closures of the trees $T_v(a)$ with weights $\rho(\cdot)$ can be done in $O(n)$ time by applying the above algorithm on T_r only once for each a. We denote by W the set of all possible values a.

Remark 2 *Given a vertex $v \in V$ and a descendant u of v in T_r, for each $a \in W$, consider the trees $T_v(a)$ and $T_u(a)$. Two cases are possible: either i) $T_u(a)$ is a subtree of $T_v(a)$, or ii) the vertex sets of $T_v(a)$ and $T_u(a)$ do not intersect. In particular, case i) holds when the weights of all vertices in the unique path joining v and u belong to the interval $[a, a+\gamma]$, otherwise case ii) holds.*

The above remark applies in particular when u is a child of v and, in case i), the weight of a minimum closure of $T_u(a)$ is the cumulative weight needed in the algorithm for computing the minimum closure of $T_v(a)$; on the contrary, in case ii), u is not involved in the computation of the min closure of $T_v(a)$. It follows that, for a given minimum weight a, the cumulative weight of each vertex must be computed at most once during a bottom up visit of T_r. In order to implement such a visit we define the following quantities:

$$r_v(a) = \begin{cases} +\infty & \text{if } w_v \notin [a, a+\gamma] \\ 0 & \text{if } v \text{ is a leaf and } w_v \in [a, a+\gamma] \\ \sum\limits_{u \in Succ(v)} \min\{0; r_u(a) + \rho(u)\} & \text{if } v \text{ is not a leaf and } w_v \in [a, a+\gamma] \end{cases} \tag{5}$$

In order to build the resulting γ_a-best partition of T, when v is not a leaf and $w_v \in [a, a+\gamma]$, it is useful to store the following sets used in the computation of the quantities $r_v(a)$:

$$S_v(a) = \{u \in Succ(v) : r_u(a) + \rho(u) < 0\}. \tag{6}$$

The procedure for finding a γ-best partition of T is illustrated in Algorithm 1.

Algorithm 1 ALGORITHM γ-PARTITION

```
Input: A tree $T$ and a vertex $r$; a weight $w_v$ for each $v \in V$; a positive integer value $\gamma$
Output: A $\gamma$-best partition of $T$
 1: Root $T$ at an arbitrary vertex $r$
 2: $W \leftarrow \{w_v : v \in V\}$
 3: for each vertex $v$ do          ▷ Visit the tree bottom up starting from the leaves to the root
 4:     if $v$ is a leaf then
 5:         $K_v \leftarrow 0$;
 6:     else
 7:         $K_v \leftarrow \sum_{z \in Succ(v)} n_z$
 8:     for each $a \in W$ do
 9:         Compute $r_v(a)$ according to formula (5)
10:         if $v$ is not a leaf and $w_v \in [a, a+\gamma]$ then
11:             $S_v(a) \leftarrow \{u \in Succ(v) : r_u(a) + \rho(u) < 0\}$          ▷ as in formula (6)
12:         $n_v(a) \leftarrow 1 + K_v + r_v(a)$          ▷ as in formula (4)
13:     $n_v \leftarrow \min_{a \in W}\{n_v(a)\}$
14:     $a_v \leftarrow \arg\min_{a \in W}\{n_v(a)\}$
15:     if $v \neq r$ then
16:         $\rho(v) \leftarrow K_v - n_v$          ▷ as in formula (2)
    ▷ $n_r$ is the minimum number of components of a partition of $T$ with gap at most $\gamma$
17: Visit the tree $T$ top-down, using $a_v$ and $S_v(a_v)$ to build a $\gamma$-best partition
```

From the above discussion, we can conclude about the computational complexity of the min-max MGGPP on trees.

Theorem 1 *The min-max MGGPP on trees can be solved in $O(n^2 \log n)$ time.*

Proof

For a given γ, ALGORITHM γ-PARTITION solves MGGPP(γ) on T in $O(n^2)$. In fact, for each vertex v, $O(n)$ quantities must be computed and they are used only once to compute the same quantities at $p(v)$ during the bottom-up visit of T_r (except when $v = r$). The same holds for the computation of the $O(n)$ sets $S_v(a)$ related to v and those related to $p(v)$. Finally, a γ-best partition of T can be found in $O(n)$ time by a top-down visit of the tree on the basis of the sets $S_v(a)$. Hence, by using the result of Proposition 1, the overall time complexity for solving the min-max MGGPP on T is $O(n^2 \log n)$. □

To let the reader understand how the solution procedure for MGGPP(γ) works, in the following we provide a simple example in which we find a γ_a-best partition of T_r for a given value of γ. Consider the tree with $n = 5$ vertices shown in Figure 1, and assume $\gamma = 2$. In Table 2 we report the computation of the quantities that allow to find a γ-best partition of T_r.

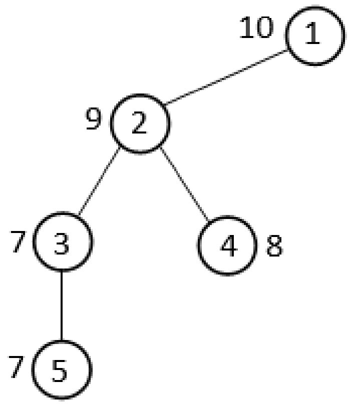

Figure 1: A sample application of ALGORITHM γ-PARTITION. The number beside each vertex corresponds to its weight. We assume that the tree is rooted at vertex 1.

Vertex	K_v	$r_v(a)$				$n_v(a)$				n_v	$\rho(v)$
		$a=7$	$a=8$	$a=9$	$a=10$	$a=7$	$a=8$	$a=9$	$a=10$		
1	1	$+\infty$	0	0	0	$+\infty$	2	2	2	2	-
2	2	-2	-1	0	$+\infty$	1	2	3	$+\infty$	1	1
3	1	-1	$+\infty$	$+\infty$	$+\infty$	1	$+\infty$	$+\infty$	$+\infty$	1	0
4	0	0	0	$+\infty$	$+\infty$	1	1	$+\infty$	$+\infty$	1	-1
5	0	0	$+\infty$	$+\infty$	$+\infty$	1	$+\infty$	$+\infty$	$+\infty$	1	-1

Table 2: The values in the table are computed filling in each row (from row 5 to row 1) applying ALGORITHM γ-PARTITION. The value $n_v = 2$ for $v = 1$ (i.e., the root of T) returns the minimum number of components in a γ-best partition of T_1. In this example, this minimum number can be obtained for $a = 8, 9, 10$ and corresponds to a partition formed either by the two sets of vertices $\{2, 3, 4, 5\}$ and $\{1\}$, or by the two sets of vertices $\{3, 5\}$ and $\{1, 2, 4\}$.

4 The gap partitioning problem on special classes of trees

In this section we study MGGPP on some special classes of tree graphs. We first recall some preliminary results, related to paths, which were already presented at the CTW2018 conference and published in the related Proceedings volume [7]. These results are useful in the procedures provided in this section for solving MGGPP problems on other special classes of trees. Also spider graphs were previously considered in [7] where it was shown that min-max MGGPP can be solved in $O(n \log^2 n)$ time on this class of graphs. Since in [7] we presented only a sketch of the proof of this result, here, we give a more detailed proof. In addition, in this paper, we provide new results related to the min-sum version of MGGPP on spider and caterpillar graphs. As we remark at the end of this section, after the result in Theorem 1, in this paper we improve on the complexity of the min-max MGGPP on caterpillar graphs w.r.t. the one provided in [7].

