European Journal of Operational Research 261 (2017) 43–53

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

Clustering data that are graph connected

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

[a] Dipartimento di Sociologia e Ricerca Sociale, Università di Trento, Via Verdi 26, 38122 Trento. Italy
[b] IMUS. Universidad de Sevilla, Avda. Reina Mercedes, s/n, 41012 Sevilla. Spain
[c] Faculty of Sciences, Universidad de Cádiz, Avda. República Saharaui, 11510 Puerto Real (Cádiz). Spain

ARTICLE INFO

Article history:
Received 18 June 2016
Accepted 7 February 2017
Available online 12 February 2017

Keywords:
Combinatorial optimization
Clustering
Clique partitioning
Integer programming

ABSTRACT

A new combinatorial model for clustering is proposed for all applications in which individual and relational data are available. Individual data refer to the intrinsic features of units, they are stored in a matrix D, and are the typical input of all clustering algorithms proposed so far. Relational data refer to the observed links between units, representing social ties such as friendship, joint participation to social events, and so on. Relational data are stored in the graph $G = (V, E)$, and the data available for clustering are the triplet $G = (V, E, D)$, called attributed graph. Known clustering algorithms can take advantage of the relational structure of G to redefine and refine the units membership. For example, uncertain membership of units to groups can be resolved using the sociological principle that ties are more likely to form between similar units. The model proposed here shows how to take into account the graph information, combining the clique partitioning objective function (a known clustering methodology) with connectivity as the structural constraint of the resulting clusters. The model can be formulated and solved using Integer Linear Programming and a new family of cutting planes. Moderate size problems are solved, and heuristic procedures are developed for instances in which the optimal solution can only be approximated. Finally, tests conducted on simulated data show that the clusters quality is greatly improved through this methodology.

1. Introduction

The problem of clustering consists in discovering or detecting how a population is partitioned into two or more subgroups, each subgroup specified by distinct features. The typical outcome of a clustering algorithm is the assignment of observations to groups and, most of the times, some estimation of the characteristics of every group. There are many techniques proposed for clustering. Some of them are constructive, in the sense that observations are merged together one at a time until the required partition is found, hierarchical trees being an example. Other techniques have combinatorial structure, for example the k-means, in which clusters are the solutions of an optimization problem, and different partitions are evaluated through the objective function. Finally, some techniques assume that data can be described through probability distributions, so that the optimal clustering is calculated maximizing the likelihood function, for example with the EM algorithm. Each technique has its pros and cons, consequently it can be the most appropriate for some application. Clustering is an easy task if groups are well separated. Conversely, if the borders between groups are uncertain and populated by many observations of different groups with similar features, then the precision of the classification decreases to the point that algorithms just output random clusters.

In this contribution, it is shown that the capability of detecting the true clusters is enhanced if, together with the standard individual data (that is, data that are pertinent to the single population unit), the researcher can observe relational data too, that is, data describing the connections among units. To make an example in social network analysis, suppose that a researcher wants to determine groups with homogeneous cultural orientations within a population of individuals. Data come from a survey in which people answered to questions about their attitudes to religion, politics, family and so on. It is often the case that individuals classified as *"secular"* are not neatly separated from individuals classified as *"religious"*, as they can share the same views on social matters different from religion, see (Inglehart & Baker, 2000). Therefore the clustering algorithm does not correctly classify individuals and the subsequent analysis is flawed, for example the estimation of the groups parameters could result wrong. We propose that a possible remedy to the border uncertainty is to consider an additional data: the relationships, such as friendship, kinship, sympathy, and so on,

* Corresponding author.
E-mail address: antonio.rodriguezchia@uca.es (A.M. Rodríguez-Chía).

http://dx.doi.org/10.1016/j.ejor.2017.02.009
0377-2217/

that exist between individuals. This datum presents an important additional information, expressed by social scientists as the principle of *homophylia*, see (McPherson, Smith-Lovin, & Cook, 2001). According to this principle, people tend to have like-minded friends. With this datum at hand, the uncertain group attributions of a clustering algorithm can be resolved.

Relational data are widely used in social network analysis. In this sense, we mention correlation clustering that operates in a scenario where the relationships between the objects are known instead of the actual representations of the objects, see (Bansal, Blum, & Chawla, 2004; Charikar, Guruswami, & Wirth, 2005; Swamy, 2004). It is discussed in Wasserman and Faust (1994) that there is an intrinsic difference between individual and relational data. Individual data are specific attributes that form the so-called compositional dimension of the social actor. This dimension is to be distinguished from the so-called structural dimension of the social actor, represented by the relations between individuals. Following Wasserman and Faust (1994), there is also the third dimension of an actor, that is called affiliation: It is the individual membership to specific groups, as clubs, companies or social and cultural classes. Often this variable is unknown and must be determined through the classification process. The methodology developed here is aimed at this purpose.

Individual data are given as a matrix D of n rows and m columns, in which n is the number of individuals and m is the number of features. Relational data are given as an undirected graph $G = (V, E)$, in which V are the individuals, $|V| = n$, and there is an edge $e_{ij} \in E$ iff there is a relation between $i, j \in V$. The data structure that combines the graph $G = (V, E)$ with the data matrix D forms the triplet $G = (V, E, D)$ and is called attributed graph, as the matrix D can be interpreted as node data, see (Bothorel, Cruz, Magnani, & Micenkova, 2015).

Early research on attributed graph simplified the problem to apply standard clustering algorithms to the reduction. One form of simplification is obtained projecting the matrix D into the graph G, defining costs or distances c_{ij} on arcs (i, j), depending on the dissimilarity between individual i, j. The weighted graph G' is so obtained and then analyzed using known graph partitioning techniques, see (Neville, Adler, & Jensen, 2003). Another form of simplification proceeds the other way round, that is, projecting the connections of E into the matrix D, and then applying a clustering algorithm, (Combe, Largeron, Egyed-Zsigmond, & Géry, 2012). A more ingenious way of combining G with D is suggested in Cheng, Zhou, Huang, and Yu (2012), in which the two data structures are recoded to form an augmented graph with two classes of nodes. One class of nodes represents individuals and the other class represents features, with arcs connecting both kinds of nodes. Node distances are calculated through random walks, and then the k-medoids (or p-median) clustering is applied. To summarize these previous results: Whether G is projected to D or D to G, the researcher has the advantage that no new algorithm is to be developed for clustering. But the drawback is that compositional and relational data are treated as homogeneous information, even though their nature is intrinsically different.

