European Journal of Operational Research xxx (xxxx) xxx

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

A branch-and-price procedure for clustering data that are graph connected

Stefano Benati[a], Diego Ponce[b], Justo Puerto[b,*], Antonio M. Rodríguez-Chía[c]

[a] Dipartimento di Sociologia e Ricerca Sociale, Università di Trento, Italy
[b] IMUS and Departamento de Estadística e Investigación Operativa, Universidad de Sevilla, Spain
[c] Departamento de Estadística e Investigación Operativa, Universidad de Cádiz, Spain

ARTICLE INFO

Article history:
Received 28 May 2020
Accepted 25 May 2021
Available online xxx

Keywords:
Combinatorial optimization
Clustering
Mixed integer programming
Branch-and-price

ABSTRACT

This paper studies the Graph-Connected Clique-Partitioning Problem (GCCP), a clustering optimization model in which units are characterized by both individual and relational data. This problem, introduced by Benati, Puerto, and Rodríguez-Chía (2017) under the name of Connected Partitioning Problem, shows that the combination of the two data types improves the clustering quality in comparison with other methodologies. Nevertheless, the resulting optimization problem is difficult to solve; only small-sized instances can be solved exactly, large-sized instances require the application of heuristic algorithms. In this paper we improve the exact and the heuristic algorithms previously proposed. Here, we provide a new Integer Linear Programming (ILP) formulation, that solves larger instances, but at the cost of using an exponential number of variables. In order to limit the number of variables necessary to calculate the optimum, the new ILP formulation is solved implementing a branch-and-price (B&P) algorithm. The resulting pricing problem is itself a new combinatorial model: the Maximum-weighted Graph-Connected Single-Clique problem (MGCSC), that we solve testing various Mixed Integer Linear Programming (MILP) formulations and proposing a new fast "random shrink" heuristic. In this way, we are able to improve the previous algorithms: The B&P method outperforms the computational times of the previous MILP algorithms and the new random shrink heuristic, when applied to GCCP, is both faster and more accurate than the previous heuristic methods. Moreover, the combination of column generation and random shrink is itself a new MILP-relaxed matheuristic that can be applied to large instances too. Its main advantage is that all heuristic local optima are combined together in a restricted MILP, consisting in the application of the exact B&P method but solving heuristically the pricing problem.

1. Introduction

We consider a clustering problem in which units are characterized by both individual and relational data. Individual data take the form of a matrix F of n rows, representing units, and m columns, representing features that are measured for individuals. Individual data are then complemented by relational data, for example representing friendship, communication, co-participation and so on. Relational data are described as an undirected graph $G = (V, E)$ in which V are the units, $|V| = n$, and there is an edge $e_{ij} \in E$ if and only if there is a relation between $i, j \in V$. The data structure that combines the graph $G = (V, E)$ with the data matrix F forms the triplet $G = (V, E, F)$, called *attributed graph*, see Bothorel, Cruz, Magnani, and Micenkova (2015).

The simplest method of clustering attributed graphs is projecting the relational data into the individual data, or vice versa, the individual into the relational. In the former case, a dissimilarity measure between units i and j is calculated using *both* individual measures of F *and* the existence/non-existence of an arc i, j, Combe, Largeron, Egyed-Zsigmond, and Géry (2012), Cheng, Zhou, Huang, and Yu (2012). In the latter case, the matrix F is used to calculate a distance d_{ij} attached to an existing arc e_{ij}, and to convert the *unweighted* graph into a *weighted* one, Neville, Adler, and Jensen (2003). In both cases, the problem is reduced to standard clustering or graph partitioning problems respectively, and any solution methods for those problems can be applied. The interested reader is referred to Gambella, Ghaddar, and Naoum-Sawaya (2021) for a recent survey. Alternatively, the two data structures are kept separate and then one can formulate an optimization model to determine the best classification. The optimization model

* Corresponding author.
E-mail addresses: stefano.benati@unitn.it (S. Benati), dponce@us.es (D. Ponce), puerto@us.es (J. Puerto), antonio.rodriguezchia@uca.es (A.M. Rodríguez-Chía).

https://doi.org/10.1016/j.ejor.2021.05.043
0377-2217/

must be formulated in such a way that it takes into account that relational data give additional information about the similarity between units. That is, the objective function or the constraints set must reflect some connectivity requirement. In Benati, Puerto, and Rodríguez-Chía (2017), clustering with graph-connected units is modeled as a combinatorial problem in which the most similar groups are evaluated through the clique partition of individual data, namely, induced by the information in F but with the additional constraint that those cliques must be additionally connected through the underlying graph G, representing the relational data. The problem so formulated has been called the Graph-Connected Clique-Partitioning Problem (GCCP). Benati et al. (2017) shows that this model is superior than classical clustering methods in finding true clusters.

More formally, the GCCP consists in the following. An attributed graph $G = (V, E, F)$ is given, so that the similarity/dissimilarity distances c_{ij} between all pairs i, j can be calculated using only the information contained in F, see Benati et al. (2017) for further details on its computation. These c_{ij} are used to formulate the objective function of a clique partitioning problem as done in Grötschel and Wakabayashi (1989). Relational data E are used imposing that the optimal clique partition $\Pi = \{V_1, \ldots, V_p\}, 1 \leq p \leq n,$ must be composed of components $V_k \subseteq V, k = 1, \ldots, p,$ *connected* trough the arcs of E. Some empirical experiments have shown that combining the two data sources through this model improves the clustering quality.

In Benati et al. (2017), exact and heuristic methods are proposed to solve the GCCP. Exact solutions are calculated though different MILP models. Those models differ on how they impose connectivity through a set of linear constraints. Connectivity can be imposed through flow conservation laws, or using constraints describing the forest/tree decomposition, and models can be strengthened with valid inequalities. Some formulations are more advantageous than others, but, in all cases, only problems of moderate size can be solved exactly. Various implementations of local search heuristics are tested as well, but, even though those methods find reasonably accurate solutions, their computational times are high. Therefore, it is worth exploring the possibility of improving on these previous findings.

In this paper we are proposing three new methods, one exact procedure and two heuristics, to solve the GCCP. The exact method is based on a branch-and-price algorithm (B&P), a technique that has been proved successful when applied to other clustering problems, Mehrotra and Trick (1998), Aloise et al. (2010). The interested reader can also see Lübbecke and Desrosiers (2005), Gualandi and Malucelli (2013), and the references therein to gain further insight into column generation techniques.

The first step of the algorithm is to formulate GCCP as a Set Partitioning (SP) problem. In the SP, a binary variable y_S is defined for every feasible subsets $S \subseteq V,$ and then a set of linear constraints defines the feasible solutions. Obviously, the straight solution of the model is impeded by the exponential number of variables, $O(2^n),$ but actually there is no need to consider them all just from the beginning. Rather, one can start with a MILP formulation including only a few of the y_S's, solve the problem, and then adding new variables only after the result of the reduced cost test. The reduced cost test relies on the exact or heuristic solution of a new combinatorial problem, the Maximum-weighted Graph-Connected Single-Clique problem (MGCSC). We formulate the MGCSC as a MILP model, testing the effectiveness of various formulations. Moreover, as it is important to find a solution quickly, thus a fast, greedy-like constructive heuristic has been developed, inspired by the noising method proposed in Charon and Hudry (2006). As a result, it has been found that this heuristic can be applied to the GCCP as well, providing a faster and more accurate algorithm than local search heuristics. In addition, a MILP-relaxed matheuristic procedure is developed that combines the quickness of previously described heuristic with the accuracy of the column generation developed for the exact method. We refer the reader to Raidl (2015) and the references therein for alternative successful combinations of column generation and heuristics. Finally, we found that our implementation of B&P, the heuristic and matheuristic approaches developed in this paper are respectively improvements of the previous exact and heuristics solution procedures as they calculate faster their respectively optimal or approximate solutions.

The paper is structured in seven sections, the first being this introduction. In Section 2, we provide a formal definition of the problem and its formulation as a SP with an exponential number of variables. In Section 3, we discuss the pricing problem consisting of a new combinatorial problem, the MGCSC, so we discuss how to calculate its optimal solution. In Section 4, we describe a fast heuristic for an approximate solution of both GCCP and MGCSC, based on greedy, but enhanced through the use of some random steps. Also a MILP-relaxed matheuristic, capable of handling very large instances with good accuracy, is proposed. Section 5 is devoted to describing some details of the B&P which are not included in Section 2 for the ease of compactness. In Section 6, we report our computational analysis, comparing the exact methods to solve the GCCP by B&P through different formulations of the pricing problem and testing the performance of the heuristics too. The paper ends with some remarks on future research directions.

2. Problem definition and set partitioning formulation

In this section, we formally define the GCCP. Let $V = \{1, \ldots, n\}$ be a set of units and $C = (c_{ij})_{i,j \in V}$ a measure of similarity/dissimilarity between units, with $c_{ij} < 0$ denoting similarity, dissimilarity otherwise. Assume that units of V are embedded in a graph $G = (V, E),$ whose edges $e_{ij} \in E$ describe links between $i, j \in V$. Given $Q \subseteq V,$ let $G[Q] = (Q, E[Q])$ be the subgraph induced by $Q,$ i.e., the graph with edges $e_{ij} \in E[Q]$ iff $i, j \in Q$ and $e_{ij} \in E$. We say that $Q \subseteq V$ is connected if $G[Q] = (Q, E[Q]),$ i.e., the subgraph induced by $Q,$ is a connected subgraph.

The goal of GCCP is to find a partition $\Pi = \{V_1, \ldots, V_p\}$ of V (with parameter p not fixed in advance, i.e., $1 \leq p \leq n$), such that any $V_k, k = 1, \ldots, p,$ is connected and minimizing the objective function:

$$f(\Pi) = \sum_{k=1}^{p} \sum_{i,j \in V_k} c_{ij}.$$

Hence, GCCP can be formulated as follows:

$$\min_{\Pi \in \mathcal{P}} \; f(\Pi)$$
$$\text{s.t.} \;\; V_k \text{ is connected for all } V_k \in \Pi,$$

where $\mathcal{P}$ is the set of all the partitions of V.

As GCCP is in minimization form, units i and j for which c_{ij} is negative will tend to be in the same group, while units for which c_{ij} is positive will tend to be in different groups. Introducing a connection constraint between units implies that even though a unit can be similar to several others, it can be clustered only to the connected units.

In Benati et al. (2017), GCCP has been formulated and solved with exact and heuristic methods. Exact methods are some MILP formulations based on the Clique Partition problem with connection constraints. Heuristic methods are the improved local search heuristics Variable Neighborhood Search (VNS) and Random Restart (RR). In this paper, we introduce a new MILP formulation with an exponential number of variables that will be solved through column generation, embedded in a branch-and-price algorithm. Next we introduce two new heuristic procedures: A con-

structive heuristic based on random shrink and a MILP-relaxed matheuristic based on approximated column generation. All new methods are improvements over the old ones, as the exact method improves the computational times and the maximum size of the solved instances, while the heuristics improve the optimum approximation for a given computational time.