4.1 Paths, Spiders and Caterpillars

In [7] we provided the following results for the min-max and min-sum MGGPP on paths.

Theorem 2 *When G is a path, the min-max MGGPP can be solved in $O(n \log n)$ time.*

Theorem 3 *When G is a path, the min-sum MGGPP can be solved in $O\left(n^2 p\right)$ time.*

The first result follows from the binary search approach illustrated in Proposition 1. Theorem 3 is based on a general construction which from the given path produces an auxiliary directed graph $\mathcal{G}$, with $np+1$ nodes and $O(n^2 p)$ arcs, such that solving MGGPP on a path P is equivalent to finding a minimum cost path in $\mathcal{G}$ from the unique source node u_0 to a specific sink node. In the following we recall such construction since it is exploited later in this section to develop more sophisticated similar constructions used to provide new results for the min-sum version of MGGPP on spider and caterpillar graphs.

We assume that the vertices of a path P are numbered progressively considering an arbitrary direction, and we denote it by $P = \{v_1, \dots, v_n\}$. We build an auxiliary directed graph $\mathcal{G}$ whose node set includes a source node u_0 and p nodes $u_{j,1}, \dots, u_{j,p}$ for each v_j of V. The arc set includes an arc from u_0 to each node $u_{j,1}$ and an arc $(u_{i,r-1}, u_{j,r})$ for each pair of vertices v_i and v_j with $i < j$ and for $r = 2, \dots, p$. Arc $(u_0, u_{j,1})$ represents the possibility that the first component of the path's partition is a subpath going from v_1 to v_j, arc $(u_{i,r-1}, u_{j,r})$ refers to the possibility that the r-th subpath of the partition goes from v_{i+1} to v_j (see Figure 2). The arc costs are the gap values of the corresponding subpaths. An example for $n = 4$ and $p = 2$ is given in Figure 2.

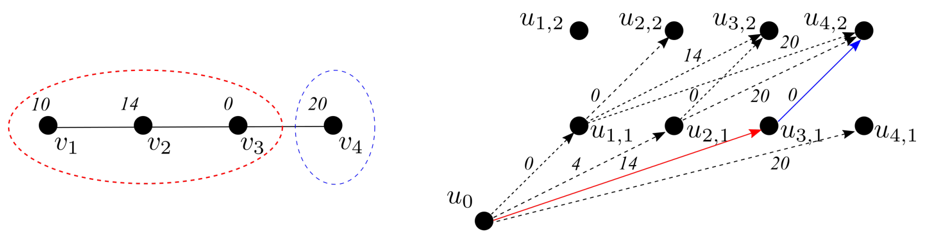

Figure 2: Construction of the auxiliary graph to partition a path P of $n = 4$ vertices into $p = 2$ subpaths.

By construction, every path from u_0 to $u_{n,p}$ in $\mathcal{G}$ consists of exactly p arcs, and corresponds to a partition of $\mathcal{G}$ into p subpaths (connected components). Depending on the objective function considered in the MGGPP, the cost of the path in $\mathcal{G}$ is defined as the sum of the costs of its arcs (*min-sum* version) or the maximum of the costs of its arcs (*min-max* version). In both cases, the problem reduces to find a minimum cost path from u_0 to $u_{n,p}$ on the auxiliary graph $\mathcal{G}$. Since the network $\mathcal{G}$ is acyclic and has $O(n^2p)$ arcs, solving the optimal path problem requires linear time with respect to the number of arcs of $\mathcal{G}$.

For solving min-sum MGGPP on spiders and caterpillars, we will introduce similar graph constructions, adapted to the specific case. These constructions share some basic features. The auxiliary graph consists of a dummy starting node u_0 and $O(p)$ layers. Each layer corresponds to a component of the required partition, and its nodes correspond to a subset of suitably ordered vertices of the original graph. The arcs of the auxiliary graph represent potential subgraphs of the original one if they link consecutive layers, or collections of subgraphs if some layers are skipped; in the latter case, the number of skipped layers is related to the number of subgraphs in the collection. The endpoints of an arc identify the vertices included in the corresponding collection of subgraphs: in particular, the head node is associated with the last vertex, and the tail node with the last vertex of the previous one. The cost of the arc represents the total gap of the corresponding collection of subgraphs (connected components of the partition).

In [7] we also presented a result related to the min-max version of MGGPP on spider graphs. In the following we sketch the idea of the algorithm reporting all the details and the technicalities in Appendix 1.
A spider graph G is a tree with at most one vertex of degree ≥ 3. We refer to this vertex as the *central vertex* v_0 of G and denote its degree by d. For this particular graph our polynomial time algorithm exploits the general solution approach based on binary search over the possible values γ of the optimal solution (see Section 2). We view a spider graph G as a set of d paths each one starting from one of the d different leaves of G and all ending at v_0 (we also refer to these paths as *legs* of the spider). Then, the idea of the algorithm consists in finding a partition of G in which $p - 1$ components are subpaths of $p - 1$ legs, and the remaining one, i.e., the *central component* C_{v_0}, is obtained by complement and it contains v_0.

Deciding which subpaths can be merged in the central component can be done in $O(n \log n)$ time. In Appendix 1 we show how this can be guaranteed by applying a procedure that scans all the ordered minimum and maximum weights of the vertices of the subpaths in linear time w.r.t. n. In light of Proposition 1, this implies the following results.

Proposition 2 *For a fixed value of γ, the time complexity for finding the central component C_{v_0} is $O(n \log n)$.*

Theorem 4 *[7]. When G is a spider, the* min-max MGGPP *can be solved in $O(n \log^2 n)$ time.*

Also the min-sum MGGPP can be solved in polynomial time on a spider graph. In this case we exploit the $O\left(n^2 p\right)$ procedure for MGGPP on paths as a subroutine (see [7]). The basic idea is to evaluate all possible weight ranges for the central component and build the remaining ones as subpaths of the spider's legs. For each pair of vertices (u, v) we check by a simple visit of the spider whether there exists a central component admitting them as the minimum and maximum weight vertex. Of course, the paths linking u and v with v_0 will belong to the central component and $w_u \leq w_{v_0} \leq w_v$. We denote as *grey vertices* all the other vertices whose weight belongs to the range $[w_u, w_v]$ and that can be reached from v_0 visiting only vertices with weights within that range. These vertices are candidate elements for the central component. The remaining $n' \leq n$ vertices, that must belong to the $p-1$ noncentral components, will be denoted as *white vertices.* The latter vertices are ordered by increasing values of the leg index k and (inside each leg) from the leaf to the central vertex (in other words, as if the legs of the spider were appended to form a single path).

We now build an auxiliary directed graph including a dummy source node u_0 and $p-1$ nodes for each white vertex, distributed on as many layers, denoted by $u_{i,r}$ with $i = 1, \ldots, n'$ and $r = 1, \ldots, p-1$. The arcs of the auxiliary graph represent potential subgraphs. An arc connects u_0 to each node of the first layer associated with leg $k = 1$; an arc connects each node of layer r (with $r = 1, \ldots, p-1$) to the following nodes associated with the same leg on layer $r+1$; finally, an arc connects the last node associated with a leg to each node associated with the next leg on the next layer. These limitations guarantee that the subgraph represented by an arc is fully included in a single leg. The cost of an arc is equal to the gap of the corresponding subgraph. The optimal path from u_0 to $u_{n',p-1}$ identifies the optimal partition w.r.t. the given range $[w_u, w_v]$, and it requires $O(n'^2 p)$ time. It is possible that no path exists from u_0 to $u_{n',p-1}$. This happens either when the number of legs is larger than $p-1$ or when the number of white vertices is smaller than $p-1$. In the former case, the problem for the current pair (u, v) is infeasible. In the latter, the grey vertices can be used to build the lacking subgraphs, as long as the total number of white and grey vertices is at least $p-1$. In this case, all noncentral components reduce to single vertices and the cost of the solution is zero. Hence, the overall computational complexity is $O(n^4 p)$.