An approach that keeps separated the two data types is presented in Xu, Ke, Wang, Cheng, and Cheng (2014). Here, it is assumed that there is a probabilistic model underlying relational and compositional data. As commonly assumed in model-based clustering, each unit belongs to a latent class, that determines the probabilities of peculiar features and links. Each class is characterized by one distribution function with its parameters, all data being generated by the distributions mixture. Parameters, including membership, are estimated through the maximization of the likelihood function and using the EM algorithm. The approach is elegant, but requires high computational times and the explicit assumption of what multivariate distribution generated the data.

In this paper, we propose a combinatorial optimization problem to cluster attributed graph. The combinatorial structure of the objective function and constraints is borrowed from a previous clustering model, namely the clique partitioning (at its origin, the clique partitioning problem has been formulated for clustering binary (or qualitative) data, see (Bertsimas & King, 2016; Bertsimas & Shioda, 2007; Grötschel & Wakabayashi, 1989; Jaehn & Pesch, 2013; Johnson, Mehrotra, & Nemhauser, 1993; Marcotorchino & Michaud, 1982; Zhou, Hao, & Goëffon, 2016). However, in our model it is imposed that the eligible groups must be connected through the arcs of G, in the sense that there must be a path from each pair i, j belonging to the same group. We use clique partitioning as the clustering model because, unlike the k-means, the number of groups k is not fixed in advance, but it is the outcome of the algorithm, gaining flexibility in its applications[1]. Our new approach differs from the previous ones in two main aspects. First, a combinatorial objective function is faster to optimize than maximum likelihood, just as the k-means algorithm is faster than EM optimization. Second, compositional and relational data are not merged, but kept separate to play different roles in the problem formulation: objective function and constraints.

From a pure optimization point of view, the model developed here combines two combinatorial problems, namely clique partition, coming from clustering individual data, and the spanning tree detection, coming from imposing connections on the relational data. The first one is a well-known NP-hard problem. Moreover, although the spanning tree design can be modeled as a continuous linear program, when superimposed on other combinatorial structure (clique partition in this case) becomes also extremely difficult. Thus, it is not surprising that our model is NP-hard and also extremely difficult, as it will be clear in the following sections. In spite of that, we have found several valid integer linear programming formulations, based on different rationale. These formulations provide exact solutions for the considered problems for medium size instances (up to 40 units). Beyond this limit, we propose to use heuristic approaches whose performances are compared with exact solutions up to admissible sizes.

The paper is so structured in 6 sections. The first section is the introduction where the problem is motivated and positioned in the related literature. In the second section, we introduce the formal definition of the problem and present several mathematical programming formulations. We develop two kinds of formulations: compact and extended. The former are mixed integer linear programs whose set of constraints are polynomial in the input size whereas the latter are formulations with an exponential number of constraints. Both set of formulations are solved either directly by MIP solvers or with an *ad hoc* Branch-and-Cut scheme in Section 3. Section 4 presents our heuristic algorithms. We propose two of them which are variations of an adapted local search method. The fifth section contains the computational experiments. We compare between them the different exact methods (those based on MIP formulations) on a testbed of random instances and we show that this approach can solve to optimality medium size instances. Next, we apply the two heuristic algorithms to larger sizes reporting their good behavior both on CPU time and quality of the obtained solutions. The paper ends with some remarks on future research directions.

[1] A warning is necessary here: The reader should not be mislead by the term *clique*. The way in which the term is used in cluster analysis depends on the fact that the individual data D are viewed as embedded on a complete graph, so that clusters are viewed as cliques. In this new model and in the following sections, connectivity is a property established with reference to the underlying graph G.

2. Problem formulation

Let $V = \{1, \ldots, n\}$ be the set of units and let $F_k, k = 1, \ldots m$, be the set of binary features measured on V. Let data be collected in the $n \times m$-matrix $D = [d_{ik}]$. Each binary feature defines an equivalence relation between units, that is, unit i is equivalent to j according to features k iff $d_{ik} = d_{jk}$. The number of equivalence relations between two objects defines a cost function c_{ij} in the following way: Let $m_{ij} = \#$(equivalence relations between i and j), then $c_{ij} = m - 2 * m_{ij}$. The value of c_{ij} ranges from the minimum $-m$, denoting full concordance between units i and j, to the maximum of m, denoting discordance between i and j. The clique partitioning problem is defined as finding the partition $\Pi = \{V_1, \ldots, V_p\}$ such that the following objective function is minimized:

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

As the problem is in minimization form, units for which c_{ij} is negative tend to be in the same group. Conversely, units for which c_{ij} is positive tend to be in different groups. As the number p of groups is not determined in advance, but it is the output of the algorithm, and since weights are both positive and negative, the problem is different from the p-cut problem, the k-means and the p-median, and all other clustering algorithms in which the number of clusters must be defined a-priori.

We consider the case that relational data are available as stored in the graph $G = (V, E)$, in which nodes $i \in V$ are the units, and edges $e_{ij} \in E$ describe the unit links. For $Q \subseteq V$, $G[Q] = (Q, E[Q])$ is the subgraph induced by Q, e.g. the graph with edges $e_{ij} \in E[Q]$ iff i, $j \in Q$. The partition $\Pi = \{V_1, \ldots, V_p\}$ is feasible iff $G[V_i], i = 1, \ldots, p$ are all connected subgraphs. We say that there is an *interlink* between two units i and j if they belong to the same partition element. A subset of nodes is said to be interlinked if it is included in the same partition element. Then the Connected Partitioning Problem is to find the partition $\Pi = \{V_1, \ldots, V_p\}$ that is formed by connected components on G and that minimizes the objective function (1).