2.1. The set partitioning formulation

In this section, a new formulation of GCCP is introduced, in which an exponential number of variables are needed. Suppose that we can list all connected subsets S of V: Let $\mathcal{S} = \{S \mid S \subseteq V, G[S] \text{ is connected}\}$ and let $c_S = \sum_{i\in S}\sum_{j\in S: j>i} c_{ij}$. Let y_S be a binary variable defined for all $S \in \mathcal{S}$ such that:

$$y_S = \begin{cases} 1, & \text{if } S \in \Pi, \\ 0, & \text{otherwise.} \end{cases}$$

Hence, GCCP can be formulated as follows:

$$(\mathbf{MP}) \ \min \sum_{S\in\mathcal{S}} c_S y_S$$
$$s.t. \sum_{S\in\mathcal{S}: i\in S} y_S = 1, \quad \forall i \in V,$$
$$y_S \in \{0,1\}, \quad \forall S \in \mathcal{S}.$$

The problem constraints ensure that a unit is included in exactly one cluster, so that subsets S must form a partition Π. The value of a partition is given by the problem objective function. The drawback of (MP) is that it contains an exponential number of binary variables to explicitly define $\mathcal{S}$. Hence, we consider its restricted version. The idea is to formulate (MP) with only a fraction of the y_S variables. Then, solving its linear relaxation, we can obtain reduced costs for the absent variables y_S and determine whether a new variable/column y_S is to be introduced in the relaxed and restricted (MP), or the current solution is optimal for that problem. Branching is applied each time a not integral solution is found until optimality is proved. The reader is referred to the following works and the references therein for further details on the following topics: Desrosiers and Lübbecke (2005), for a precise presentation about column generation; Barnhart, Johnson, Nemhauser, Savelsbergh, and Vance (1996) for a detailed explanation about branch-and-price; and to Deleplanque, Labbé, Ponce, and Puerto (2020), for a recent application of those techniques. A pseudocode of this method is provided in Algorithm 1 and explained in detail in Section 5.

2.2. Relaxed restricted master problem

Here we explain the solution procedure of the relaxed master problem at the root node. The same procedure is applied in the remaining nodes. The particularities involved in the solution of branched nodes can be found in Section 5.

Let $\mathbb{S} \subseteq \mathcal{S}$ be a subset of all the feasible clusters. The relaxed and restricted master problem is:

$$(\mathbf{RelaxedMP})_{\mathbb{S}} \quad \min \sum_{S\in\mathbb{S}} c_S y_S \qquad \textbf{Dual Multipliers}$$
$$s.t. \sum_{S\in\mathbb{S}: i\in S} y_S = 1, \qquad \forall i \in V, \gamma_i \text{ unrestricted}$$
$$y_S \geq 0, \quad \forall S \in \mathbb{S}.$$

Observe that the dual multipliers associated with each constraint are emphasized in the right-hand side of the formulation above.

The dual of the relaxed and restricted master problem is

$$(\mathbf{DP})_{\mathbb{S}} \qquad \max \sum_{i=1}^{n} \gamma_i$$
$$s.t. \sum_{i\in S} \gamma_i \leq c_S, \quad \forall S \in \mathbb{S},$$
$$\gamma_i \text{ unrestricted}, \quad \forall i \in V.$$

Given an optimal solution γ^* of $(\mathbf{DP})_{\mathbb{S}}$, we can obtain the reduced cost of an absent variable y_S of the master problem as:

$$\tilde{c}(y_S) = c_S - \sum_{i\in S} \gamma_i^*.$$

If it can be proved that the reduced costs of all the missing variables are nonnegative, then the master problem is solved to optimality. Otherwise, any variable y_S with $\tilde{c}(y_S) < 0$ induces a new column to be included in $(\mathbf{RelaxedMP})_{\mathbb{S}}$ to (possibly) improve the incumbent solution. We refer to the pricing problem as the problem of finding a cluster $S \in \mathcal{S}$ such that $c_S - \sum_{i\in S} \gamma_i^* < 0$, or to prove that it does not exist. If, after solving the pricing problem, one or more new variables y_S are introduced in $(\mathbf{RelaxedMP})_{\mathbb{S}}$, then it is solved again. Otherwise, the relaxed master problem is solved to optimality.

Algorithm 1: B&P for GCCP.

Input: An instance of GCCP with data $C, G = (V, E)$.
Output: An optimal partition Π of V.

```
1  𝕊 ← Initiate(C, G)
2  optimality ← false
3  node ← root node
4  while optimality = false do
5      (γ*, y*) ← Solve((RelaxedMP)_𝕊, node)
6      S ← Solve_Pricing_Problem(γ*, node)
7      if c̃(y_S) < 0 then
8          𝕊 ← 𝕊 ∪ S
9      else
10         if y* integral then
11             if Upper_Bound(y*) then
12                 optimality ← true
13             else
14                 node ← Next_Node(MP)
15         else
16             if Lower_Bound(y*) then
17                 optimality ← true
18             else
19                 Branch(y*)
20                 node ← Next_Node(MP)
```

3. The pricing problem

Step 6 of Algorithm 1, i.e., the solution of the pricing problem, is an important step. The problem consists in answering the question "Is $c_S - \sum_{i\in S} \gamma_i^* < 0$ for some $S \in \mathcal{S}$?" To respond to the query we define a new combinatorial problem on the graph $G = (V, E)$ where inputs are the costs c_{ij} associated to each pair of nodes $i, j \in V$ and node weights $-\gamma_i^*$ for all $i \in V$. Then, the Maximum-weighted Graph-Connected Single-Clique (MGCSC) on G consists in: Given a graph $G = (V, E)$ with weights associated with each pair of nodes and each individual node, find a connected subset of V, minimizing the sum of both node weights and pairs-of-nodes weights. The reader should observe that in our application we solve minimization problems since weights can be positive and negative. This problem is related with the prize collecting Steiner tree problem, Ljubić et al. (2006), and the maximum

weight connected subgraph problem, Álvarez-Miranda, Ljubić, and Mutzel (2013), although in both cases the graph structure, weights and the objective function are different. The MGCSC reduces to the maximum-weighted clique problem when G is a complete graph, therefore the former is trivially $\mathcal{NP}$-hard, Balas, Chvátal, and Nesetril (1987).

The problem described above can be formulated in each pricing iteration at the root node of the master problem. As before, we leave the necessary branching modifications to Section 5.

In spite of its exponential worst-case complexity, Step 6 can be implemented in such a way to maintain an efficient computation. In fact, it is not necessary to find the optimal S, that is, to calculate the exact minimum $\bar{c}(y_S)$. It is sufficient to find any $S \in \mathcal{S}$ for which $\bar{c}(y_S) < 0$ (and even more than one of such S if possible). Therefore, we can solve MGCSC using a heuristic method and only when the heuristic fails, we calculate its exact solution. At the end of the algorithm, the exact solution of the pricing problem is surely needed to certify optimality in (MP), that is, proving that all missing S's are such that $\bar{c}(y_S) \geq 0$. Nevertheless, before that, hopefully a large amount of required variables are detected heuristically.

In the next sections, we propose some MILP models to solve MGCSC. The main differences among models are the type of constraints that impose connectivity.

3.1. Flow-based formulation

The idea behind this formulation is that if a set $S \subseteq V$ is connected, then a source node can send a unit of flow to any node of S using the auxiliary network induced by S. Let $G_D = (V, A)$ be a digraph with set of arcs, A, so defined: Two arcs (i, j) and (j, i) for every edge $e_{ij} (= e_{ji}) \in E$. For each subset $S \subset V$ one of its nodes is assumed to be a source and all the remaining nodes ask for a unit of flow that must be sent from that source. Then, an objective function is minimized with respect to a node set S, but constraints will try to establish a flow from the source to the nodes of S. If a flow is permissible, then those nodes are connected and S is feasible, so that y_S is a candidate variable/column for the restricted master problem. Although in principle, we may assume that any node of V could be the source, this would produce many symmetric solutions. They are broken imposing that, for any connected S, the only source within S is the largest index node.

For this formulation one needs flow variables f_{ij} defined for all pairs i, j such that $(i, j) \in A$. In addition, the following variables are required. For $i \in V$, the variable x_i is defined as:

$$x_i = \begin{cases} 1, & \text{if node } i \text{ is in the cluster,} \\ 0, & \text{otherwise.} \end{cases}$$

For any $i, j = 1, \ldots, n$ such that $i < j$, the variable z_{ij} is defined as:

$$z_{ij} = \begin{cases} 1, & \text{if nodes } i \text{ and } j \text{ are in the cluster,} \\ 0, & \text{otherwise.} \end{cases}$$

For any $(i, j) \in A$, the variable f_{ij} is defined as:

f_{ij} = amount of flow sent from node i to node j.

The flow-based formulation of MGCSC is:

$$(\mathbf{F_{flow}}) \ \min \sum_{i \in V} \sum_{j \in V: j > i}^{n} c_{ij} z_{ij} - \sum_{i=1}^{n} \gamma_i^* x_i \tag{1}$$

$$s.t. \ z_{ij} \leq x_i, \quad \forall i, j \in V : i < j, \tag{2}$$

$$z_{ij} \leq x_j, \quad \forall i, j \in V : i < j, \tag{3}$$

$$z_{ij} \geq x_i + x_j - 1, \quad \forall i, j \in V : i < j, \tag{4}$$

$$\sum_{i \in V: (i,k) \in A} f_{ik} - \sum_{i \in V: (k,i) \in A} f_{ki} \geq x_k + (n-2)(x_j - 1), \forall k, j \in V : j > k, \tag{5}$$

$$\sum_{j \in V: (i,j) \in A} f_{ij} \leq \sum_{j \in V: j < i} z_{ji} + \sum_{j \in V: i < j} z_{ij}, \quad \forall i \in V, \tag{6}$$

$$z_{ij} \geq 0, \quad \forall i, j \in V : i < j, \tag{7}$$

$$f_{ij} \geq 0, \quad \forall (i, j) \in A, \tag{8}$$

$$x_i \in \{0, 1\}, \quad \forall i \in V. \tag{9}$$

The objective function (1) accounts for the reduced cost. Constraints (2)–(4) are the usual inequalities of the Clique Partitioning problem to ensure that $z_{ij} = x_i x_j$. Constraints (5) are the flow conservation law, valid for all nodes of the cluster except for the node with the greatest index. This node is the source, so a flow of the cardinality of the cluster minus one can leave the node. Constraints (6) provide an upper bound of the outflow from any node $i \in V$, in addition, if this node does not belong to the cluster the right hand side of the constraints is 0, i.e., there is not outflow. Lastly, (7)–(9) define the domain of the variables.