Theorem 5 *The min-sum MGGPP on spider graphs can be solved in $O(n^4 p)$ time.*

The left-hand side of Figure 3 shows a spider with 8 vertices and 3 legs. Let us focus on the subproblem determined by the pair of vertices (v_5, v_7), with range $[10, 20]$. The central component certainly includes v_0, v_5 and v_7. Vertex v_6 is the only grey vertex, whereas all vertices from v_1 to v_4 are white. First, notice that for $p \leq 2$ and $p \geq 7$, the subproblem is infeasible; in fact, the white vertices cannot form less than two connected subgraphs, while the white and grey vertices cannot form more than five subgraphs. The right-hand side of Figure 3 shows the auxiliary graph for $p = 4$: this consists of a dummy node u_0 and $p - 1 = 3$ layers of four nodes. Each layer corresponds to a subgraph different from the central one, and each node on a layer corresponds to one of the four white vertices. The minimum cost path from u_0 to

$u_{4,3}$ identifies the optimal partition: v_1 and v_4 are singletons (with zero gap), v_2 and v_3 form a subgraph with gap equal to 5, the other vertices form the central subgraph, with gap equal to 10.

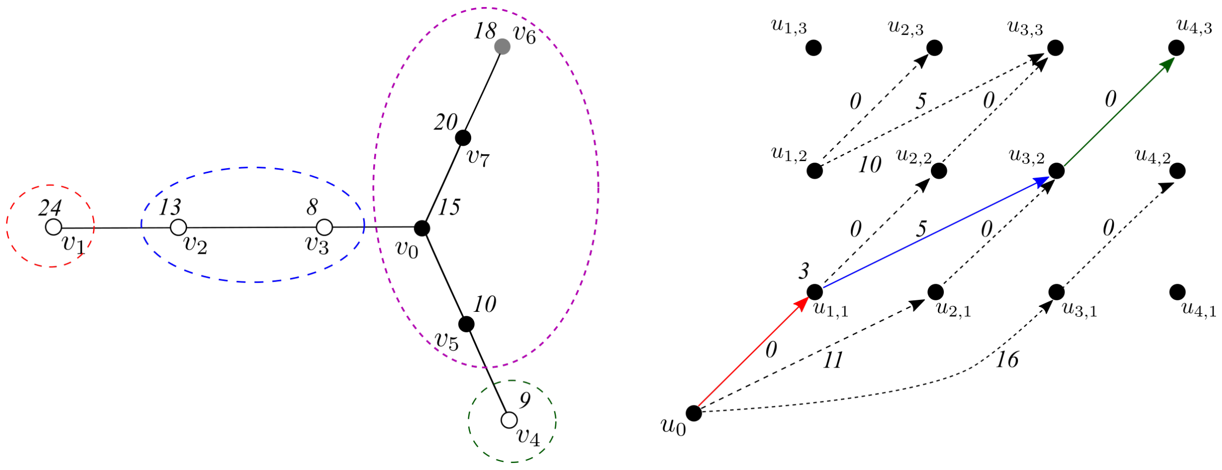

Figure 3: Construction of the auxiliary graph to partition a spider graph into $p = 4$ subgraphs with v_5 and v_7 as the vertices of minimum and maximum weight in the central component

A caterpillar graph is a tree which consists in a central path and, possibly, some leaves adjacent to it. For this kind of trees, the $O\left(n^2 p\right)$ approach adopted for paths can be extended for the min-sum version of MGGPP. In fact, in any connected partition of G a leaf will be either in the same component of its unique adjacent node on the central path, or it will form a singleton component. We number progressively the $n_c < n$ vertices of the central path as $v_1, ..., v_{n_C}$. We denote by l_{ij} the number of leaves adjacent to vertices $v_i, \ldots, v_j$ and by $n_L = n - n_C$ the total number of leaves. To solve MGGPP in a caterpillar graph, we exploit the construction of an auxiliary graph similar to the one already provided for a path. A detailed description of the construction of the auxiliary graph is provided in Appendix 2, implying the following result.

Theorem 6 *When G is a caterpillar the min-sum MGGPP can be solved in $O\left(n^3 \log n + n^2 p^2\right)$ time.*

We observe that according to the procedure described in Appendix 2, the computational complexity result of Theorem 6 applies also for solving the min-max MGGPP on a cartepillar graph, but such complexity is dominated by the one stated in Theorem 1 for general trees.

5 A variant of the minimum gap partitioning problem on graphs

In this section we investigate a version of MGGPP in which an additional constraint is introduced so that in a feasible partition singleton components are forbidden (in short, MGGPP2). We survey some special cases considered above and discuss the differences implied by the new constraint.

5.1 Paths

The $O(n \log n)$ time algorithm quoted in Theorem 2 for the min-max MGGPP on paths cannot be directly applied to the new version because properties P1 and P2 introduced in Section

2 do not hold any longer: if the minimum number of subpaths $q(\gamma)$ with gap not exceeding the threshold γ is strictly smaller than p, it is not always possible to split some of them in order to obtain a partition with exactly p components. In the following, we show that the $O(n \log n)$ time algorithm in [7] can actually be applied when the number of components is sufficiently small ($p \leq \lfloor n/3 \rfloor$), while a slightly less efficient algorithm can be designed for the remaining cases. The problem is given by the $q^o(\gamma)$ components of odd cardinality, that cannot be divided exactly into pairs of vertices; denoted as n_r the cardinality of component r, in fact, $\sum_{r=1}^{q(\gamma)} \lfloor n_r/2 \rfloor = (n - q^o(\gamma))/2$. However, since the odd components have at least three vertices, $q^o(\gamma) \leq \lfloor n/3 \rfloor$ and $(n - q^o(\gamma))/2 \geq n/3$. Therefore, as long as $p \leq n/3$, the solution of the basic algorithm can be turned into a feasible solution with exactly p subpaths.

When $p > n/3$, the algorithm must be modified to cope with the additional constraint that at most $n - 2p$ subpaths have odd cardinality. The basic algorithm scans the path and divides it removing an edge and starting a new subpath every time the threshold γ is going to be violated. The modified algorithm builds the subpaths in this way, but, when their cardinality is odd, it also stores the longest subpath of even cardinality which respects the threshold. This provides two possible initial vertices for the next subpaths. Dynamic programming allows to avoid an exponential explosion of the partitions. Indeed, denoting by $J_{r,q}$ the index of the last vertex of the r-th subpath in an optimal partition with $q \leq r$ odd-cardinality subpaths, the following recurrent equations hold:

$$J_{r,q} = \begin{cases} 0 & \text{if } r = 0 \text{ and } q = 0 \\ J_{r-1,0} + \mathrm{CardEvenSp}(J_{r-1,0}, \gamma) & \text{if } r > 0 \text{ and } q = 0 \\ \max(J_{r-1,q} + \mathrm{CardEvenSp}(J_{r-1,q}, \gamma), & \\ \qquad J_{r-1,q-1} + \mathrm{CardOddSp}(J_{r-1,q-1}, \gamma)) & \text{if } r > 0 \text{ and } q > 0 \end{cases} \tag{7}$$

where CardEvenSp(j, γ) and CardOddSp(j, γ) are the cardinalities of the longest even and odd subpath starting from vertex v_{j+1}, respectively. In the end, the original problem admits a solution which respects the threshold γ if and only if $J_{p,q} = n$ for at least one value $q \in \{0, \dots, n - 2p\}$, that is, if it is possible to partition the whole path into p subpaths, such that at most $n - 2p$ have odd cardinality. Since q ranges from 0 to $n - 2p$ and the procedures CardEvenSp$(\cdot)$ and CardOddSp$(\cdot)$ amount to a visit of the path for each value of s, we can conclude that the complexity of the algorithm is $O(n^2)$, which is repeated for each possible value of γ, leading to an overall time complexity of $O(n^2 \log n)$.