2.1. Flow based formulation with two indices variables

Recall that we have stated in the introduction that we consider an undirected network $G = (V, E)$ with node set $V = \{1, ..., n\}$ and edge set E, where $e_{ij} (= e_{ji}) \in E$ represents the edge joining the units/nodes i and j; and for any $i, j \in V$ with $i < j$, c_{ij} is the interlink cost (benefit) between i and j, i.e., the cost (benefit) generated by including in the same block these two nodes.

In the following, we present a valid formulation for the problem of finding the cheapest partition of the nodes of a graph with respect to the full pairwise interlink cost with the requirement that all nodes within a block must be connected in the subgraph induced by the block. In order to guarantee the connection of the nodes in the same block, we use the flow based formulation of the Spanning Tree given by Gavish (1983) for the capacitated minimal directed tree problem. However, since in our case we only need the connection among all the nodes in the same block, it is not mandatory to avoid cycles in the resulting graph connecting these nodes, i.e., it is not necessary to have a tree, so we can use a relaxed version of that formulation.

In the formulation of our problem we will use continuous flow variables to guarantee the connection among the nodes of the same block, defined on the arcs of the directed network $G_D = (V, A)$, where A consists of the set of arcs (i, j) and (j, i) such that the edge $e_{ij} (= e_{ji}) \in E$. We will assume that we have the same number of single source nodes as the number of blocks, which will be the nodes with the highest index in each block, with outflow the cardinality of the corresponding block minus one and zero inflow. All other nodes have a demand of one unit.

Therefore, in order to give a formulation of this problem based on the above ideas, we define the following families of variables. For any $i, k = 1, \ldots, n$ such that $i \le k$, the variable z_{ik} is defined as:

$$z_{ik} = \begin{cases} 1, & \text{if node } i \text{ is assigned to block } k, \\ 0, & \text{otherwise.} \end{cases}$$

Observe that, without loss of generality, we can assume that to represent a block we use the node with the highest index within the block. Therefore, z_{ik} variables have to be defined only for $i \le k$.

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

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

And, 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.

Therefore the flow-based formulation is as follows:

$$(F_{flow}) \quad \min \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{ij} x_{ij} \tag{2}$$

$$s.t. \quad x_{ij} \le z_{ik} + \sum_{\ell=j, \ell \ne k}^{n} z_{j\ell}, \quad \forall i, j, k = 1, \ldots, n, \ i < j, i \le k. \tag{3}$$

$$x_{ij} \ge z_{ik} + z_{jk} - 1, \quad \forall i, j, k = 1, \ldots, n, \ i < j \le k, \tag{4}$$

$$z_{ik} \le z_{kk}, \quad \forall i, k = 1, \ldots, n, \ i \le k, \tag{5}$$

$$\sum_{k=i}^{n} z_{ik} = 1, \quad \forall i = 1, \ldots, n, \tag{6}$$

$$\sum_{i=1:\,(k,i) \in A}^{n} f_{ki} - \sum_{i=1:\,(i,k) \in A}^{n} f_{ik} = \sum_{i=1}^{k} z_{ik} - 1, \quad \forall k = 1, \ldots, n, \tag{7}$$

$$f_{ij} \le (n-1) x_{ij}, \quad \forall (i, j) \in A, \ i < j, \tag{8}$$

$$f_{ji} \le (n-1) x_{ij}, \quad \forall (j, i) \in A, \ i < j, \tag{9}$$

$$f_{ij} \ge 0, \quad \forall i, j = 1, \ldots, n, \tag{10}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i, j = 1, \ldots, n, \ i < j, \tag{11}$$

$$z_{ik} \in \{0, 1\}, \quad \forall i, k = 1, \ldots, n, \ i \le k. \tag{12}$$

The objective function (2) accounts for the total cost of the nodes within the same block. The family of constraints (3) ensure that if i and j go in the same block then there is at least one representative of a block where both nodes can be assigned. With (4) we guarantee that the variable x_{ij} takes the value 1 if and only if nodes i and j are in the same block. Both together ensure the interlink among the nodes of the same block. The family of constraints (5) ensures that each node is assigned to a block represented by a node v_k if v_k is assigned to this block. In addition, the family (6) guarantees that each node belong to just one block and this is represented by the node with the greatest index. Constraints (7) are the equations of balance of flow for the nodes of the graph. In particular, the node representing each block has an outgoing flow equal to the number of nodes in this block minus

one and the remaining nodes of the block has demand 1. The families of constraints (8) and (9) avoid the flow between nodes of different blocks. Finally, (11) and (12) give us the binary condition over the variables x and z, respectively. Observe that using (3) and (4), the integrality condition over the x variables can be relaxed and then, family of constraints (11) can be removed.

2.1.1. Valid inequalities

In the following, we propose some families of valid inequalities for the above formulation:

1. The interlink among the nodes of the same block is reinforced as follows, see (Grötschel & Wakabayashi, 1990; Johnson et al., 1993):

$$x_{ij} + x_{j\ell} - x_{i\ell} \leq 1, \quad \forall i, j, l = 1, \ldots, n,\ i < j < \ell,$$
$$x_{i\ell} + x_{j\ell} - x_{ij} \leq 1, \quad \forall i, j, \ell = 1, \ldots, n,\ i < j < \ell,$$
$$x_{ij} + x_{i\ell} - x_{j\ell} \leq 1, \quad \forall i, j, \ell = 1, \ldots, n,\ i < j < \ell.$$

However, in our model we have applied a reduced number of inequalities where redundant constraints in the above set have been removed using the result in Miyauchi and Sukegawa (2015). These inequalities are:

$$x_{ij} + x_{j\ell} - x_{i\ell} \leq 1, \quad \forall i, j, l = 1, \ldots, n,\ i < j < \ell,$$
$$c_{ij} \leq 0 \text{ or } c_{j\ell} \leq 0 \tag{13}$$

$$x_{i\ell} + x_{j\ell} - x_{ij} \leq 1, \quad \forall i, j, \ell = 1, \ldots, n,\ i < j < \ell,$$
$$c_{i\ell} \leq 0 \text{ or } c_{j\ell} \leq 0 \tag{14}$$

$$x_{ij} + x_{i\ell} - x_{j\ell} \leq 1, \quad \forall i, j, \ell = 1, \ldots, n,\ i < j < \ell,$$
$$c_{ij} \leq 0 \text{ or } c_{i\ell} \leq 0. \tag{15}$$