An alternative formulation is given in Appendix; where an auxiliary node is considered as source node. That formulation is more natural and intuitive than the one given in this section, but it provides worse computational results. In spite of that, we decided to keep it in this manuscript because it can ease the understanding of the formulation in this section.

Formulation ($\mathbf{F_{flow}}$) can be strengthened with the families of valid inequalities described in B.1 in Appendix.

The minimum reduced cost is $\bar{c}_S = c_S - \sum_{i=1}^{n} \gamma_i^* x_i^*$, where $c_S = \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{ij} z_{ij}^*$. If $\bar{c}_S \geq 0$, then the linear relaxation of the master problem is optimal. Otherwise the column y_S, that is the incident vector of S, is introduced to the restricted master problem (see Step 8 of Algorithm 1).

3.2. Arborescence formulation

The rationale behind this formulation is that if a node set S is connected, then we can establish a directed spanning subtree using any node of S as the root, and assigning labels to all other nodes of S representing their corresponding positions in the ascending ordered sequence of distances from the root to the nodes. Those type of constraints are known as Miller-Tucker-Zemlin (MTZ) inequalities, introduced to solve the Traveling Salesman Problem in Miller, Tucker, and Zemlin (1960), and used in other routing problems, Laporte (1992), Gouveia (1996), Bektaş and Gouveia (2014); Landete and Marín (2014).

Let $G_D = (V, A)$ be an auxiliary network as defined in Section 3.1. The MTZ description of the Spanning Tree builds an arborescence rooted at the source node, and in which the arcs follow the direction from the root to the leaves: Binary variables t_{ij}, defined for every $(i, j) \in A$, will take value 1 if the arc $(i, j) \in A$ belongs to the arborescence, 0 otherwise. Then, continuous variables ℓ_i will indicate the position according to the distance from the root to node i in the ordered sequence of distances from the root to the nodes using only arcs of the arborescence. Binary variables x and z

are defined as in the previous formulation and, as before, to avoid symmetric optimal solutions, for any node set S only the node with the highest index can be the root.

Thus, the arborescence-based formulation of MGCSC is:

$$(\mathbf{F_{MTZ}}) \min \sum_{i\in V}\sum_{j\in V:i<j} c_{ij}z_{ij} - \sum_{i\in V}\gamma_i^* x_i$$

$$s.t.\ (2)-(4),(7),(9)$$

$$\ell_i + 1 \le \ell_j + n(1-t_{ij}), \quad \forall (i,j)\in A, \tag{10}$$

$$t_{ij} + t_{ji} \le z_{ij}, \quad \forall (i,j)\in A : i<j, \tag{11}$$

$$\sum_{i\in V:(i,k)\in A} t_{ik} \ge x_j + x_k - 1, \quad \forall k, j \in V : j>k, \tag{12}$$

$$t_{ij}\in\{0,1\}, \quad \forall (i,j)\in A, \tag{13}$$

$$\ell_i \in \mathbb{R}, \quad \forall i \in V. \tag{14}$$

Constraints (10) guarantee that the label assigned to node j is at least as great as the label assigned to node i when the arc $(i,j)\in A$ is chosen. Actually, these constraints only avoid cycles, but combined with constraints (11), they also exclude arcs incident to any node i not in S. Constraints (12) ensure that there is at least one arc incident to all the nodes of S (with the exception of the one with the highest index). Those arcs will form an arborescence, the nodes of the arborescence are the optimal connected component S. Finally, the domain of the variables is defined in (13) and (14).

Formulation ($\mathbf{F_{MTZ}}$) can be strengthened with the family of valid inequalities described in B.2 of the Appendix.

An alternative formulation, where an auxiliary node is used as source node, is presented in Section A.2 in the Appendix. The formulation ($\mathbf{F_{MTZ}}$) outperforms that formulation. Nevertheless, we decided to keep it in this manuscript because the former is more natural and intuitive.

3.3. Relaxations

If some of the models above are formulated without some "model constraints", then the resulting formulation will be referred to as "relaxation". Relaxations are solved faster, but of course, the solution can be unfeasible to the original model. The idea is to iteratively add constraints to the relaxation, hopefully not too many, from the removed constraints set to find a feasible solution of the original model. Relaxations have been coded in SCIP by implementing a constraint handler (Gleixner et al., 2018). We have explored the possibility of improving the computational times using two relaxations.

3.3.1. Clique relaxation

In the first relaxation, clique equations $z_{ij} = x_i x_j$, modeled by constraints (2)–(4), are discarded. Since this type of constraints involves binary variables and they are $O(n^2)$, all MILP problems can be solved faster without their explicit representation. Then, given an incumbent solution, a separation oracle tests by full enumeration whether it violates some clique inequality and if so, it is inserted into the MILP model. As it will be seen in the computational section, in some cases this strategy has obtained good results.

3.3.2. Connectivity relaxation

In the second relaxation, connectivity constraints of $\mathbf{F_{flow}}$ and $\mathbf{F_{MTZ}}$ are discarded (while retaining the clique constraints (2)–(4)). Suppose that an oracle determines that a node subset S is not connected because at least one pair $i, j \in S$ is not connected in $G[S]$. Then, if i, j should be in the same node subset, it should include at least one node out of S to be the *bridge* used to connect i and j. That is, for a given S and $i, j \in S$, the connectivity constraints are represented by:

$$\sum_{\ell=i+1:\ell\notin S}^{n} z_{i\ell} + \sum_{\ell=1:\ell\notin S}^{i-1} z_{\ell i} - z_{ij} \ge 0. \tag{15}$$

The formal proof of this result can be found as Theorem 2.1 in Benati et al. (2017). Note that the number of constraints (15) is exponential, but they can be separated efficiently. For a given element of a partition $S \subseteq V$ of an incumbent solution, consider the auxiliary complete graph G_S in which edge lengths are $l_{ij} = 0$ if $(i,j)\in E[S]$, $l_{ij} = 1$ otherwise. Let $LSP(i,j)$ be the shortest path distance from node i to node j (this can be computed by the Floyd–Warshall algorithm). If the maximum value of $LSP(i,j)$ for $i, j \in S$ is equal to 1, then subset S is not connected. The formal description of the separation procedure is described in Algorithm 2 .

Algorithm 2: Separation Algorithm.

Input: $G = (V, E)$, $(\bar{z}, \bar{x})$ a solution of connectivity relaxation, $S := \{k : \bar{x}_k > 0\}$.

Output: Violated cuts of the family (15).

1 **for** $i, j (i < j) \in S$ **do**

2 Compute $LSP(i,j)$ in the complete graph G_S with length of edges defined by:

$$l_{ij} := \begin{cases} 0, & \text{if } e_{ij}\in E[S], \\ 1, & \text{otherwise.} \end{cases}$$

if $LSP(i,j) > 0$ *(i and j are not connected in G[S])* **then**

3 Add the following inequality of family (15):

$$\sum_{\ell=i+1:\ell\notin S}^{n} z_{i\ell} + \sum_{\ell=1:\ell\notin S_k}^{i-1} z_{\ell i} - z_{ij} \ge 0. \tag{16}$$

4 **return:** *All violated cuts found from family (15).*

4. A shrinking-based and a MILP-relaxed matheuristic

In this section, two new heuristics for the GCCP are described. The first one, called *Random Shrink* (RS) heuristic, is a fast, constructive method to compute quickly an approximate solution. It is flexible enough to be applied to MGCSC problem as well, and in fact it is the heuristic that has been used to solve the pricing problem. The second heuristic is based on the approximated solution of the linear relaxation of (MP) in each node of the branch-and-bound (B&B) tree, in which the pricing problems are solved only through the RS heuristic. If the heuristic cannot find a negative reduced cost column, then the algorithm stops.

4.1. Random shrink heuristic

This section describes the first new heuristic algorithm devised to solve quickly both, GCCP and the MGCSC (with minimal modifications). Finding a feasible solution of the former problem is necessary in Step 1 of Algorithm 1, because the master problem must be initialized with a set of variables y_S, while solving the latter problem is necessary in Step 6 to find one or more new variables

y_S with negative reduced costs. As the algorithm is embedded in a B&P scheme, it must run quickly.

The new algorithm is based on the idea of shrinking the nodes of the graph $G = (V, E)$ in such a way that we have, in each iteration, a feasible partition, i.e., the elements of the partition are connected subsets as subgraphs on G. As a matter of fact, the GCCP data input is itself a partition, the one in which every singleton is a cluster. If two connected nodes are shrunk, the resulting graph will contain $|V| - 1$ nodes, but one node is actually containing two of the original, that is, the partition begins to have a structure. So, shrink can be repeated over and over, until a stopping criterion is satisfied.

More formally, shrink is the operation described in Algorithm 3. Input are the data structure $G^h =< V^h, E^h, c^h, \pi^h >$ and the node pair $i, j \in V^h$ with $e_{ij} \in E^h$, where: V^h is the active node set, each node representing a clique; E^h is the active edge set; c^h are the shrinking costs, defined for every pair $i, j \in V^h$ (when $c^h < 0$ it is actually a gain); π_i^h are the clique costs, defined for every active node $i \in V^h$. Furthermore, we define $f(V^h)$ as the objective function of partition V^h, $f(V^h) = \sum_{i \in V^h} \pi_i^h$.

Algorithm 3: Subroutine SHRINK.

Input: Data structure: $G^h =< V^h, E^h, c^h, \pi^h >$, the pair $i, j \in V^h$ with $e_{ij} \in E^h$.
Output: The data structure: $G^{h+1} =< V^{h+1}, E^{h+1}, c^{h+1}, \pi^{h+1} >$.

1 $V^{h+1} \leftarrow V^h \setminus \{j\}$
2 $E^{h+1} \leftarrow E^h \setminus \{e_{ij}\}$
3 $\pi_k^{h+1} \leftarrow \pi_k^h \; \forall k (\neq i, j) \in V^h$
4 $\pi_i^{h+1} \leftarrow \pi_i^h + c_{ij}^h$
5 $c_{k\ell}^{h+1} \leftarrow c_{k\ell}^h \; \forall e_{k\ell} \in E^{h+1}$
6 **for** $k \in V^h : e_{jk} \in E^h$ **do**
7 $\quad E^{h+1} \leftarrow E^{h+1} \cup (i, k) - (j, k)$
8 $\quad c_{ik}^{h+1} \leftarrow c_{ik}^{h+1} + c_{jk}^h$
9 $f(V^{h+1}) \leftarrow \sum_{k \in V^{h+1}} \pi_k^{h+1}$
10 **return** G^{h+1}

The output is a data structure $G^{h+1} =< V^{h+1}, E^{h+1}, c^{h+1}, \pi^{h+1} >$, in which $|V^{h+1}| = |V^h| - 1$. When pair $i, j \in V^h$ is shrunk, then j and (i, j) are deleted from nodes and edges respectively. Then, the clique costs π_i^h increases or decreases by cost c_{ij}^h, see Steps 3 and 4. All links and the costs of j are allocated to i, see Steps 5–8. Finally, the objective function $f(V^h)$ of the GCCP is updated in Step 9. Note that if we have to solve the MGCSC, then in Step 9 we can define $f(V^{h+1}) = \min_{i \in V^{h+1}} \{\pi_i^{h+1}\}$.