Theorem 7 *The min-max MGGPP2 on paths can be solved in $O(n \log n)$ time for $p \leq n/3$ and in $O(n^2 \log n)$ time for $p > n/3$.*

We observe that for the min-sum MGGPP it suffices to consider the auxiliary graph $\mathcal{G}$ introduced for paths in Section 4.1, and to remove the arcs that correspond to singletons, that is arc $(u_0, v_{1,1})$ and all arcs $(u_{i,r-1}, u_{i+1,r})$ for $i = 1, \dots, n_C - 1$ and $r = 2, \dots, p$. The optimal path from u_0 to u_{np} identifies the optimal solution satisfying the cardinality constraint and can be detected with the same worst-case asymptotic complexity, that is in $O\left(n^2 p\right)$ time (see [7]).

5.2 Spiders

The $O\left(n^4 p\right)$ algorithm proposed for the min-sum MGGPP can be extended by taking into account that neither the central component nor the subpaths obtained partitioning the legs of the spider should reduce to singletons. The first condition can be directly imposed. The second one requires some modifications to the auxiliary directed graph. On the one hand, the arcs representing singletons should be removed; on the other hand, grey vertices should be explicitly represented as they could be necessary to form feasible subgraphs. Therefore, $p-1$ nodes (one for each layer) are included also for each grey vertex, following the same order indicated above: for increasing values of the leg index and from each leaf to the central vertex. Every node is linked by arcs to all following nodes associated with the same leg. The nodes associated with the last white vertex and with all grey vertices of each leg have outgoing arcs that reach all nodes associated with the next leg. These arcs represent subsets which actually include only the vertices of the final leg and their costs are computed accordingly. This accounts for the fact that the grey vertices can also be included in the central component. For the same reason, if a leg is fully composed of grey vertices, additional arcs will reach the nodes of all following legs, up to the first one with white vertices. The optimal path from u_0 to the last node of the last layer, $u_{n',p}$, identifies the optimal partition. For example, if in Figure 4 we consider the central component identified by $u = v_3$ and $v = v_0$, vertices v_2, v_4 and v_5 are grey and all other vertices are white. The auxiliary graph for $p = 3$ consists of a dummy node u_0 and $p-1 = 2$ layers of 6 nodes. The path marked on the right, made up by arcs $(u_0, u_{2,1})$ and $(u_{2,1}, u_{7,2})$, represents the two remaining subgraphs: one induced by v_1 and v_2, with a gap equal to 11, and one induced by v_6 and v_7 (since the leg with $k = 2$ is fully composed of grey vertices), with a gap equal to 2. When the number of legs with white vertices is too large the problem becomes infeasible.

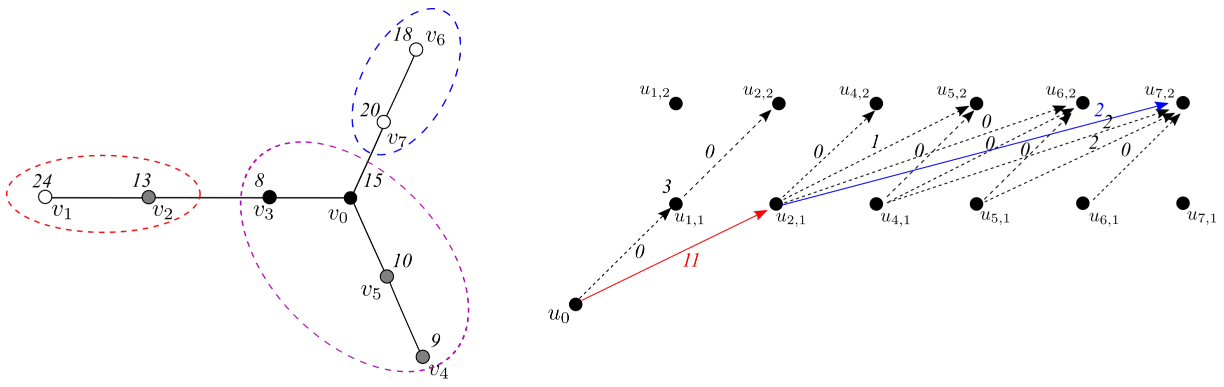

Figure 4: Construction of the auxiliary graph to partition a spider graph into $p = 3$ subgraphs of cardinality at least 2, with v_3 and v_0 as the vertices of minimum and maximum weight in the central component

Theorem 8 *The min-sum MGGPP2 on spiders can be solved in $O(n^4 p)$ time.*

The min-max MGGPP2 can be solved in a similar way, testing with a binary search all possible thresholds γ, enumerating the $O(n^2)$ central components that respect such a threshold, appending the white and grey vertices to build a single path, and finding whether it is possible to partition this path into $p-1$ subpaths with gap $\leq \gamma$ with the algorithm described in Section 4.1. This algorithm must be extended to account for the fact that vertices of different legs must belong to different subpaths. This additional constraint can be easily accommodated in the procedures CardEvenSp($\cdot$) and CardOddSp($\cdot$). The resulting algorithm has time complexity equal to $O(n^4 \log n)$.

Theorem 9 *The min-max MGGPP2 on spiders can be solved in $O(n^4 \log n)$ time.*

6 Complexity on general graphs

In this section, we prove that the *min-max* MGGPP is *strongly* $\mathcal{NP}$-hard. As well, the *min-sum MGGPP2* is *strongly* $\mathcal{NP}$-hard, whereas the complexity of the *min-sum* MGGPP is still open. Some preliminary results can be found in [5] although without the necessary technical details. Hence, for the sake of completeness we report them in this section.

Theorem 10 *The* min-max *MGGPP and* MGGPP2 *are strongly* $\mathcal{NP}$*-hard even if* $p = 2$.

Proof

The decision version of the considered problem amounts to determine whether a given instance admits a solution such that all subgraphs have a gap not larger than a given threshold. The problem is obviously in $\mathcal{NP}$: given a certificate that specifies, for each vertex, whether it belongs to V_1 or V_2, it takes polynomial time to verify whether the two induced subgraphs are connected and their gaps do not exceed the threshold.