2. The idea that each block is represented by the node with the largest index is reinforced as follows:

$$z_{kk} + \sum_{j=k+1}^{n} x_{kj} \geq 1, \quad \forall k = 1, \ldots, n, \tag{16}$$

$$z_{kk} + x_{kj} \leq 1, \quad \forall j, k = 1, \ldots, n,\ j > k. \tag{17}$$

3. The relationship between variable x and z is strengthened as follows:

$$z_{ij} \leq x_{ij}, \quad \forall i, j = 1, \ldots, n,\ i < j. \tag{18}$$

Taking into account these valid inequalities, we present the following strengthening of F_{flow},

$$(\overline{F_{flow}}) \ \min \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{ij} x_{ij}$$
$$s.t. \ (3) - (18).$$

2.2. MTZ based formulation with two indices variables

In this section we propose a second formulation for our problem using the Miller–Tucker–Zemlin (MTZ) inequalities. MTZ inequalities guarantee the connectivity of the solutions and prevent cycles, these constraints were initially proposed by (Miller, Tucker, & Zemlin, 1960) in the context of the Traveling Salesman Problem. They have been adapted to other problems and reinforced by different authors, see, e.g. (Bektaş & Gouveia, 2014; Gouveia, 1996; Landete & Marín, 2014; Laporte, 1992). The MTZ formulation for the Spanning Tree problem builds an arborescence rooted at a specified node $r \in V$, in which arcs follow the direction from the root to the leaves. It uses binary variables to represent the arcs of the arborescence. Each edge $e_{ij} \in E$, is associated with a pair of binary variables, f_{ij} and f_{ji}, which take the value 1 if and only if arcs (i, j) and $(j, i) \in A$ belong to the arborescence, respectively. In addition, it uses continuous variables ℓ_i, denoting the position that node v_i occupies in the arborescence with respect to r. In our case, we build as many trees as the number of blocks, and the roots of these trees will be the nodes with the largest index in each block. Since we will use this family of inequalities to ensure the connection among the nodes in the same block (not to build an arborescence), some of the constraints of this family can be removed.

Hence, in order to give a formulation for our problem based on the above ideas, we use the two families of binary variables that we have used in the previous formulation, i.e. x and z, and now, as mentioned, the variables f_{ij} are also binary and defined as

$$f_{ij} = \begin{cases} 1, & \text{if the arc } (i, j) \in A \text{ is chosen,} \\ 0, & \text{otherwise.} \end{cases}$$

Using these variables, the formulation of our problem is:

$$(F_{MTZ}) \ \min \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{ij} x_{ij}$$
$$s.t. \ (3) - (6), \ (11), (12),$$
$$\ell_i + 1 \leq \ell_j + n(1 - f_{ij}), \quad \forall i, j = 1, \ldots, n,\ (i, j) \in A, \tag{19}$$

$$\sum_{i=1,\,(i,j)\in E}^{n} f_{ij} = 1 - z_{jj}, \quad \forall j = 1, \ldots, n, \tag{20}$$

$$f_{ij} + f_{ji} \leq x_{ij}, \quad \forall (i, j) \in A,\ i < j, \tag{21}$$

$$f_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E. \tag{22}$$

The family of constraints (19) guarantee the label assigned to node j is at least the label assigned to node i if the arc $(i, j) \in A$ is chosen. Actually, this family avoids the tours. In addition, (20) ensure there is, at least, an incident arc at each node different from the one representing its block. The family (21) guarantees that arcs joining nodes of different blocks cannot be chosen. Finally, constraints (22) give the binary condition of the f variables. As in F_{flow} formulation, we can relax the integrality condition over the x variables; and then, family of constraints (11) can be removed.

2.2.1. Valid inequalities

Valid inequalities (13)–(18) derived for the flow formulation are still valid for this formulation. Moreover, we have some additional ones.

Through the following two families of inequalities, we guarantee that the label assigned to the nodes representing the blocks is 1;

$$\ell_i \geq z_{ii}, \quad \forall i = 1, \ldots, n, \tag{23}$$

$$\ell_i \leq 1 + n(1 - z_{ii}), \quad \forall i = 1, \ldots, n. \tag{24}$$

The following inequalities ensure that the label associated with a node which is not representing its block will be at least 2,

$$2(1 - z_{ii}) \leq \ell_i, \quad \forall i = 1, \ldots, n. \tag{25}$$

Taking into account these valid inequalities, we present the following strengthening of F_{MTZ},

$$(\overline{F_{MTZ}}) \ \min \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{ij} x_{ij}$$
$$s.t. \ (3) - (6), \ (11) - (25).$$

2.2.2. Alternative formulation

Based on the previous formulation we can provide an alternative formulation where the family of variables z has been substituted by variables y_k $\forall k = 1, \ldots, n$, defined as:

$$y_k = \begin{cases} 1, & \text{if node } k \text{ is the node with the highest index in its block,} \\ 0, & \text{otherwise.} \end{cases}$$

The formulation of the problem using this new family of variables is:

$$(F2_{MTZ}) \ \min \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{ij} x_{ij}$$

$$s.t. \ \ (11),\ (13)-(15),\ (19),\ (21),\ (22),$$

$$y_k + \sum_{j=k+1}^{n} x_{kj} \geq 1, \quad \forall k = 1, \ldots, n, \tag{26}$$

$$y_k + x_{kj} \leq 1, \quad \forall k < j = 1, \ldots, n, \tag{27}$$

$$\sum_{i=1,\,(i,j)\in A}^{n} f_{ij} = 1 - y_j, \quad \forall j = 1, \ldots, n, \tag{28}$$

$$y_k \in \{0, 1\}, \quad \forall k = 1, \ldots, n. \tag{29}$$

The families of constraints (26) and (27) guarantee that each node is assigned to just one block and this is represented by the node with the greatest index. Actually, these two families of constraints are a rewriting of constraints (16) and (17) using variables y. In the same sense, constraints (28) are an adaptation of constraints (20) using variables y. Again, the integrality condition on the y variables can be relaxed by constraints (28).