Before applying subroutine SHRINK in Algorithm 3, an edge $e_{ij} \in E^h$ must be elicited, but then, the choice can favor optimality or diversification. According to the *optimality criterion*, i and j must be such that the cost c_{ij}^h is minimum. In this way, if the cost is negative, shrinking i, j is the best decrease of the incumbent objective function $f(V^h)$. According to the *diversification criterion*, i and j can be selected randomly, but preferably the pair has been often assigned to different clusters in previous local optima.

The *Random Shrink* (RS) procedure is described in Algorithm 4 . Input data are an instance of GCCP, and parameters: *max_start* and *max_random_move*. At the beginning, every cluster is a singleton: $V^h = V$, $E^h = E$, $\pi^h = 0$, $f(V^h) = 0$. Then the graph is shrunk until a local optimum is found. In the first run, the method is greedy: Random moves are skipped, see Step 2. From the second round onwards, the first selections of pairs i, j, such that $e_{ij} \in E^h$, are random, see Steps 4–7. The number of random moves is itself random (drawn from a discrete uniform distribution from 1 to *max_random_move*), and depends on the input parameter *max_random_move*, see Step 3.

Algorithm 4: Random Shrink.

Input: The GCCP problem, max_start, max_random_move.
Output: The optimal partition: G^{best}.

1 **for** $s := 1$ **to** *max_start* **do**
2 $\quad$ **if** $s > 1$ **then**
3 $\quad\quad$ random_move = Unif(1,max_random_move)
4 $\quad\quad$ **for** $t := 1$ **to** *random_move* **do**
5 $\quad\quad\quad$ $e_{ij} \leftarrow$ Random_choice(W)
6 $\quad\quad$ $G^{h+1} \leftarrow$ Shrink(G^h)
7 $\quad\quad$ $h \leftarrow q$
8 $\quad$ fine := false
9 $\quad$ **while** *fine = false* **do**
10 $\quad\quad$ $e_{ij} \leftarrow \arg\min\{c_{ij}^h\}$
11 $\quad\quad$ **if** $c_{ij}^h < 0$ **then**
12 $\quad\quad\quad$ $G^{h+1} \leftarrow$ Shrink(G^h)
13 $\quad\quad\quad$ $h \leftarrow q$
14 $\quad\quad$ **else**
15 $\quad\quad\quad$ $G^{best} \leftarrow$ Update_Best(G^h)
16 $\quad\quad\quad$ $W \leftarrow$ Update_Weight(W)
17 $\quad\quad\quad$ fine := true
18 **return** G^{best}

The loop of Steps 9–17 is a standard greedy procedure, in which the best edge e_{ij} is selected in Step 10. The graph is shrunk if it provides an improvement of the objective function (Steps 11-13), otherwise, if necessary, the procedure updates the best solution so far (Steps 15–17). All is iterated *max_start* times, an input parameter, see Step 1. In every iteration, information about all local optima is stored in matrix W. The role of W is to lead the diversification: When implementing the random choice of e_{ij}, it is taken into account how many times an edge e_{ij} has been in local optima (that is, i and j were put into different clusters). The most it has been excluded from local optima, the highest is the probability of being selected randomly. To this purpose, when an egde e_{ij} is not in the local optimum E^h, then the value w_{ij} is augmented by one. When doing a random choice, the probability of choosing e_{ij} is $Pr[i, j] = w_{ij}/W$, with $W = \sum_{e_{ij} \in E} w_{ij}$.

4.2. A new MILP-relaxed matheuristic

The B&P described in Algorithm 1 can be readily modified to calculate an approximate solution instead of the optimum. It is sufficient to solve the pricing problems using only the RS heuristic, and never calculate the exact solution of the different MGCSC problems. Branching is still done to solve the master problem GCCP, as it is usually not much time consuming. In other words, (MP) is solved adding columns which empirically are tested to be useful, but not enough to certify optimality. In this way, GCCP is heuristically solved very quickly, but at the price of only solving approximately each linear relaxation of the master problem at any node of the B&B tree. In spite of that, as our computational results show, the quality of the solutions are rather good.

5. A branch-and-price implementation

In this section, we describe technical details of Algorithm 1, that were set aside so far for the sake of brevity. They are the generation of an initial solution, the branching rule, the Farkas pricing, and the convergence of column generation.

5.1. Starting solutions

Good starting solutions, that is, the initial clusters y_S's with their costs, are important to prune the searching tree. So, in this phase the RS algorithm is run with an abundant iteration limit and all local optima are used to define initial variables y_S's and feasible solutions of GCCP.

5.2. Ryan-and-Foster branching

Branching occurs when the LP solution of the master problem contains fractional variables. In B&P, it is not trivial to define a branching rule to resolve fractional solutions without fixing variables that were already in the pool of columns, Barnhart et al. (1996). Here, in Step 19 of Algorithm 1, the Ryan-and-Foster branching has been implemented, as it has considerable advantages over alternatives.

The Ryan-and-Foster (R-F) has been introduced to solve set partitioning problems, see Ryan and Foster (1981), and now is one of the most widespread techniques. If at a node of the master problem a solution contains fractional variables, the R-F rule creates two new branches as follows: Given two elements $i_1, i_2 \in V$, in one branch they will always be in the same cluster, whereas in the other branch they will always be in different cluster.

To implement this branching, we can take advantage of the x_i variables defined on the previous section for the pricing subproblem:

- **Left branch:** If i_1 and i_2 must be in different clusters implies that none of the variables corresponding to clusters containing i_1 and i_2 can assume positive values, i.e.,

$$\sum_{S \ni i_1, i_2} y_S = 0 \Rightarrow x_{i_1} + x_{i_2} \leq 1.$$

- **Right branch:** Since i_1 and i_2 must be in the same cluster then the following sum must be equal to 1:

$$\sum_{S \ni i_1, i_2} y_S = 1 \Rightarrow x_{i_1} = x_{i_2}.$$

In practice, when a new node is created (or candidate to be solved), existing y_S variables local bounds are modified according to the above constraints. These bounds are taken into account in Step 5 of Algorithm 1 when function Solve($(\textbf{RelaxedMP})_{\mathbb{S}}$, node) is called.

Furthermore, to solve the pricing problem, new variables not satisfying node requirements should be avoid. In function *Solve_Pricing_Problem*(γ^*, *node*) (Step 6 of Algorithm 1) we include the information of the ancestor nodes: $x_{i_1} + x_{i_2} \leq 1$ (left branch); and $x_{i_1} = x_{i_2}$ (right branch).

Finally, Steps 9–19 of Algorithm 1 work similar to the common B&B algorithm with some particularities of our R-F branching. When a fractional solution is found Branch(y^*) finds a pair $i_1, i_2 \in I$ for which

$$0 < \sum_{S \ni i_1, i_2} y_S < 1,$$

to create left and right nodes, using most fractional criterion. Lower_Bound(y^*) and Upper_Bound(y^*) update the lower and upper bound of (MP), respectively. Both functions return TRUE in case that bounds coincide, so (MP) is solved. Otherwise, Next_Node(MP) decides which is the next node to be studied. We let solver (SCIP, Gleixner et al., 2018) default-settings decide on the next node to be explored.

5.3. Farkas pricing

Another important element in any B&P algorithm is the so called Farkas pricing. This is the subroutine that provides new columns to the restricted master if it is locally infeasible.

We observe that infeasibility only can happen on a new node of the branching tree. If it happens because the R-F branching produces incompatible conditions, then the node is declared infeasible and no call to any pricing problem is necessary. Otherwise, the R-F conditions are compatible but perhaps not enough y_S variables are available in the pool to build a feasible solution. However, one can ensure *fictitious* feasible solutions by the following construction.

Proposition 5.1. *Assume that one initializes the pool of columns with all the singletons $y_{\{i\}}$ for all $i \in V$ and all pairs $y_{\{i,j\}}$ for all $i, j \in V$. Then, if the R-F branching leads to a node with compatible conditions GCCP is always feasible.*

Proof. The reader may note that if $e_{ij} \notin E$, we are augmenting in the initial pool a fictitious edge to E with cost $\hat{c}_{ij} = +M, \quad M \gg 0$. These variables always ensure *fictitious* feasible solutions of the restricted master problem (actually they may not be connected). Moreover, if it happens that in a node, one of those fictitious elements is used in a partition, it would represent an actual infeasible solution but it will never be optimal. □

In conclusion, the above result justifies that GCCP does not need a Farkas pricing routine.

5.4. Convergence of column generation

In column generation, it is well-known that the columns which certify optimality emerge at the last iterations of the procedure. This phenomenon has been studied and different solutions have been proposed in the literature to overcome it. Among others, du Merle, Villeneuve, Desrosiers, and Hansen (1999), Pessoa, Uchoa, Poggi, and Rodrigues (2010), and Sato and Fukumura (2012) have designed procedures to minimize the negative impact of the issue in the convergence of column generation algorithms. See also Sato and Izunaga (2019) or Deleplanque et al. (2020) for other recents applications of those techniques.

Those stabilization procedures are based on the principle that adding in each iteration the column with the best reduced cost may lead to convergence problems. In some way the conclusion of those papers is that the pricing problem optimal solution should be taken into account only in the latter iterations. Following that principle and basing on the results of Section 6.1, we solve the pricing problem heuristically for the first iterations. Hence, our algorithm stabilizes itself (Blanco, Japón, Ponce, & Puerto, 2021) as it is supported with our empirical results shown in Table 1.

6. Computational studies

Algorithms are tested on the instances previously used in Benati et al. (2017), and on new instances with greater size. The experiment layout is as proposed in Neville et al. (2003): Data are composed of n units on which m binary features, $F_i = \{0, 1\}, i = 1, \ldots, m$, are recorded. Units belong to one of two groups, each group is composed of $n/2$ units. If one unit belongs to group 1, then $\Pr[F_i = 1] = p_c$ for all $i = 1, \ldots, m$, otherwise, if the unit belongs to group 2, then $\Pr[F_i = 1] = 1 - p_c$ for all $i = 1, \ldots, m$. If p_c is close to one, then the two groups are well separated, as p_c gets closer to 0.5, the separation is less and less precise. Units are connected through arcs: If two units (or nodes) belong to the same group, then the probability of a joining arc is p_{in} (the probability of an inner arc). If the two nodes belong to two different groups, then the probability of a joining arc is p_{out} (the probability of an

Table 1
Average number of variables using the combined heuristic and exact pricers or only using the exact pricer for n =20,30,36 (the exact pricer uses two formulations: ($\mathbf{F_{flow}}$) and ($\mathbf{F_{MTZ}}$)).