The proof of $\mathcal{NP}$-hardness is by reduction from *SAT*. Given a generic instance of *SAT* with m logical clauses C_j depending on Boolean variables x_i ($i = 1, \ldots, n$), we build the following auxiliary graph. For each literal, we introduce a vertex (v_i or $\bar{v}_i$) with $w_{v_i} = w_{\bar{v}_i} = 2$; for each clause, we introduce a vertex c_j with weight $w_{c_j} = 1$; finally, we introduce two dummy vertices v_0 and v_f with weight $w_0 = w_f = 3$. Vertex v_0 is connected to v_1 and $\bar{v}_1$; vertex v_f is connected to v_n and $\bar{v}_n$; each vertex v_i is connected to v_{i+1} and $\bar{v}_{i+1}$ ($i = 1, \ldots, n-1$) and to all the clause vertices c_j such that literal x_i occurs in clause C_j; finally, each vertex $\bar{v}_i$ is connected to v_{i+1} and $\bar{v}_{i+1}$ ($i = 1, \ldots, n-1$) and to all the clause vertices c_j such that literal $\bar{x}_i$ occurs in clause C_j. The threshold for the objective function is set to 1 and the number of subgraphs to 2: we are therefore looking for $p = 2$ connected subgraphs with gaps γ_1 and γ_2 not larger than 1. Figure 5 shows the auxiliary graph corresponding to the Boolean formula $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3)$.

All solutions in which vertices v_0 and v_f belong to different subgraphs have a gap larger than 1, because assigning the clause vertices to any of the two subgraphs would yield a gap equal to 2, which is strictly larger than the threshold. Thus, v_0 and v_f must belong to the same subgraph, which cannot include any clause vertex c_j. To be connected, this subgraph must contain a path between v_0 and v_f, consisting only of vertices v_i or $\bar{v}_i$. By construction, this path must contain at least one of the two vertices associated with each variable x_i. This subgraph has a gap equal to 1. The remaining subgraph must contain all the clause vertices c_j and must be connected. If such a subgraph exists, its gap is equal to 1. Moreover, its vertices v_i and $\bar{v}_i$ correspond to a subset of literals which identify a truth assignment. Such a truth assignment is consistent because at most one vertex for each pair $(v_i, \bar{v}_i)$ is allowed, and satisfies all logical clauses because each clause vertex c_j is connected to the other clause vertices through a vertex v_i or $\bar{v}_i$. In Figure 5, the edges marked in continuous lines provide a feasible solution that corresponds to the satisfying truth assignment $x_1 = x_2 =$ true, $x_3 =$ false.

Vice versa, any satisfying truth assignment identifies a partition of the graph into two subgraphs with the required maximum gap.

Note that requiring a gap ≤ 1 forbids singletons by construction, although we do not require it explicitly; in fact, in such a solution one subgraph must necessarily include vertices v_0 and v_f, and the other subgraph must include all the clause vertices. Therefore, the proof covers also the MGGPP2. $\square$

Corollary 1 *The* min-max MGGPP *and* MGGPP2 *cannot be approximated for any constant* $\alpha < 2$ *unless* $\mathcal{P} = \mathcal{NP}$.

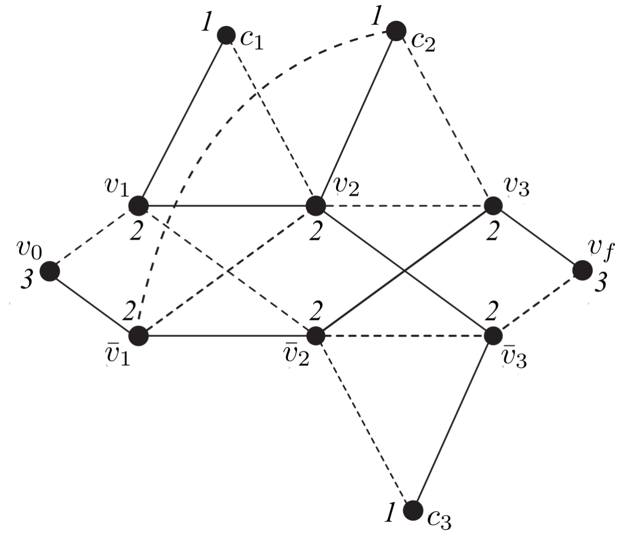

Figure 5: Graph construction to prove the strong $\mathcal{NP}$-hardness of the min-max MGGPP and *MGGPP2*: the auxiliary graph corresponds to the *CNF* $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3)$ with three clauses and three logical variables. The feasible solution formed by the edges drawn in continuous lines corresponds to the truth assignment $x_1 = x_2 =$ true, $x_3 =$ false.

Proof

By contradiction, let us assume the existence of a polynomial algorithm with a constant approximation guarantee $\alpha < 2$. The graph construction employed in the proof of Theorem 10 allows to build an instance of both problems with an optimum equal to 1 for any YES-instance of *SAT* and an instance of both problems with an optimum equal to 2 for any NO-instance. When applied to the former, the postulated approximation algorithm would return a solution with a value < 2, that is 1 (by integrality), thus recognizing it as a YES-instances and solving *SAT* in polynomial time. $\square$

Theorem 11 *The* min-sum MGGPP2 *is strongly $\mathcal{NP}$-hard, even if $\eta_V = 2$.*

Proof

The decision version of this problem, which is obviously in $\mathcal{NP}$, amounts to verifying the existence of a solution such that the total gap of all subgraphs is not larger than a given threshold.

The proof of $\mathcal{NP}$-hardness is by reduction from 3-*SAT*. Given a generic instance of 3-*SAT*, we build the following auxiliary graph. For each clause C_j, we introduce a *clause vertex* c_j with $w_{c_j} = 1$ and a triplet of *occurrence vertices*, o_{j1}, o_{j2}, o_{j3} with zero weight; these four vertices are reciprocally connected. For each variable x_i, we introduce a *variable vertex* x_i with $w_{x_i} = 1$, and four *literal vertices* v_i, v'_i, $\bar{v}_i$ and $\bar{v}'_i$ with zero weight. Vertex v'_i is connected to v_i, vertex $\bar{v}'_i$ to $\bar{v}_i$, vertex x_i is connected to v_i and $\bar{v}_i$. Each occurrence vertex is connected to the vertex v_i or $\bar{v}_i$ associated with the corresponding literal. Finally, if m is the number of clauses and n the number of variables, we ask for a solution to the *MGGPP2* where the number of subgraphs is $p = m + 2n$ and the threshold on the total gap is n. Figure 6 shows the graph corresponding to $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3)$.

By construction, any feasible solution consists of s_0 subgraphs of zero gap and s_1 subgraphs of unitary gap. As $s_0 + s_1 = m + 2n$ and $s_1 \leq n$ (because of the threshold), necessarily $s_0 \geq m + n$.

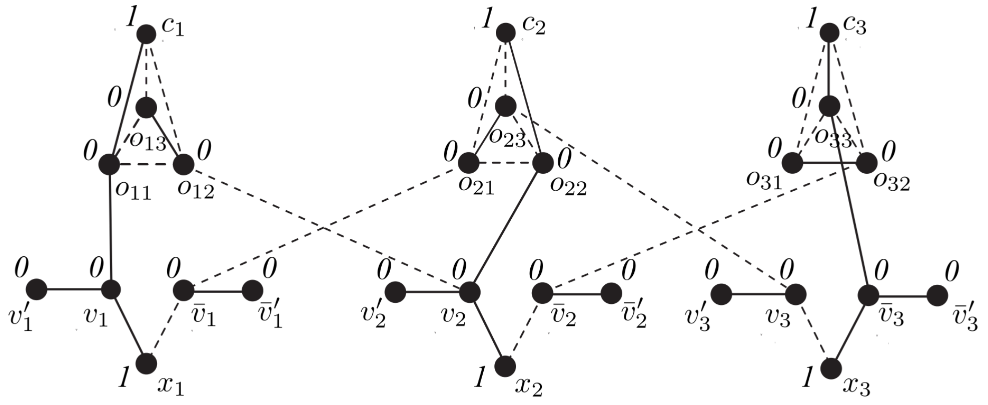

Figure 6: Graph construction to prove the strong $\mathcal{NP}$-hardness of the *min-sum MGGPP2*: the auxiliary graph corresponds to the *CNF* $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3)$ with three clauses and three logical variables. The feasible solution formed by the edges drawn in continuous lines corresponds to the truth assignment $x_1 = x_2 =$ true, $x_3 =$ false.