The families of valid inequalities (23)–(25) are still valid for this formulation (replacing z_{ii} by y_i for all $i \in V$.)

Taking into account these valid inequalities, we present the following strengthening of $F2_{MTZ}$,

$$(\overline{F2_{MTZ}}) \ \min \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{ij} x_{ij}$$

$$s.t. \ \ (11),\ (13)-(15),\ (19),\ (21)-(29).$$

2.3. A formulation with an exponential number of constraints

In this section we propose a third type of formulation for our problem using a family of exponential number of inequalities to guarantee the connectivity of the solutions.

Theorem 2.1. *Let x be a solution of formulation F_{flow} ($\overline{F_{flow}}$) removing constraints (7)–(9) or of formulation F_{MTZ} ($\overline{F_{MTZ}}$) removing constraints (19)–(21), such that x does not satisfy the connectivity of nodes within some block. Then, x does not satisfy the inequality*

$$\sum_{\ell=i+1\,:\,\ell\notin S}^{n} x_{i\ell} + \sum_{\ell=1\,:\,\ell\notin S}^{i-1} x_{\ell i} - x_{ij} \geq 0, \tag{30}$$

for some $S \subseteq V$ and for any $i, j (i < j) \in S$, such that, i and j are not connected in $G[S]$.

Proof. The rationale behind inequality (30) is the following. A set $S \subseteq V$ would become a block if $G[S]$ (the graph induced by its nodes) were connected. On the contrary, if $i, j \in V$ were not connected in $G[S]$ then any block containing i and j should include at least another node, $\ell \notin S$, to ensure the connection of the resulting induced graph. This condition is encoded in the inequality (30) assuming that either the inequality $x_{i\ell} \geq x_{ij}$ or $x_{\ell i} \geq x_{ij}$ is fulfilled for some $\ell \notin S$, i.e., if i and j belong to the same block (which forces $x_{ij} = 1$) then $\ell \notin S$ should exist, belonging to the same block, either $x_{i\ell} = 1$ or $x_{\ell i} = 1$. □

The reader may note that, from its own construction, if a solution x satisfies the inequalities (30) for all $S \subseteq V$ and for any $i, j (i < j) \in S$, such that, i and j are not connected in $G[S]$, then x induces a partition such that its nodes are connected within its blocks.

Observe that formulations in Sections 2.1 and 2.2 have two main groups of constraints; the first one to guarantee the interlink among all the nodes of the same block and the second one, to ensure the connection of the subgraph induced by each block. Hence, we can obtain a third family of valid formulations just combining the different ways to model the interlink among the nodes of the same block with the family of constraints (30) for all $S \subseteq V$ and for all $i, j (i < j) \in S$, such that, i and j are not connected in $G[S]$; that ensures connectivity within blocks, as stated in the proof of Theorem 2.1. Among all the possible combinations, in the following, we show the two valid formulations of this kind that have provided the best computational results out of a number of tests among different alternatives. These are based on formulations $\overline{F_{MTZ}}$ and $\overline{F2_{MTZ}}$ after replacing the families of constraints (19), (23)–(25) by (30). The first one uses variables x and z;

$$(FE_1) \ \min \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{ij} x_{ij}$$

$$s.t. \ \ (3)-(6),\ (11)-(18),\ (20)-(22),\ (30).$$

The second one uses only variables x and results in:

$$(FE_2) \ \min \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{ij} x_{ij}$$

$$s.t. \ \ (11),\ (13)-(15),\ (21),\ (22),\ (26)-(29),\ (30).$$

As mentioned above, the family of constraints (30) contains a exponential number of inequalities. Therefore, in order to give a solution approach for formulations FE_1 and FE_2, we will need to develop a procedure that allows us to separate unfeasible solutions in an efficient way. In order to describe this procedure, assume we are given a solution $(\bar{x}, \bar{z}, \bar{f})$ of Formulation FE_1 removing the family of constraints (30) (or including, at most, a polynomial subset of them). The following algorithm separates those solutions that do not satisfy the connection condition within each block. In this algorithm we will refer to $LSP(i, j)$ as the length of the shortest path from node i to node j in the graph $G = (V, E)$; this can be computed, for instance, by the algorithm of Floyd–Warshall.

Observe that $(\bar{x}, \bar{z}, \bar{f})$ is cut by (31) because due to (3) we have that:

$$\sum_{\ell=i+1\,:\,\ell\notin S_k}^{k-1} x_{i\ell} + \sum_{\ell=k+1}^{n} x_{i\ell} + \sum_{\ell=1\,:\,\ell\notin S_k}^{i-1} x_{\ell i} = 0.$$

Alternatively, for formulation FE_2 where the z variables are not used anymore, we consider the same type of cuts but redefining $S_k := \{\ell \,:\, \bar{x}_{\ell k} = 1 \text{ and } \bar{y}_k = 1\}$. Observe that the same inequality (31) with this S_k also cuts away the solution $(\bar{x}, \bar{y}, \bar{f})$ in FE_2.

3. Exact solution approaches

In addition to solving the previous formulations using a commercial solver, we propose two ad-hoc solution approaches for our problem. The first one consists on a Branch & Cut procedure and the second one is based on the use of incomplete formulations and sequentially introducing cuts in the nodes of the branch and bound tree to obtain feasible solutions. In what follows we describe both methodologies.

3.1. Branch & cut procedure

For the formulations $\overline{F_{flow}}$, $\overline{F_{MTZ}}$ and $\overline{F2_{MTZ}}$, we have developed a B&C procedure, where we have introduced a family of cuts in each node of the branch and bound tree. Indeed, in each node of the tree, we iteratively solve the resulting problem after including cuts of the type (31) for formulations $\overline{F_{flow}}$, $\overline{F_{MTZ}}$ and $\overline{F2_{MTZ}}$ by applying Algorithm 1. In this case, since we cannot guarantee the integrality of the optimal solution in each node of the B&B tree, we redefine S_k imposing that $\bar{z}_{\ell k} > \varepsilon$ for the first two formulations and $\bar{x}_{\ell k} > \varepsilon$ for the third one, instead of $\bar{z}_{\ell k} = 1$ and $\bar{x}_{\ell k} = 1$ for any $\varepsilon > 0$, respectively, in Table 1.