	$n=20$				$n=30$				$n=36$			
Heurvar	Initial	Heur	Exact	Total	Initial	Heur	Exact	Total	Initial	Heur	Exact	Total
FALSE	51.5	0.0	27.6	**79.1**	79.4	0.0	69.5	**148.9**	98.3	0.0	164.9	**263.2**
TRUE	51.5	23.5	5.4	**80.4**	79.4	43.7	12.8	**135.9**	98.3	102.8	34.3	**235.4**
Variation				**+1.7%**				**−8.8%**				**−10.6%**

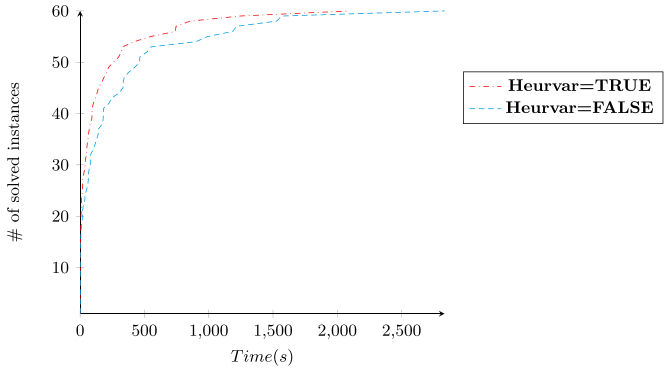

Fig. 1. Performance profile graph of #solved instances using the combined heuristic and exact pricers or only using the exact pricer for n =20,30,36 (the exact pricer uses two formulations: ($\mathbf{F_{flow}}$) and ($\mathbf{F_{MTZ}}$)).

outer arc). For the effect of the probabilities, the number X_{in} of vertices of the same group and the number X_{out} of vertices of the other group to which a given vertex $i \in V$ is connected are two random variables, with expected values $E[X_{in}] \approx np_{in}/2$ and $E[X_{out}] \approx np_{out}/2$. All experiments are run with $p_{in} > p_{out}$, so that connectivity provides information: If a node i, whose membership is uncertain, is connected with a node j that is known to belong to Group k, then it is likely that i belongs to k as well.

In all our computational experience, models are coded in C and solved with SCIP 6.0.1, Gleixner et al. (2018), using the optimization solver CPLEX 12.8 on an Intel(R) Core(TM) i7-4790 CPU @4.00 GHz 32 GB RAM. SCIP is a C library of subroutines specially devised to implement branch-cut-and-price and is distributed free-of-charge, under academic license, by the Zuse Institute Berlin (ZIB). We thank the SCIP team for the helpful technical advices in the course of this research.

6.1. Deciding the pricing problem implementation

From now on, we call heuristic pricer the application of any heuristic for solving the pricing problem. In case that the pricing problem is optimally solved, we call it exact pricer. First of all, we want to decide whether combining exact and heuristic pricers is worth. We have begun by analyzing the performance of Algorithm 1 for solving to optimality GCCP. For that reason, to test the usefulness of combining the heuristic and the exact pricers, we run a pilot study on instances of size 20, 30 and 36 nodes. We have compared two different implementations: One combining heuristic and the exact pricers, the other one only using the exact pricer. In addition, two versions of exact pricers have been tested as well, one using the Flow-based formulation, see Section 3.1, and the other using the Arborescence formulation, see Section 3.2. Models ($\mathbf{F^0_{flow}}$) and ($\mathbf{F^0_{MTZ}}$), see Appendix A, were discarded at an early stage of our computational experiments since preliminary results show that the use of the auxiliary node does not add any advantage concerning computational time.

Fig. 1 reports the results of the 60 instances tested for each implementation (three sizes, ten instances per size, and two formulations). It compares the number of solved instances versus time of Algorithm 1 using the Flow-based and the Arborescence formulations, and combining or excluding the heuristic pricer. One can observe that the combination of the exact and the heuristic pricer (line Heurvar=TRUE) is better than excluding the heuristic pricer (Heurvar=FALSE). These results suggest that solving the pricing problem combining the RS heuristic and any exact MILP is more efficient than using only MILP. Therefore, this is the strategy implemented to the largest instances too.

Concerning the tailing off effect of this heuristic pricer, Table 1 shows the number of necessary variables to certify optimality for instances of different size, depending on whether heuristic pricer is applied (*TRUE*) or not (*FALSE*). In this table, *Initial* is the average number of variables added from the beginning, *Heur* is the average number of variables added after the heuristic pricer interation, and *Exact* is the average number of variables added when the pricing problem is solved exactly.

When the heuristic pricer is applied, the problem is solved using a smaller number of variables. It means that the pricer heuristic not only saves computational time but also reduces degeneracy (that is, the situation in which reduced cost variables do not decrease the objective function). Furthermore, it can be seen that the impact is more remarkable for bigger instances.

6.2. Comparison of different formulations of MGCSC

We have continued our study solving (MP) using different alternatives for the pricing problem subroutine. In this set of experiments, as recommended by the above pilot study, Algorithm 1 has been run combining the RS heuristic, and using the different MILP formulations (see Section 3) to solve the pricing problem.

Five different MILP pricing routines are compared. The first two MILP models were ($\mathbf{F_{flow}}$) and ($\mathbf{F_{MTZ}}$). In the next three formulations, MILP are initialized without some constraints and/or variables, that are included whenever necessary to separate infeasible solutions only after that the separation subroutine is invoked. We refer to Flow Clique Relaxation and MTZ Clique Relaxation when clique constraints are removed (see Section 3.3.1) from Flow-based and Arborescence formulations, respectively. The fifth formulation, Connectivity Relaxation, removes the connectivity constraints (see Section 3.3.2).

The results reported in Table 2 are averages calculated after solving ten instances of each size, letting a maximum of 24 hours of computation. This table contains five blocks, one for each implementation of Algorithm 1. We report there the average solution time (Av.Time), the average gap at termination (Av.GAP), and the number of unsolved instances after the time limit is reached (Unsol). The best results in terms of times, gaps and number of unsolved problems are written in bold.

Remark 6.1. The lower bound used to calculate the gap at termination is given by the B&B process as usually. However, if the linear relaxation of the MP has not been solved at the time limit, another lower bound is still available, see Lübbecke and Desrosiers (2005). Particularly, for the GCCP the lower bound during the resolution of

Table 2
Average results for models with pricing problems based on formulations introduced in Section 3.

Model		($\mathbf{F_{flow}}$)			($\mathbf{F_{MTZ}}$)			Flow Clique Relaxation			MTZ Clique Relaxation			Connectivity Relaxation		
n		Av.Time	Av.GAP	Unsol	Av.Time	Av.GAP	Unsol	Av.Time	Av.GAP	Unsol	Av.Time	Av.GAP	Unsol	Av.Time	Av.GAP	Unsol
20		1.39	**0.00**	**0**	4.74	**0.00**	**0**	2.14	**0.00**	**0**	2.16	**0.00**	**0**	**1.25**	**0.00**	**0**
30		**34.71**	**0.00**	**0**	64.36	**0.00**	**0**	62.49	**0.00**	**0**	40.75	**0.00**	**0**	53.37	**0.00**	**0**
36		**419.81**	**0.00**	**0**	547.49	**0.00**	**0**	546.17	**0.00**	**0**	739.49	**0.00**	**0**	817.11	**0.00**	**0**
40		3545.18	**0.00**	**0**	1503.33	**0.00**	**0**	2731.25	**0.00**	**0**	**1444.89**	**0.00**	**0**	5319.05	**0.00**	**0**
50		18331.38	**0.00**	**0**	24820.23	14.03	2	**16634.83**	**0.00**	**0**	21320.21	1.49	2	40006.47	0.01	1
54		50646.86	1.03	**3**	**43684.52**	5.01	**3**	51861.38	1.30	**3**	44095.21	**0.66**	**3**	71221.69	4.12	7
60		81152.89	1.83	5	60697.51	28.28	6	80950.89	3.74	7	**56988.24**	**1.96**	**4**	86405.08	6.68	10
Total Result		**22018.89**	**0.41**	**8**	**18760.31**	**6.76**	**11**	**21391.74**	**0.71**	**10**	**17423.4**	**0.59**	**9**	**29117.72**	**1.54**	**18**

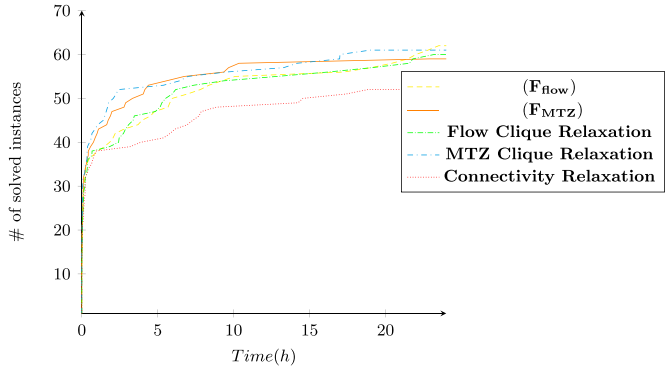

Fig. 2. Performance profile graph of #solved instances using different pricing problem formulations for n =20-60 (70 instances).

the root node is

$$LB = \sum_{i=1}^{n} \gamma_i^* + n \min_{S \in \mathcal{S}} \bar{c}(y_S),$$

provided that the last pricing problem has been solved exactly.

The results in this table point out that the best formulations are ($\mathbf{F_{flow}}$) and MTZ Clique Relaxation. The former solves 62 out 70 instances up to optimality, the latter solves 61 instances, that is, one problem less, but with a slightly smaller average computational time. Comparing Algorithm 1 with previous MILP methods, reported in Benati et al. (2017), we can observe that the maximum solved size has been improved from 40 to 60 units with the same time limit, and that computational times for solved instances have improved as well.

In Fig. 2, the results of Table 2 are summarized. Profiles show that pricing routines based on ($\mathbf{F_{MTZ}}$) and MTZ Clique Relaxation formulations are giving the best performance in terms of number of solved instances. However, Arborescence formulations ($\mathbf{F_{MTZ}}$) and the (MTZ Clique Relaxation) are giving the best solution times when the instances are solved. It can also be seen that removing the connectivity constraints (Connectivity Relaxation) does not work better than removing the clique constraints (Flow Clique Relaxation and MTZ Clique Relaxation). The latter do not solve 10 and 9 instances, respectively, and the former 18 instances.