The subgraphs of zero gap contain only occurrence or literal vertices, and at least $2m + 2n$ vertices overall (at least two vertices each). The subgraph containing c_j must contain at least an adjacent occurrence vertex and the subgraph containing x_i must contain at least v_i and v'_i, or $\bar{v}_i$ and $\bar{v}'_i$. Thus, at most $2m$ out of $3m$ occurrence vertices and at most $2n$ out of $4n$ literal vertices are available for the subgraphs of zero gap, which is exactly the minimum required cardinality: therefore every feasible solution contains exactly $m+n$ subgraphs of zero gap, with exactly two vertices each. By construction, n of these subgraphs consist of two adjacent literal vertices and m consist of two adjacent occurrence vertices. The remaining n subgraphs have unitary gap. Each contains at most one variable vertex: otherwise, the subgraph should include two adjacent occurrence vertices, but each triplet of occurrence vertices has a single remaining available vertex (since the other two occurrence vertices are used for the aforementioned m subgraphs of zero gap). So, there are exactly n subgraphs of unitary gap, each one with a single variable vertex. The clause vertices are necessarily included in these subgraphs since their weight is 1 while all the other subgraphs contain only vertices of weight 0. Such vertices are connected to the subgraphs of unitary gap through an occurrence vertex and a literal vertex that identifies a satisfying truth assignment. Since the complementary literal vertices belong to the zero gap subgraphs, the truth assignment is consistent. In Figure 6, the edges marked in continuous lines provide a feasible solution that corresponds to the satisfying truth assignment $x_1 = x_2 =$ true, $x_3 =$ false.

Vice versa, any satisfying truth assignment identifies a partition of the graph into $m+n$ subgraphs of zero gap and n with gap equal to 1. $\square$

The following theorem shows that the MGGPP with $p = 2$ includes two families of instances: the former can be solved exactly in polynomial time, whereas the latter has the property that the value of every feasible solution cannot be worse that twice the optimal one.

Theorem 12 *The MGGPP and the* MGGPP2 *with both the* min-max *and the* min-sum *objective function are* 2*-approximable in* $O(\eta_V(m+n))$ *time for* $p = 2$.

Proof

Let V_1^* and V_2^* be the unknown subsets of vertices of the optimal solution. Considering the ranges of the weights in the two subgraphs, $\left[m_{V_1^*}, M_{V_1^*}\right]$ and $\left[m_{V_2^*}, M_{V_2^*}\right]$, there are two possible cases: either the two ranges are separate or overlapping.

In the former case, all the vertices in a subgraph have weights strictly smaller than those in the other, and therefore the two ranges can be represented as $\left[m_V, w^{(k)}\right]$ and $\left[w^{(k+1)}, M_V\right]$ for a suitable value of index $k \in \{1, \dots, \eta_V - 1\}$. Then, the optimal solution can be found by exhaustively considering all values of k from 1 to $\eta_V - 1$ and checking every time whether the subgraphs induced on G by the vertices whose weights fall in the two intervals are connected. For the *MGGPP2*, we must also verify that the two subgraphs are not singletons. This procedure takes $O\left(\eta_V \left(m + n\right)\right)$ time overall, because the connectivity can be verified in $O\left(m + n\right)$ time.

In the case of overlapping ranges, $f_{MS}^* = \gamma_{V_1^*} + \gamma_{V_2^*} \geq \gamma_V$, which implies $f_{MM}^* = \max\left(\gamma_1^*, \gamma_2^*\right) \geq \gamma_V/2$. Every feasible solution enjoys the trivial property that $f_{MS} \leq 2\gamma_V$ and $f^{MM} \leq \gamma_V$. Consequently, $f_{MS} \leq 2 f_{MS}^*$ and $f_{MM} \leq 2 f_{MM}^*$ for every feasible solution. □

7 Conclusions and further research

In this paper we analyzed a special class of range-based partitioning problems introduced in [13]. What differentiate our contribution w.r.t. the one presented in [13] is that we study range-based partitioning problems with connectivity constraints on special classes of graphs.

For the first time we provided a polynomial time algorithm for the min-max version of this problem on trees. We also specialized our techniques to some special classes of trees. The complexity status of the min-sum version of MGGPP on general trees remains open. This problem is currently being studied and it will be the topic of a future work.

Acknowledgements

The research of the fourth and fifth author was partially supported by the Spanish/FEDER Grant Number MTM2016-74983-C2-1-R and by the research program supported by Junta de Andalucia, Consejeria de Economia y Conocimiento n. US-1256951.

Appendix 1 - Detecting the central component in spiders

Given a spider graph G and its central vertex v_0 of degree d, we root G at v_0 and perform a binary search on the possible values of γ solving each time problem MGGPP(γ). For a given γ, we denote by $P_1, \dots, P_d$ the legs of G, P_k being the path starting from leaf k, with $k = 1, \dots, d$, and ending at v_0. Given a feasible solution, we denote by $SP_r(P_k)$, $r = 1, \dots, n_k$, the n_k subpaths in which leg P_k is partitioned; in particular, we denote by g_k the one containing v_0, that is, $SP_{n_k}(P_k)$. The central component of the partition of the spider, C_{v_0}, should contain the central vertex v_0 and a subset of the subpaths g_k, $k = 1, \dots, d$.

To find the partition of each P_k, $k = 1, \dots, d$, we apply the $O\left(n \log n\right)$ time algorithm described in [7] to each of the d paths connecting the leaves of G to the root by visiting G bottom-up. For a given γ, for each path P_k we consider all its components but the one including v_0 ($n_k - 1$ components, $k = 1, \dots, d$). In total, we have $\sum_{k=1}^{d} \big(n_k - 1\big)$ components. Clearly, if $\sum_{k=1}^{d} \big(n_k - 1\big) \geq p$, it is impossible to find a partition in exactly p components with gap less than or equal to γ. Otherwise, one has to find the central component of the partition formed by a subset of the subpaths g_k, $k = 1, \dots, d$. In principle, it is possible that g_k, $k = 1, \dots, d$, cannot be merged altogether in the central component without exceeding γ. Therefore, one has to merge in the central component as many subpaths g_k as possible. At the end, those among $g_1, \dots, g_d$ that cannot be merged are disconnected from the central vertex and the total number of components so obtained is computed in order to check whether it is greater than p (no solution to MGGPP(γ) exists), or it is smaller than p (a solution to MGGPP(γ) can be found by further dividing some components).
In order to find C_{v_0}, let m_k and M_k be the minimum and maximum vertex weight respectively in the subpath g_k, $k = 1, \dots, d$. We first observe that two subpaths g_h and $g_{h'}$ can be combined in a component if and only if

$$\max\{M_h, M_{h'}\} - \min\{m_h, m_{h'}\} \leq \gamma.$$

We suppose, without loss of generality, that $m_h \neq m_{h'}$ for all pairs h and h', $h \neq h'$. In fact, if two paths g_h and $g_{h'}$ have $m_h = m_{h'}$ they can be merged in a unique component having maximum equal to $\max\{M_h, M_{h'}\}$ and gap less than or equal to γ.