Algorithm 1 Separation algorithm.

procedure ADDING CUTS
  for $k \in V$ with $\bar{z}_{kk} = 1$ **do**
    $S_k := \{\ell : \bar{z}_{\ell k} = 1\}$ ▷ The kth block
    for $i, j (i < j) \in S_k$ **do**
      Compute $LSP(i, j)$ in $G = (V, E)$ with length of edges defined by:

$$length(e_{ij}) := \begin{cases} 0, & \text{if } e_{ij} \in E[S_k] \\ 1, & \text{otherwise.} \end{cases}$$

      if $LSP(i, j) > 0$ **then** ▷ i and j are not connected in $G[S_k]$
        Add the following inequality of family (30):

$$\sum_{\ell=i+1\,:\,\ell\notin S_k}^{k-1} x_{i\ell} + \sum_{\ell=k+1}^{n} x_{i\ell} + \sum_{\ell=1\,:\,\ell\notin S_k}^{i-1} x_{\ell i} - x_{ij} \geq 0. \qquad (31)$$

      end if
    end for
  end for
end procedure

The results obtained applying this Branch & Cut procedure to the formulations $\overline{F_{flow}}$, $\overline{F_{MTZ}}$ and $\overline{F2_{MTZ}}$ will be referred to as B&C_$\overline{F_{flow}}$, B&C_$\overline{F_{MTZ}}$, and B&C_$\overline{F2_{MTZ}}$, respectively.

3.2. Incomplete formulations

In order to obtain better computational results in the solution times of the previous formulations and to increase the size of the instances that we are able to solve, we propose to use incomplete formulations. That is, we propose to use formulations which are relaxations for our problem and introduce cuts progressively in the nodes of the branch and bound tree to obtain feasible solutions. Based on this idea, we follow three strategies to generate incomplete formulations.

The first two strategies consist of removing the group of constraints that ensures the connection, (30), from FE_1 and FE_2, respectively. And, the third strategy i) removes (30) and ii) partially removes the group of constraints that ensure the interlink among the nodes of the same block.

3.2.1. First incomplete formulation

In order to generate the first incomplete formulation, we consider formulation FE_1 without constraints (30). Therefore, the resulting formulation ensures the blocks are interlinked, i.e., all the x variables between pair of nodes in the same block take the value 1. However, the connection between each pair of nodes of the same block is not guaranteed. Hence, in the solution procedure we will proceed as in the B&C scheme previously described.

Table 1
Computational results of formulations in Section 2.

n	Formulation	#	Time	GAP	GAP_R	Nodes
20	$\overline{F_{flow}}$	0	7.2	0.0	25.6	1441.5
	$\overline{F_{MTZ}}$	0	55.1	0.0	21.2	29616.1
	$\overline{F2_{MTZ}}$	0	9.9	0.0	18.6	13698.4
	B&C_$\overline{F_{flow}}$	0	12.3	0.0	25.6	2819.3
	B&C_$\overline{F_{MTZ}}$	0	18.0	0.0	21.6	11176.8
	B&C_$\overline{F2_{MTZ}}$	0	5.7	0.0	18.6	4861.5
	FI_1	0	2.3	0.0	16.0	257.2
	FI_2	0	1.7	0.0	17.0	59.2
	FI_3	0	0.8	0.0	27.9	105.1
	FI_3+sol_heu	0	0.8	0.0	27.9	31.4
30	$\overline{F_{flow}}$	4	4190.1	12.1	33.0	74002.0
	$\overline{F_{MTZ}}$	5	4878.8	6.6	29.4	307347.0
	$\overline{F2_{MTZ}}$	0	1747.4	0.0	25.9	435875.2
	B&C_$\overline{F_{flow}}$	4	4289.4	9.7	32.4	77710.2
	B&C_$\overline{F_{MTZ}}$	4	4566.7	6.5	29.7	262300.1
	B&C_$\overline{F2_{MTZ}}$	0	2235.4	0.0	25.9	454493.0
	FI_1	1	953.5	1.3	23.5	5301.4
	FI_2	0	121.4	0.0	24.2	1158.4
	FI_3	0	36.6	0.0	34.9	2383.5
	FI_3+sol_heu	0	25.0	0.0	34.9	1129.4
36	$\overline{F_{flow}}$	10	7200.1	30.6	43.4	28365.3
	$\overline{F_{MTZ}}$	10	7199.8	39.6	52.3	97495.2
	$\overline{F2_{MTZ}}$	6	6249.5	15.1	34.6	362969.4
	B&C_$\overline{F_{flow}}$	10	7200.0	26.3	41.3	33382.1
	B&C_$\overline{F_{MTZ}}$	10	7199.8	39.6	51.5	92430.5
	B&C_$\overline{F2_{MTZ}}$	5	5568.8	9.0	30.3	347770.6
	FI_1	3	3339.7	2.5	23.7	8817.0
	FI_2	0	1240.1	0.0	24.2	4608.8
	FI_3	0	1016.1	0.0	34.8	25810.4
	FI_3+sol_heu	0	487.2	0.0	34.8	8875.8
40	$\overline{F_{flow}}$	10.0	7200.2	54.2	59.2	12746.0
	$\overline{F_{MTZ}}$	10.0	7200.0	22.5	34.1	38550.8
	$\overline{F2_{MTZ}}$	7.0	5790.6	34.7	46.8	116288.9
	B&C_$\overline{F_{flow}}$	10.0	7200.0	55.6	60.1	14611.8
	B&C_$\overline{F_{MTZ}}$	10.0	7199.9	38.1	47.1	37762.5
	B&C_$\overline{F2_{MTZ}}$	8.0	6556.0	31.5	43.8	120047.3
	FI_1	8.0	6480.2	11.4	26.5	9047.5
	FI_2	4.0	3993.7	8.0	26.7	6239.9
	FI_3	4	3400.7	7.1	37.7	34733.6
	FI_3+sol_heu	2	2656.3	2.2	35.3	27093.8