To better understand the B&P algorithm performance, the reader can see in Table 3 different parameters computed as averages on ten instances: the gap at the root node (RootNodeGap(%)); the number of necessary variables (Total) split by the variables added at the beginning (Initial), obtained trough the heuristic pricer (Heur), and given by the exact pricer (Exact); the number of times that this latter routine is called (ExactIter); the nodes of the master problem branch-and-bound tree (Nodes); and the percentage of CPU time that the algorithm uses to solve the pricing problem (PricingTime(%)).

Remark 6.2. We have used multiple pricing in each iteration of the exact pricer. Therefore, in addition to the variable provided by the optimal solution of the pricing problem, we have added some other feasible solutions with negative reduced cost obtained during the solving process.

The results are shown until $n = 50$ to focus on those instances that were solved to optimality. We are presenting the results only for one of the five different formulations of the pricing problem, namely *Flow Clique Relaxation formulation*, since all the others exhibit similar results. The reader should note that the parameters which are analyzed refer to the MP rather that to the pricing problem and therefore the formulation used in the pricing problem is not very important to explain their behaviour.

Table 3
Branch-and-price performance.

n	RootNodeGap(%)	Total	Initial	Heur	Exact	ExactIter	Nodes	PricingTime(%)
$n = 20$	0.50	**80.70**	53.40	23.80	3.50	3.50	1.60	97.95
$n = 30$	0.13	**137.20**	81.60	44.30	11.30	6.10	1.20	99.91
$n = 36$	0.21	**234.40**	100.90	103.70	29.80	12.00	1.60	99.98
$n = 40$	0.00	**286.20**	108.50	126.10	51.60	18.00	2.10	99.99
$n = 50$	0.01	**443.40**	136.50	226.60	80.30	34.80	1.30	100.00

Table 4
Results on medium-sized problems for different heuristics.

Problem	fo[$\|V\|/3$]	it_b	fo[$\|V\|/2$]	it_b	fo[$2\|V\|/3$]	it_b	RR	VNS	Optimal Solution
G20_1	-114	1	-114	1	-114	1	-114	-114	-114
G20_2	-74	2	-74	53	-74	46	-74	-34	-74
G20_3	-122	12	-122	7	-122	10	-122	-122	-122
G20_4	-112	2	-112	4	-112	16	-112	-110	-112
G20_5	-128	1	-128	1	-128	1	-128	-102	-128
G20_6	-102	2	-102	93	-102	125	-102	-96	-102
G20_7	-154	11	-154	23	-154	23	-154	-102	-154
G20_8	-96	149	-94	188	-96	14	-94	-92	-96
G20_9	-116	1	-116	1	-116	1	-116	-116	-116
G20_10	-140	1	-140	1	-140	1	-140	-140	-140
n=20	**0.0 %**	**118.2**	**0.2 %**	**138.6**	**0.0 %**	**124.0**	**0.2 %**	**12.0 %**	
G30_1	-244	217	-248	293	-248	187	-254	-248	-254
G30_2	-152	33	-152	83	-152	22	-152	-152	-152
G30_3	-210	294	-206	197	-210	108	-200	-144	-210
G30_4	-200	191	-200	3	-192	57	-200	-170	-200
G30_5	-288	104	-288	87	-276	81	-288	-276	-288
G30_6	-260	29	-260	22	-260	13	-260	-260	-260
G30_7	-228	121	-228	25	-228	124	-228	-222	-228
G30_8	-126	29	-126	29	-126	22	-122	-108	-126
G30_9	-276	148	-274	66	-276	124	-276	-136	-276
G30_10	-174	57	-158	8	-174	14	-168	-154	-176
n=30	**0.5 %**	**155.2**	**1.5 %**	**136.6**	**1.2 %**	**132.6**	**1.2 %**	**13.3 %**	
G36_1	-296	61	-300	135	-296	134	-296	-296	-300
G36_2	-304	140	-304	254	-304	188	-300	-300	-304
G36_3	-390	36	-390	3	-390	329	-356	-340	-390
G36_4	-336	13	-336	3	-336	83	-326	-304	-340
G36_5	-300	57	-300	42	-300	237	-300	-300	-300
G36_6	-286	1	-286	1	-286	1	-286	-286	-286
G36_7	-344	324	-320	143	-318	131	-310	-324	-344
G36_8	-230	143	-240	78	-230	117	-230	-204	-246
G36_9	-268	85	-246	34	-242	4	-260	-242	-268
G36_10	-290	41	-290	121	-290	350	-290	-290	-296
n=36	**1.1 %**	**129.7**	**2.1 %**	**127.1**	**2.8 %**	**150.0**	**3.7 %**	**6.1 %**	
G40_1	-306	300	-290	164	-292	134	-294	-284	-318
G40_2	-514	107	-514	54	-514	196	-514	-508	-514
G40_3	-306	174	-300	70	-306	184	-288	-238	-332
G40_4	-406	333	-394	285	-384	46	-384	-384	-412
G40_5	-342	196	-342	372	-342	141	-326	-342	-342
G40_6	-336	204	-326	19	-296	99	-292	-252	-336
G40_7	-306	377	-280	42	-306	344	-272	-184	-330
G40_8	-294	175	-314	111	-286	5	-270	-252	-314
G40_9	-374	170	-386	329	-354	1	-396	-376	-396
G40_10	-444	304	-444	255	-444	201	-420	-372	-456
n=40	**3.5 %**	**164.6**	**4.6 %**	**145.4**	**6.4 %**	**136.8**	**8.5 %**	**15.9 %**	
mean	-249.70	116.15	-247.45	92.50	-245.40	97.88	-242.85	-224.40	-253.80
#solved	26		23		22		20	9	

Here we can see the strength of the B&P algorithm: it can solve the problem using very few variables comparing with other already proposed formulations to solve the GCCP, which are of the order of $O(n^2)$ (see Benati et al., 2017). The goodness of the root node gap makes the size of the B&B tree small. The counterpart is the time that the algorithm spends solving the pricing problem. To deal with it one has to save calls to the exact routine what is done by means of initial columns, the heuristic pricer, and adding several variables with negative reduced cost in each iteration.

6.3. Comparing different heuristic algorithms on GCCP

The previous computational section shows that heuristic procedures are still needed to solve the largest instances of the GCCP. Here, results about Algorithm 4 (RS) are reported. Parameter $\theta = max\_random\_steps$ has been fixed such that $\theta \in \{\lceil |V|/3 \rceil, \lceil |V|/2 \rceil, \lceil 2|V|/3 \rceil\}$, (the greater the number, the more the algorithm is driven by random choices of e_{ij}-s). The number of starting solutions is $max\_start = 10|V|$, the same number used in

Table 5
Results on moderately large-sized problems for different heuristics.

Problem	fo[$\|V\|/3$]	it_b	fo[$\|V\|/2$]	it_b	fo[$2\|V\|/3$]	it_b	RR	VNS	Optimal Solution
G50_1	-494	399	-510	316	-510	390	-492	-472	-562
G50_2	-636	273	-636	225	-642	31	-642	-642	-650
G50_3	-610	326	-630	49	-664	74	-576	-576	-674
G50_4	-488	232	-470	173	-464	15	-446	-450	-504
G50_5	-644	33	-644	131	-644	213	-644	-644	-644
G50_6	-358	99	-380	202	-400	35	-298	-264	-400
G50_7	-536	435	-532	186	-534	14	-498	-498	-564
G50_8	-614	10	-614	418	-614	32	-622	-502	-674
G50_9	-638	322	-636	3	-636	283	-614	-586	-642
G50_10	-462	288	-446	3	-446	232	-446	-448	-464
n=50	**5.2 %**	**142.8**	**4.9 %**	**129.9**	**3.9 %**	**123.1**	**9.3 %**	**12.6 %**	
G54_1	-790	80	-788	1	-788	1	-788	-694	-790
G54_2	-588	393	-580	524	-580	150	-596	-522	-662
G54_3	-542	67	-542	206	-542	26	-510	-496	-544[1]
G54_4	-560	146	-538	192	-544	382	-568	-446	-576
G54_5	-654	73	-654	513	-638	431	-650	-578	-670
G54_6	-568	61	-568	107	-568	80	-560	-564	-594
G54_7	-628	196	-624	105	-628	165	-614	-614	-640
G54_8	-606	181	-614	385	-606	104	-588	-578	-624
G54_9	-464	270	-450	261	-450	85	-394	-462	-490[1]
G54_10	-804	368	-766	406	-800	378	-808	-808	-808
n=54	**3.2 %**	**128.9**	**4.4 %**	**142.7**	**4.2 %**	**127.8**	**5.6 %**	**10.1 %**	
G60_1	-726	5	-766	142	-682	199	-660	-604	-782[1]
G60_2	-732	106	-732	168	-732	219	-636	-600	-732[1]
G60_3	-836	85	-834	48	-828	83	-758	-832	-832
G60_4	-680	8	-684	596	-690	382	-666	-558	-750
G60_5	-666	39	-650	467	-660	36	-626	-666	-712
G60_6	-938	51	-938	469	-938	193	-836	-788	-964
G60_7	-562	257	-538	424	-518	534	-502	-500	-606
G60_8	-650	288	-620	484	-624	500	-582	-658	-664
G60_9	-894	17	-894	233	-858	305	-832	-848	-912
G60_10	-622	325	-604	255	-636	342	-602	-510	-682[1]
n=60	**4.5 %**	**117.2**	**5.3 %**	**144.6**	**6.4 %**	**138.8**	**12.4 %**	**14.2 %**	
mean	-633.00	181.10	-629.40	256.40	-628.80	197.13	-601.80	-580.27	-660.40
# best solution	15		8		11		6	5	

[1] Best solution found by Algorithm 1

Benati et al. (2017) to test heuristics Variable Neighborhood Search (VNS) and Random Restart (RR). That is, all algorithms try to improve the same number of initial solutions. For the test, algorithms were coded in Julia version 1.03 (Bezanson, Edelman, Karpinski, & Shah, 2017) and run on HP EliteBook with a Intel I5-core CPU.

Computational results are contained in Tables 4, 5, 6 and 7, in which are reported the objective function and the first iteration (it_b) in which the best solution has been found (the largest the value, the most important is the diversification phase). For every n, we summarize the percentage gap to the optimal/best value and the average number of iterations.

Table 4 considers medium-sized problems for which we know the optimal objective function. Here we compare the new results of RS with the old ones obtained by RR, of VNS, and the optimal values, Benati et al. (2017). Note that some of those optimal solutions could not be found in Benati et al. (2017) but are certified with the results of our B&P. It can be seen that average objective values are in favor of the new heuristic RS, as the average results of all three implementations are always better than the corresponding ones of both RR and VNS. If instead it is compared how many times the best solution is found, then it happened 26, 23, 22 for the three version of RS and only 20 times for RR and 9 times for VNS, so the same conclusion holds.