The idea of the algorithm is to find for each h the maximum number of paths that can be merged in a unique component having minimum weight equal to m_h and gap at most γ. An index h^* that maximizes the number of combined paths provides the central component C_{v_0}. In the following we describe how h^* can be found in $O(n \log n)$ time.

We consider the two sets $\{m_1, m_2, \dots, m_d\}$ and $\{M_1, M_2, \dots, M_d\}$ and we order their elements in nondecreasing order. For the sake of simplicity, we suppose that the set $\{m_1, m_2, \dots, m_d\}$ is already ordered, i.e. $m_h < m_{h+1}$ for $h = 1, \dots, d-1$. Let $\{\overline{M}_1, \dots, \overline{M}_d\}$ be the set maxima in non decreasing order and let $p(k)$ be the index of the path having $\overline{M}_k$ as maximum.

Property 1 *By construction, all subpaths g_k, $k = 1, \dots, d$ contain v_0. Let w_{v_0} be the weight of v_0, then:*

$$m_d \leq w_{v_0} \leq \overline{M}_1$$

We will say that m_h is *consistent* with $\overline{M}_k$, and viceversa, if $\overline{M}_k - m_h \leq \gamma$. Notice that if m_h is not consistent with $\overline{M}_k$ then g_h and $g_{p(k)}$ cannot be combined; otherwise, if m_h is consistent with $\overline{M}_k$ and $m_{p(k)} > m_h$ then g_h and $g_{p(k)}$ can be combined. This is not necessarily true if $m_{p(k)} < m_h$.

Property 2 *If m_h is consistent with $\overline{M}_k$ then m_h is consistent with $\overline{M}_{k'}$ for $k' < k$.*

For each h, let $K(h)$ be the largest index k such that m_h is consistent with $\overline{M}_k$, i.e.

$$K(h) = \max_{k=1,\dots,d}\{k : \overline{M}_k - m_h \leq \gamma\}$$

By Property 2, we have:

$$K(h) \leq K(h+1),\ h = 1,\dots,d-1.$$

It follows that it is possible to find $K(h)$ for all h by scanning the sets $\{m_1, m_2, \dots, m_d\}$ and $\{\overline{M}_1, \dots, \overline{M}_d\}$ simultaneously with different indices h and k, respectively. For a fixed h, $h = 1,\dots,d$, $K(h)$ is determined by scanning $\overline{M}_k$ from $k = K(h-1)$ (or $k = 1$ if $h = 1$) and checking whether m_h is consistent with $\overline{M}_k$ until the value of $K(h)$ is found.

Thanks to the above discussion and to the following result, once the minima and the maxima have been ordered in $O(n \log n)$ time, $K(h)$ has been determined for all h in $O(n)$ time, the maximum number of paths that can be combined in a unique component having minimum weight equal to m_h and gap at most γ can be computed in constant time for each h.

Proposition 3 *For each $h = 1,\dots,d$, the maximum number of paths that can be combined in a component having minimum weight equal to m_h and gap at most γ is equal to $K(h) - h + 1$.*

Proof

The thesis is a consequence of the following facts that hold for each $h = 1,\dots,d$.

- $K(h)$ is equal to the number of $\overline{M}_k$ that are consistent with m_h.
- A path $g_{h'}$, $h' > h$, can be combined to g_h if and only if m_h is consistent with the maximum of $g_{h'}$.
- The $h-1$ paths $g_{h'}$, $h' < h$, have maximum weight less than or equal to $\overline{M}_{K(h)}$. Hence $h-1$ must be subtracted from $K(h)$.

□

Appendix 2 - Building the auxiliary graph in caterpillars

The auxiliary graph includes a source node u_0 and p nodes $u_{j,1},\dots,u_{j,p}$ for each vertex v_j on the central path. The arcs connect u_0 to each node $u_{j,s}$, $j = 1,\dots,n_C$ and $s = 1,\dots,\min(p, l_{1j}+1)$, and each node $u_{i,r}$ to each node $u_{j,s}$ with $i, j \in \{1,\dots,n_C\}$ and $r, s \in \{1,\dots,p\}$ such that $i < j$ and $r < s \leq r + \min(p, l_{ij}+1)$. Arc $(u_0, u_{j,s})$ represents s subgraphs of the final partition, given by $s-1$ isolated leaves and the subpath of the central path $\{v_1,\dots,v_j\}$ with the remaining $l_{1j}-s+1$ leaves. Arc $(u_{i,r}, u_{j,s})$ represents $s-r$ subgraphs of the final partition, given by $s-r-1$ isolated leaves and the subpath of the central path $\{v_i,\dots,v_j\}$ with the remaining $l_{ij}-s+r+1$ leaves. The cost of an arc is the gap of the subpath, since the singletons have zero gap. Since there are multiple feasible choices for the leaves to be cut off from the subpath, we have to solve an auxiliary problem consisting in selecting which leaf must be cut. This is exactly the min-sum MGGPP on star graphs that can be solved by applying the $O(n \log n)$ time algorithm provided in [7]. In fact, the subpath can be collapsed into a single vertex with a range of weights rather than a single weight. This vertex can be considered as the central vertex of a star, whose leaves coincide with those of the original subpath. Since computing $\gamma(V_i)$ for all central components defined in Section 4.1 can still be done in linear time, the dominating term of the computational time is still given by the leaves ordering, that is $O(n_L \log n_L)$. Figure 7 shows such construction for the partition into $p = 3$ components of a caterpillar with $n = 7$ vertices overall and $n_C = 4$ vertices on the central path. The missing arcs represent collections of subgraphs that cannot be obtained from the given graph; for example, arc $(u_0, u_{3,3})$ does not exist because the central subpath $\{v_1, v_2, v_3\}$ has a single leaf ($l_{13} = 1$), and therefore it is not possible to obtain $s = 3$ subgraphs by cutting leaves. Computing the arc costs requires $O\left(n_C^2 n_L \log n_L\right)$ time, thanks to the following procedure. First, all arcs with the same indices i, j and the same difference $s - r$ have the same gap, as they represent the same collection of $s - r$ subgraphs. Then, given i and j, the gap for a single subgraph ($s - r = 1$) can be computed in $O\left(l_{jj}\right)$ time, adding vertex j and its l_{jj} adjacent leaves to the vertex set of the previous subgraph identified by indices i and $j - 1$. The gaps for the other values of $s - r$, from 2 up to $\min\left(p, l_{ij} + 1\right)$, can be computed sorting the weights of the adjacent leaves (in $O\left(n_L \log n_L\right)$ time) and cutting $s - r - 1$ times the leaf of minimum or maximum weight to obtain the required number of subgraphs. Since $\sum_{j=1}^{n_C} l_{jj} = n_L$, the sorting operations dominate and the computation of the arc costs takes $O\left(n_C^2 n_L \log n_L\right)$ time.