This incomplete formulation has been reinforced with the following valid inequality:

$$\sum_{\substack{i=1 \\ e_{ij}\in E}}^{j-1} x_{ij} + \sum_{\substack{i=j+1 \\ e_{ji}\in E}}^{n} x_{ji} \geq 1 - z_{jj}, \quad \forall j = 1, \ldots, n. \qquad (32)$$

The family of inequalities (32) gives a necessary condition for the connection of the nodes in the same group but it is not sufficient. Actually, it implies that every node of a block except the representative of the block must be connected with another node of the block via an edge of E. Therefore, the first incomplete formulation is:

$$(FI_1) \ \min \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{ij} x_{ij}$$
$$s.t. \ \ (3) - (6), (11) - (18), (20) - (22), (32).$$

3.2.2. Second incomplete formulation

In the second incomplete formulation, we consider formulation FE_2 where the family of constraints (30) has been removed and (32) is included instead. Therefore, the second incomplete formulation is:

$$(FI_2) \ \min \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{ij} x_{ij}$$
$$s.t. \ \ (11), (13) - (15), (21), (22), (26) - (29), (32).$$

As before, these constraints ensure that all the x variables between pair of nodes in the same block take the value 1, but the connection between each pair of nodes of the same block is not guaranteed. The difference with respect to the previous one is that this formulation does not use z variables. Therefore, the non-connected solutions obtained in each node of the branch-and-bound tree are cut following a similar procedure to the B&C previously defined.

3.2.3. Third incomplete formulation

In the third incomplete formulation, we consider formulation FI_2 where the family of constraints (13) and (14) have been partially removed. Therefore, the third incomplete formulation is:

$$(FI_3)\ \min \sum_{i=1}^{n}\sum_{j=i+1}^{n} c_{ij}x_{ij}$$
$$s.t.\ \ (11), (15), (21), (22), (26)-(29), (32),$$
$$x_{1j} + x_{j\ell} - x_{1\ell} \le 1, \quad \forall j, l = 1,\dots,n,\ i < j < \ell, \tag{33}$$

$$x_{in} + x_{jn} - x_{ij} \le 1, \quad \forall i, j = 1,\dots,n,\ i < j < \ell. \tag{34}$$

In this case, besides to not guarantee the connection of the nodes within the same block, this formulation does not even ensure the interlink among all the nodes of the same block. Therefore, in addition to adding the violated cuts of the family (31), we need some additional valid inequalities, namely the violated constraints of the family (13) and (14).

4. Heuristic

The problem complexity requires the development of heuristic methods for two reasons. The first reason is that we want to solve problems of any size, but integer linear programming solves medium-sized problems only. The second reason is that a good feasible solution to the problem may be beneficial for the initialization of an integer linear programming algorithm.

The two methods that are proposed here are plain variations on local search heuristics. It can be readily seen from the combinatorial structure of the problem that the clusters of an incumbent solution can be modified re-assigning objects to different clusters. If the connectivity constraints are still satisfied and the objective function improves, then the incumbent solution is updated. The process is then repeated and halted when clustering cannot be improved. In this case it is said that a local optimum has been reached. More formally, for every node i, let $N[i] = \{j : e_{ij} \in E\} \subseteq V$ be the node set of the neighbors of i. Let $\Pi = \{V_1, \dots, V_p\}$ be a partition of V, the partition is feasible if sets V_j are connected for all $j = 1, \dots, p$. Let $V_{c(i)}$ be that set such that $i \in V_j$, that is, $c(i) = j$.

A move $m = (i, q)$ is a pair $i \in V$, $q = 0, \dots, p$. If $q \ge 1$, then the move represents the possible assignment of unit i to cluster V_q; the move is feasible if both new cluster $V_{c(i)}\backslash\{i\}$ and $V_q \cup \{i\}$ are connected. If $q = 0$, then the move represents the assignment of i to an empty cluster, that is i becomes a singleton and eventually $V_{p+1} = \{i\}$. The evaluation of a feasible move is calculated as the value:

$$\delta_{iq} = \begin{cases} \sum_{j\in V_q} c_{ij} - \sum_{j\in V_{c(i)}} c_{ij} \text{ if } q \ge 1; \\ -\sum_{j\in V_{c(i)}} c_{ij} \text{ if } q = 0. \end{cases} \tag{35}$$

The Local Search Algorithm 2, called g-Clique begins with a feasible partition Π, then tries to improve the objective function moving units to different clusters. If there is no feasible move that decreases the objective function, then Π is a local optimum. When a local optimum is found, then the procedure can be applied again to new initial partitions, for a maximum t^{max} attempted re-starts.

At the beginning, Algorithm 2 calculates the objective function f of a random clustering Π. Partition Π is selected by one of the methods "Random Restart" (RR) or "Variable Neighborhood Search" (VNS), that are described later. The while loop of instructions from 5 to 14 leads Π to a local optimum, applying repeatedly the procedures Moves_Compilation and Choose_Move. The output of Moves_Compilation is a matrix $\Delta = [\delta_{iq}], i = 1 \dots n, q = 0 \dots p$ listing the improvement of the objective function for each feasible move. The output of the procedure Choose_Move is a list of moves $M = \{m_1, \dots, m_w\}$ and a value d. The list of moves M, when applied to Π, leads to a new feasible partition with the value of the objective function decreased by d. The moves of M are chosen in a greedy way from the matrix Δ, considering that there are multiple moves improving the objective function without needing to recalculate Δ. Indeed, if a move $m_t = (i, q)$ is chosen and applied to Π, then values δ_{js} remain the same if $s \ne q$, $c(i)$ and $c(j) \ne q$, $c(i)$. More precisely, procedure Choose_Move is described in Algorithm 3.