The results on new instances, for which in most cases the optimal solutions have been found in this research, are reported in Table 5. It can be seen again that RS with any parameter is on average better than both VNS and RR, even though this time the heuristics have seldom found the optimal solutions. Best solutions have been found 15, 8, 11 times by the three RS's, 6 by RR, and 5 times by VNS. This shows that there is still room for improving the heuristic algorithms (see next section).

Finally, in Table 6, larger instances are considered, and again averages of objective values are in favor of the new heuristic, as the means of all three implementations are better than those from RR and VNS, and counting how many times the best known solution is found, respectively 14, 8, 2, 1, 2, is still in favor of RS. Regarding what parameter choice of RS is best, it can be seen that it does not make a great difference when the instance size is small, but when it gets larger, it seems that $max\_random\_steps = \lceil |V|/3 \rceil$ gives better results. Finally, the iteration in which the best solution is found (columns it_b) exhibits a great variability: This fact suggests that the diversification mechanism devised to explore different solutions has been effective. Heuristics try to improve the same number of initial solutions and therefore they found the same number of local optima. So why RR and VNS, apparently more sophisticated, are left behind by RS? The reason could be the diversification. Both RR and VNS could be too constrained by the initial solution and they stop too early in inferior local optima.

Regarding computational times, finding an improved solution with RS is much faster than with RR or VNS, and the whole computational times are reported in Table 7. As expected given the simplicity of the algorithm, the RS times are much less than VNS and RR. The reason is that the loop of Steps 9–17 of Algorithm 4 is operated in $O(n^2)$, and it is repeated at most $O(n)$ times. There-

Table 6
Results on large-sized problems for different heuristics.

Problem	fo[|V|/3]	it_b	fo[|V|/2]	it_b	fo[2|V|/3]	it_b	RR	VNS	Best Solution
G80_1	-1176	780	-1172	171	-1164	671	-1030	-1086	-1180
G80_2	-1120	31	-1096	385	-1060	667	-968	-1042	-1120
G80_3	-1314	220	-1314	361	-1290	769	-1274	-1260	-1314
G80_4	-1066	696	-1036	638	-1048	780	-976	-900	-1078
G80_5	-1346	213	-1316	30	-1346	568	-1234	-1370	-1370
G80_6	-956	46	-956	194	-930	619	-936	-818	-1008
G80_7	-1298	193	-1282	163	-1270	115	-1246	-1286	-1298
G80_8	-1142	636	-1132	15	-1128	33	-998	-904	-1166
G80_9	-1368	207	-1368	490	-1364	8	-1190	-1196	-1368
G80_10	-1504	730	-1472	489	-1472	142	-1416	-1440	-1504
n=80	**1.0 %**	**130.3**	**2.2 %**	**123.9**	**2.9 %**	**137.2**	**9.3 %**	**9.5 %**	
G100_1	-1732	156	-1732	201	-1732	446	-1630	-1482	-1746
G100_2	-2126	684	-2110	891	-2090	868	-1730	-1908	-2126
G100_3	-1544	687	-1544	843	-1492	346	-1216	-1266	-1544
G100_4	-2184	491	-2208	721	-2140	166	-2094	-1966	-2208
G100_5	-1708	19	-1724	198	-1690	503	-1442	-1386	-1724
G100_6	-2160	678	-2160	295	-2160	798	-2176	-2176	-2176
G100_7	-1860	71	-1838	44	-1914	333	-1686	-1756	-1968
G100_8	-1532	14	-1506	365	-1482	937	-1390	-1484	-1532
G100_9	-2090	876	-2084	890	-2068	309	-1934	-1798	-2090
G100_10	-2276	907	-2308	444	-2234	586	-2136	-2194	-2308
n=100	**1.0 %**	**122.6**	**1.1 %**	**125.4**	**2.2 %**	**128.0**	**10.7 %**	**10.7 %**	
mean	-1575.10	416.75	-1567.90	391.40	-1553.70	483.20	-1435.10	-1435.90	-1591.40
# best solution	14		8		2		1	2	

Table 7
Average of computational times (in seconds).

n	RS	RR	VNS
40	0.07	2.48	1.63
60	0.27	9.91	6.71
80	0.75	30.15	19.32
100	1.65	61.36	42.67

fore it takes $O(n^3)$ operations to calculate a local optimum (to be repeated max_start times). Conversely, VNS and RR are based on local interchange, whose complexity is much higher. For example, it implies a subroutine of $O(n^2)$ only to check the connectivity of interchanging two units.

Having found that the greedy descent performs much better than the local interchange, one may wonder to what extent this result may be applied to other constrained clique partitions. The result strongly depends on the computational cost of shrinking a node with respect to the cost of reassigning a unit. If connectivity constraints are replaced by community constraints, such as the one defined by modularity and/or cohesion, Cafieri, Costa, and Hansen (2015), then these are cases in which a reassignment affects the global properties of clusters. Conversely, the operations of shrinking nodes remains faster, therefore it is very likely that the solution space is explored more efficiently.

6.4. MILP-relaxed matheuristic for GCCP: combining randomized shrink heuristic with branch-and-price

The preceding methodologies can be combined for a new matheuristic: MILP-relaxed matheuristic (Truncated Column Generation). As described previously, this matheuristic consists in solving the pricing problem heuristically with RS, while branching is permitted to the master problem. From previous computational tests, the method combines the velocity of RS with the global accuracy of an ILP formulation. The method is especially useful when an instance must be solved with sufficient accuracy.

Table 8
Cpu time and % gap of different heuristics with respect to best known solution.

	RS		Matheuristic		RS + Matheur.	
n	GAP	CPU	GAP	CPU	GAP	CPU
20	0.21	0.00	0.39	0.03	0.00	0.02
30	1.25	0.03	2.05	0.08	0.48	0.10
36	3.28	0.05	2.19	0.23	2.33	0.18
40	8.02	0.07	3.18	0.27	1.68	0.32
50	7.17	0.15	5.50	1.05	3.18	0.83
54	2.98	0.18	5.75	0.82	2.91	0.62
60	4.38	0.27	7.15	1.71	3.85	1.40
80	0.30	0.75	4.80	7.75	0.00	4.81
100	0.16	1.65	8.16	27.66	0.00	11.50
200	0.22	26.80	6.63	353.24	0.02	227.35
500	0.06	1302.10	4.40	3600.00	0.05	4902.10
1000	0.38	3600.00	3.85	3600.00	0.22	7200.00
Total Result	**2.37**	**411.0**	**4.50**	**632.73**	**1.23**	**1029.10**

In the following computational tests, we compare the solution quality of the RS heuristic with the MILP-relaxed matheuristic. Therefore, we report results on three heuristic: the plain RS, the MILP-relaxed matheuristic in which RS is called only to solve the pricing problem (Matheuristic), and the situation in which RS is also called to initialize the master with a feasible solution and to solve the pricing (RS + Matheur.). Table 8 reports the results of these three algorithms on instances of sizes from $n = 20$ until $n = 1000$. This table shows the CPU time in seconds (CPU) and the gap (*GAP=100('this-heuristic-solution' - 'best-solution')/|'best-solution'|*) of these three algorithms with respect to the best known solution for each instance (note that for instances of sizes greater than 60 the comparison is with respect to the best solution found by one of our own heuristics). The reader may observe that we have imposed to each run a maximum execution time of one hour. Times reported for RS + Matheur. are the aggregation of the time running RS plus the time running matheuristic. Our intuition is confirmed by data: On average the best results are

obtained by the combination of RS with MILP-relaxed matheuristic for any instance size.

7. Conclusions and future work

This paper analyzes the Graph-Connected Clique-Partitioning Problem (GCCP) presenting three different new solution approaches: one exact and two heuristics. In Benati et al. (2017) this problem was already introduced but its solution methods could only handle small-sized instances. Our new approaches improve this drawback. We provide a new Integer Linear Programming (ILP) formulation, based on a set partitioning formulation, that approximates very-well the unknown optimal solution. This set partitioning formulation is solved implementing a branch-and-price (B&P) algorithm. The resulting pricing problem is a new combinatorial problem: the Maximum-weighted Graph-Connected Single-Clique (MGCSC), that is analyzed and solved proposing different MILP formulations.

Besides enlarging the sizes of problems that can be solved exactly, to tackle larger size problems we propose two new fast heuristics: the "random shrink" (RS) and a MILP-relaxed matheuristic. These algorithms improve the previous VNS and RR algorithms of Benati et al. (2017) since they are both faster and more accurate. Extensive computational experiments show the usefulness of our new approaches giving rise to new opportunities to apply this classification methodology, that combines individual and relational data to new actual situations.

Acknowledgments

This research has been partially supported by Spanish Ministry of Education and Science/FEDER grant number MTM2016-74983-C02-(01-02), projects FEDER-US-1256951, CEI-3-FQM331, P18-FR-1422, FEDER-UCA18-106895 by Junta Andalucía/FEDER/UCA, and Contratación de Personal Investigador Doctor. (Convocatoria 2019) 43 Contratos Capital Humano Línea 2. Paidi 2020, supported by the European Social Fund and Junta de Andalucía. The authors also acknowledge funding from project NetmeetData: Ayudas Fundación BBVA a equipos de investigación científica 2019.

Appendix A. Alternative formulations for the pricing problem

A1. Flow-based formulation with a auxiliary node

The rationale of this formulation is the same to the one described in Section 3.1, but using an auxiliary node as a source node. Let $G_D = (V \cup \{0\}, A)$ be a digraph, in which there is an auxiliary node $\{0\}$ and a set of arcs, A, so defined: Two arcs (i, j) and (j, i) for every edge $e_{ij} (= e_{ji}) \in E$; and the auxiliary arcs $(0, i)$ for all $i \in V$. Flow variables f_{ij} are defined for all pairs i, j such that $(i, j) \in A$, the node 0 is assumed to be the flow source node, and it is also assumed a demand of one flow unit from all nodes of V. To define this formulation, we use the same set of variables used for $(\mathbf{F_{flow}})$ but taking into account that now the arc set A includes the arcs with origin at 0. This alternative flow-based formulation of MGCSC is:

$$(\mathbf{F^0_{flow}}) \ \min \sum_{i\in V} \sum_{j\in V: j>i}^{n} c_{ij} z_{ij} - \sum_{i\in V} \gamma_i^* x_i \tag{17}$$

$$s.t. \ (2)-(4), (7)-(9),$$
$$f_{ij} + f_{ji} \le (n-1) z_{ij}, \quad \forall (i,j) \in A : i, j \in V, i < j, \tag{18}$$

$$\sum_{i\in V} f_{0i} = \sum_{i\in V} x_i, \tag{19}$$

$$f_{0i} + \sum_{j\in V:(j,i)\in A} f_{ji} - \sum_{j\in V:(i,j)\in A} f_{ij} = x_i, \quad \forall i \in V, \tag{20}$$

$$z_{0i} \le x_i, \quad \forall i \in V, \tag{21}$$

$$f_{0i} \le n z_{0i}, \quad \forall i \in V, \tag{22}$$

$$\sum_{i\in V} z_{0i} \le 1, \tag{23}$$

$$z_{0i} \in \{0, 1\}, \quad \forall i \in V. \tag{24}$$

Constraints (18) avoid the flow between nodes which are not included in the optimal cluster S. Constraints (19) and (20) are the node conservation flow, in which one unit of flow is retained by the crossed node. Constraints (21)–(23) ensure that the outgoing flow from the auxiliary node is sent to at most one node of the cluster (the flow upper bound is n). Lastly, (24) define the domain of the variables.