The optimal *min-sum* or *min-max* path from u_0 to $u_{n_C,p}$ identifies the optimal solution. For example, the optimal min-sum solution is given by arcs $(u_0, u_{3,2})$ and $(u_{3,2}, u_{4,3})$, where $(u_0, u_{3,2})$ represents the optimal pair of subgraphs obtained cutting $s - 1 = 1$ leaf from the subgraph identified by vertices $(v_1, \ldots, v_3)$, while $(u_{3,2}, u_{4,3})$ represents the part of the caterpillar identified by vertex v_4 and all its adjacent leaves (no leaf must be cut off, because $s - r - 1 = 2 - 1 - 1 = 0$). Detecting the optimal path takes linear time with respect to the number of arcs, i.e. $O\left(n_C^2 p \min\left(p, n_L\right)\right)$,

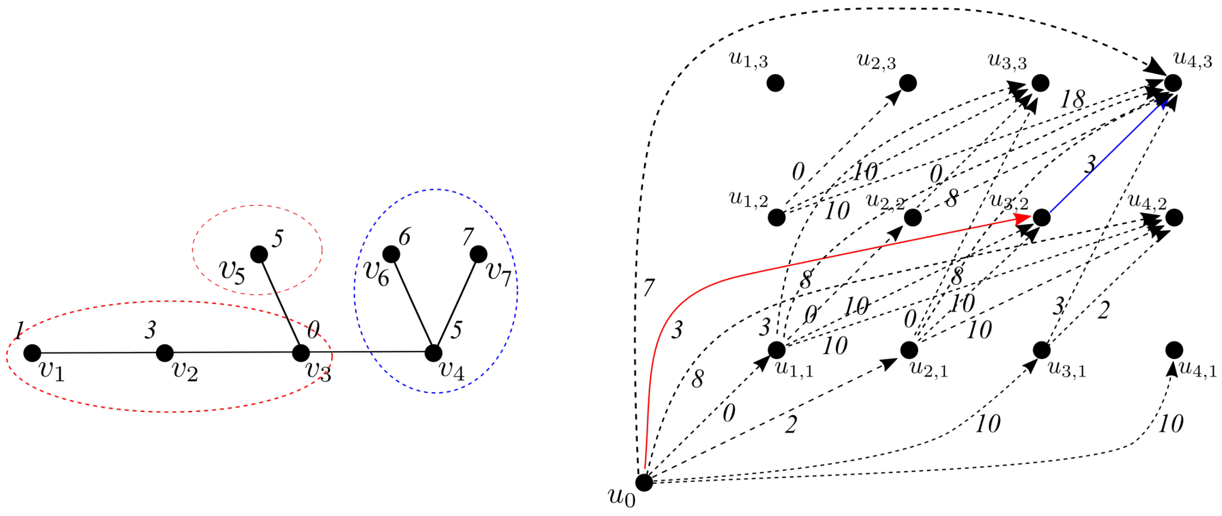

Figure 7: Construction of the auxiliary graph for a caterpillar.

References

[1] N. Apollonio, I. Lari, J. Puerto, F. Ricca, B. Simeone. Polynomial algorithms for partitioning a tree into single-center subtrees to minimize flat service costs. *Networks*, vol. 51, 78–89, 2008.

[2] D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner (eds.). *Graph partitioning and graph clustering*, v. 588 of *Contemporary Mathematics.* AMS, 2013.

[3] Li Xiao, Li Hongpeng, Niu Dongling, Wang Yan, and Liu Gang. Optimization of GNSS-controlled land leveling system and related experiments. *Transactions of the Chinese Society of Agricultural Engineering*, vol. 31, 48–55, 2015.

[4] V.M. Albornoz, M.I. Véliz, R.P. Ortega, and V. Ortìz-Araya. Integrated versus hierarchical approach for zone delineation and crop planning under uncertainty. *Annals of Operational Research*, vol. 286, 617–634, 2020.

[5] M. Bruglieri, R. Cordone. Partitioning a graph into minimum gap components. *Electronic Notes in Discrete Mathematics*, vol. 55, 33–36, 2016.

[6] M. Bruglieri, R. Cordone, V. Caurio. A metaheuristic for the minimum gap Graph Partitioning Problem. *Proceedings of the 15th Cologne-Twente Workshop (CTW2017) on Graphs and Combinatorial Optimization*, pp. 23–26, Cologne, Germany, June 6-8 2017.

[7] M. Bruglieri, R. Cordone, I. Lari, F. Ricca, A. Scozzari. Some polynomial special cases for the Minimum Gap Graph Partitioning Problem. *Proceedings of the 16th Cologne-Twente Workshop (CTW2018) on Graphs and Combinatorial Optimization*, pp. 107–110, Paris, France, June 18–20 2018.

[8] F. De Paola, N. Fontana, E. Galdiero, M. Giugni, G. Sorgenti degli Uberti, M. Vitaletti. Optimal design of district metered areas in water distribution networks. *Procedia Engineering*, vol. 70, 449–457, 2014.

[9] C. De Simone, M. Lucertini, S. Pallottino, B. Simeone. Fair dissections of spiders, worms, and caterpillars. *Networks*, vol. 20, 323–344, 1990.

[10] M. Garey, D. Johnson. Computers and intractability: A guide to the theory of $\mathcal{NP}$-completeness. New York, NY, USA: W.H. Freeman and Co, 1979.

[11] R. Gomes, J. Sousa, J. Muranho, A. Sá Marques. Different design criteria for District Metered Areas in Water Distribution Networks. *Procedia Engineering*, vol. 119, 1221–1230, 2015.

[12] P.L. Hammer, B. Simeone. Order relations of variables in 0-1 programming, Surveys in combinatorial optimization, S. Martello, G. Laporte, M. Minoux, and C.C. Ribeiro (Editors), Vol. 31, *Annals of Discrete Mathematics*, 1987, pp. 83–112.

[13] D.S. Hochbaum. Algorithms and complexity of range clustering. *Networks*, vol. 73, 170–186, 2019.

[14] D.S. Hochbaum, B. Fishbain, J. E Baumler, G. E. Gold. Automated and semi-automated knee cartilage segmentation using graph-cuts. *Manuscript UC Berkeley*, 2009.

[15] I. Lari, M. Maravalle, B. Simeone. Computing sharp bounds for hard clustering problems on trees. *Discrete Applied Mathematics*, vol. 157, 991–1008, 2009.

[16] I. Lari, J. Puerto, F. Ricca, A. Scozzari. Uniform and most uniform partitions of trees. *Discrete Optimization*, vol. 30, 96–107, 2018.

[17] I. Lari, J. Puerto, F. Ricca, A. Scozzari. Partitioning a graph into connected components with fixed centers and optimizing cost-based objective functions or equipartition criteria. *Networks*, vol. 67, 69–81, 2016.

[18] I. Lari, J. Puerto, F. Ricca, A. Scozzari. Algorithms for uniform centered partitions of trees. *Electronic Notes in Discrete Mathematics*, vol. 55, 37–40, 2016.

[19] M. Maravalle, B. Simeone, R. Naldini. Clustering on trees. *Computational Statistics & Data Analysis*, vol. 24, 217–234, 1997.

[20] A. Mirzaian, E. Arjomandi. Selection in X+Y and matrices with sorted rows and columns. *Information Processing Letters*, vol. 20, 13–17, 1985.

[21] J.C. Picard. Maximal closure of graph and applications to combinatorial problems, *Management Science*, vol. 22, 1268–1270, 1976.

[22] M. Richey. Optimal location of a path or tree on a network with with cycles. *Networks*, vol. 20, 391–407, 1990.

[23] M. Richey, A. Punnen. Minimum perfect bipartite matchings and spanning trees under categorization. *Discrete Applied Mathematics*, vol 39, 147–153, 1992.