A2. Arborescence formulation with an auxiliary node

Let $G_D = (V \cup \{0\}, A)$ be a digraph defined as in Section A.1. The rationale behind this formulation is the one followed in Section 3.2, but the MTZ description of the Spanning Tree builds an arborescence rooted at an auxiliary node 0. Binary variables t, x and z are defined as in the formulation $(\mathbf{F_{MTZ}})$ but taking into account that now the arc set A includes the arcs with origin at 0. Hence, this alternative formulation of the minimum MGCSC problem is:

$$(\mathbf{F^0_{MTZ}}) \ \min \sum_{i\in V} \sum_{j\in V: i<j} c_{ij} z_{ij} - \sum_{i\in V} \gamma_i^* x_i$$
$$s.t. \ (2)-(4), (7), (9)-(11), (13), (14),$$
$$t_{0j} + \sum_{i\in V:(i,j)\in A} t_{ij} = x_j, \quad \forall j \in V, \tag{25}$$

$$\sum_{j\in V} t_{0j} = 1. \tag{26}$$

Constraints (25) and (26) ensure there is only one incident arc to every node of the cluster, so variables t_{ij} define a directed subtree.

Formulation $(\mathbf{F^0_{MTZ}})$ can be strengthened with the some families of valid inequalities described in Section B.3 in Appendix.

Appendix B. Valid inequalities for the pricing problem formulations

B1. Valid inequalities for $(\mathbf{F_{flow}})$

Formulation $(\mathbf{F_{flow}})$ can be strengthened with the following family of valid inequalities:

$$\sum_{j\in V:(i,j)\in A} f_{ij} \ge \sum_{k\in V:k\le i} x_k - x_i - n \sum_{k\in V:k>i} z_{ik} - n(1 - x_i), \quad \forall i \in V, \tag{B27}$$

$$f_{ij} + f_{ji} \le (n-1) z_{ij}, \quad \forall (i,j) \in A : i < j, \tag{B28}$$

$$f_{ij} + f_{ji} \le (n-2) z_{ij} + \sum_{k\in V:i<k} z_{ik}, \quad \forall (i,j) \in A : i < j. \tag{B29}$$

Constraints (Appendix B27) guarantee that the outflow from the node with the highest index of the cluster is at least the number of elements of the cluster minus one. Constraints (Appendix B28) and (Appendix B29) provide upper bound of the flow crossing an edge (in both sense) being this $n-1$ for the node with the highest index in the cluster, $n-2$ for the remaining nodes in the cluster and 0 if one of the two end-nodes of the edges is not in the cluster.

B2. Valid inequalities for ($\mathbf{F_{MTZ}}$)

Formulation ($\mathbf{F_{MTZ}}$) can be strengthened with the following set of valid inequalities:

$$\sum_{i\in V:(i,j)\in A} t_{ij} \leq x_j, \quad \forall j \in V, \tag{B30}$$

$$\sum_{i\in V,(i,j)\in A} t_{ij} \leq \sum_{k\in V:k>j}^{n} z_{jk}, \quad \forall j \in V, \tag{B31}$$

$$\ell_j \leq (n-1) \sum_{k\in V:(k,j)\in A} t_{kj}, \quad \forall j \in V, \tag{B32}$$

$$\ell_i \geq \sum_{k\in V:(k,i)\in A} t_{ki}, \quad \forall i \in V. \tag{B33}$$

Constraints (Appendix B30) guarantee that there is at most an incident arc in the nodes of the cluster. Constraints (Appendix B31) ensure that the node with the greatest index (the root of the subtree) does not have incoming arcs. Constraints (Appendix B32)–(Appendix B33) impose bounds on the ℓ-variables.

B3. Valid inequalities for ($\mathbf{F^0_{MTZ}}$)

Formulation ($\mathbf{F^0_{MTZ}}$) can be strengthened with the following family of valid inequalities:

$$t_{0j} + z_{ij} \leq x_j, \quad \forall (i,j) \in A : i < j, \tag{B34}$$

$$\ell_i \geq x_i - t_{0i}, \quad \forall i \in V, \tag{B35}$$

$$\ell_j \leq (1 - t_{0j})(n-1), \quad \forall j \in V. \tag{B36}$$

Constraints (Appendix B34) establish that the fictitious node is connected with the node of the greatest index of S, therefore they break up symmetric optimal solutions. Constraints (Appendix B35)–(Appendix B36) establish valid bounds for the ℓ-variables.

References

Aloise, D., Cafieri, S., Caporossi, G., Hansen, P., Perron, S., & Liberti, L. (2010). Column generation algorithms for exact modularity maximization in networks. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics, 82*(4).

Álvarez-Miranda, E., Ljubić, I., & Mutzel, P. (2013). *The maximum weight connected subgraph problem* (pp. 245–270). Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-38189-8

Balas, E., Chvátal, V., & Nesetril, J. (1987). On the maximum weight clique problem. *Mathematics of Operations Research, 12*(3), 522–535.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., & Vance, P. H. (1996). Branch-and-price: Column generation for solving huge integer programs. *Operations Research, 46*, 316–329.

Bektaş, T., & Gouveia, L. (2014). *Requiem* for the Miller-Tucker-Zemlin subtour elimination constraints? *European Journal of Operational Research, 236*(3), 820–832.

Benati, S., Puerto, J., & Rodríguez-Chía, A. (2017). Clustering data that are graph connected. *European Journal of Operational Research, 261*, 43–53.

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review, 59*(1), 65–98.

Blanco, V., Japón, A., Ponce, D., & Puerto, J. (2021). On the multisource hyperplanes location problem to fitting set of points. *Computers & Operations Research, 128*, 105124.

Bothorel, C., Cruz, J., Magnani, M., & Micenkova, B. (2015). Clustering attributed graphs: Models, measures and methods. *Network Science, 3*, 408–444.

Cafieri, S., Costa, A., & Hansen, P. (2015). Adding cohesion constraints to models for modularity maximization in networks. *Journal of Complex Networks, 3*(3), 388–410.

Charon, I., & Hudry, O. (2006). Noising methods for a clique partitioning problem. *Discrete Applied Mathematics, 154*(5), 754–769.

Cheng, H., Zhou, Y., Huang, X., & Yu, J. X. (2012). Clustering large attributed information networks: An efficient incremental computing approach. *Data Mining and Knowledge Discovery, 25*(3), 450–477.

Combe, D., Largeron, C., Egyed-Zsigmond, E., & Géry, M. (2012). Combining relations and text in scientific network clustering. In *Proceedings of the international conference on advances in social networks analysis and mining, ASONAM 2012, Istanbul, Turkey, 26-29 August 2012, pages 1248–1253*.

Deleplanque, S., Labbé, M., Ponce, D., & Puerto, J. (2020). A branch-price-and-cut procedure for the discrete ordered median problem. *INFORMS Journal on Computing, 32*(3), 582–599.

Desrosiers, J., & Lübbecke, M. E. (2005). *A primer in column generation* (pp. 1–32). Boston, MA: Springer US. ISBN 978-0-387-25486-9

Gambella, C., Ghaddar, B., & Naoum-Sawaya, J. (2021). Optimization problems for machine learning: A survey. *European Journal of Operational Research, 290*(3), 807–828.

Gleixner, A., Bastubbe, M., Eifler, L., Gally, T., Gamrath, G., Gottwald, R. L., … Witzig, J. (2018). The SCIP optimization suite 6.0. In *Technical Report, Optimization online, july*.

Gouveia, L. (1996). Using the Miller-Tucker-Zemlin constraints to formulate a minimal spanning tree problem with hop constraints. *Computers & Operations Research, 2*(3), 959–970.

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

Gualandi, S., & Malucelli, F. (2013). Constraint programming-based column generation. *Annals of Operations Research, 204*(1), 11–32.

Landete, M., & Marín, A. (2014). Looking for edge-equitable spanning trees. *Computers & Operations Research., 41*, 44–52.

Laporte, G. (1992). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research, 59*(2), 231–247.

Ljubić, I., Weiskircher, R., Pferschy, U., Klau, G. W., Mutzel, P., & Fischetti, M. (2006). An algorithmic framework for the exact solution of the prize-collecting steiner tree problem. *Mathematical Programming, 105*, 427–449.

Lübbecke, M. E., & Desrosiers, J. (2005). Selected topics in column generation. *Operations Research, 53*(6), 1007–1023.

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

du Merle, O., Villeneuve, D., Desrosiers, J., & Hansen, P. (1999). Stabilized column generation. *Discrete Mathematics, 194*(1), 229–237.

Miller, C. E., Tucker, A. W., & Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM, 7*, 326–329.

Neville, J., Adler, M., & Jensen, D. D. (2003). Clustering relational data using attribute and link information. In *Proceedings of the workshop on text mining and link analysis, eighteenth international joint conference on artificial intelligence, Acapulco, Mexico*.

Pessoa, A., Uchoa, E., Poggi, M., & Rodrigues, R. (2010). Exact algorithm over an arc–time-indexed formulation for parallel machine scheduling problems. *Mathematical Programming Computation, 2*, 259–290.

Raidl, G. R. (2015). Decomposition based hybrid metaheuristics. *European Journal of Operational Research, 244*(1), 66–76.

Ryan, D. M., & Foster, A. (1981). An integer programming approach to scheduling. In A. Wren (Ed.), *Computer scheduling of public transport: Urban passenger vehicle and crew scheduling* (pp. 269–280). North-holland, Amsterdan.

Sato, K., & Fukumura, N. (2012). Real-time freight locomotive rescheduling and uncovered train detection during disruption. *European Journal of Operational Research, 221*(3), 636–648.

Sato, K., & Izunaga, Y. (2019). An enhanced milp-based branch-and-price approach to modularity density maximization on graphs. *Computers & Operations Research, 106*, 236–245.