Algorithm 2 g-Clique algorithm.

```
1: procedure Local Search for Connected Cluster
2:   for t from 1 to t^max do ▷ Local Search is repeated t^max times
3:     (Π, f) ← Random − g − Clique(version = "RR" or "VNS") ▷ Π is a random feasible partition, f is the objective function
4:     loc_opt = FALSE ▷ Condition for a local optimum
5:     while loc_opt = FALSE do
6:       Δ ← Moves_Compilation(Π, ...) ▷ Generate a list of moves
7:       (M, d) ← Choose_Move(Δ, ...) ▷ M a subset of moves, d variation of the objective function
8:       if d < 0 then
9:         f ← f + d, ▷ Decrease of the Objective Function
10:        Apply M to Π. ▷ Obtain a new partition Π
11:      else
12:        loc_opt = TRUE. ▷ Condition for a local optimum
13:      end if
14:    end while
15:  end for
16: end procedure
```

Algorithm 3 Choose_Move procedure.

```
1: procedure Choose_Move
2:   Input: Δ = [δ_iq], i = 1, ..., n, q = 0, ..., p ▷ q = 0 is a fictitious empty cluster.
3:   Output: M= list of moves; d = variation of the objective function
4:   V ← {1, ..., n}, K ← {1, ..., p}, M ← ∅
5:   D ← 0, f_improve ← TRUE ▷ Initialization
6:   while V ≠ ∅, K ≠ ∅, f_improve = TRUE do
7:     Let (i, q) ∈ arg min{δ_jw | j ∈ V, w ∈ K} ▷ Move Choice
8:     if δ_iq < 0 then
9:       d = d + δ_iq ▷ update d
10:      M ← M ∪ {(i, q)}
11:      V ← V \ {i}
12:      K ← K \ {c(i)}
13:      if q ≥ 1 then
14:        K ← K \ q
15:      end if
16:    else
17:      f_improve = FALSE ▷ no improvement moves are left
18:    end if
19:  end while
20: end procedure
```

It remains to describe how partitions Π are generated in Step 3 of Algorithm 2. Two alternatives are tested. The first procedure works out Π from scratch and randomly and is called Random Restart (RR). In our code, it is implemented labeling objects with random integer numbers between 1 and c^{max}, then the clusters of Π are the connected objects having the same label, with the final number of clusters possibly being larger than c^{max}. RR is chosen for its simplicity among the family of the so-called Multi-Start methods, see (Martí, Resende, & Ribeiro, 2013), and because it is often used as the benchmark to which to compare more elaborate method, for example see (Benati, 2008). In this contribution, RR is compared to a second procedure called Variable Neighborhood Search (VNS). The starting solutions of VNS blend randomness with pieces of information of the best found solution. I, as not all partitions are broken, but only part of them. In our code, VNS is implemented as follows: at the beginning Π is a random partition. Then, after that the first best solution Π^{best} is calculated, a new Π is worked out from Π^{best} relocating t objects into different clusters, with t a varying parameter. At the earliest stages, t is small, so that new partitions are in the immediate proximity of Π^{best}. If no improvement of Π^{best} is found, then t is increased to begin the search and to explore solutions that are further away from Π^{best}. In our code, when t reassignments are to be done, both objects and clusters are chosen randomly, but an object is relocated only when the new partitions are feasible, e.g., connected. The qualitative difference between RR and VNS is that the search of the latter method is more constrained than the one of the former, as it is designed to take advantage of the (potential) good clusters contained in Π^{best}. The principle of VNS are explained in Hansen, Mladenovic, and Moreno Perez (2010), recent applications of VNS to clustering problems are available Hansen, Ruiz, and Aloise (2012) and Cafieri, Hansen, and Mladenovic (2014).

5. Computational results

Algorithms are tested on simulated instances, in which clusters features and network connections are controlled by some parameters. The experiment layout is the one 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 all units of Group 1 tend to be a vector of ones and all units of Group 2 is a vector of zeros. As p_c gets closer to 0.5, the distinction between the two groups is less and less precise. Then units are connected through arcs: If two units (or nodes) belong to the same group, then the probability that there is an arc between the two units is p_{in} (the probability of an internal arc). If the two nodes belong to two different groups, then the probability that there is an arc between the two units is p_{out} (the probability of an external arc). Vertex i is connected to X_{in} vertices of the same group of i and the X_{out} vertices of the other group, where X_{in} and X_{out} are random variables. With the parameters above, $E[X_{in}] \approx np_{in}/2$ 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.

At first, parameters have been fixed to $m = 10$, $p_c = 0.60$, $E[X_{in}] \approx 3$ and $E[X_{out}] \approx 1$, and the algorithms have been run. In the first set of experiments we have computationally tested the formulations presented in Section 2 and their corresponding strengthening. Thus, we have implemented all of them in the commercial solver XPRESS-IVE 1.24.04 running on an Intel(R) Core(TM) i7-4790 CPU @400GHz 32GB RAM. The cut generation option of XPRESS was disabled in order to compare the relative performance of the formulations cleanly. The source code used for this computational analysis is available in https://github.com/antoniochia/code_clustering_data-0.

The results are reported in Table 1 and they correspond to the average of those obtained after solving ten instances for each size, the time was limited to 2 h of CPU. In a preliminar computational study we have tested that formulations $\overline{F_{flow}}$, $\overline{F_{MTZ}}$, $\overline{F2_{MTZ}}$ report better performance that F_{flow}, F_{MTZ} and $F2_{MTZ}$, respectively, in running times and gap at termination. For this reason, in the following we will only report results of $\overline{F_{flow}}$, $\overline{F_{MTZ}}$, $\overline{F2_{MTZ}}$. Hence, the first two columns of Table 1 correspond to the size and the formulation used to solve these instances, respectively. The third and fourth columns give the number of instances that were not optimally solved within the time limit and the CPU time in seconds, respectively. The three last columns, GAP, GAP_R and Nodes, stand for the averages of: gap between the best bound and the best solution after 7200 s (for the instances whose CPU time exceeded the time limit), the gap in the root node and number of nodes in the B&B tree, respectively. To obtain a general idea of the comparisons among these averaged values, for the results in the column *Time* and for different formulations, we have accounted the value 7200 s for those instances that exceed the time limit. In the same way, the values used to compute the average of the column *Nodes* have been the number of nodes of the B&B tree when the CPU time limit was reached.

The formulations described in the second column correspond to the ones described in Section 2. The last one, "$FI_3 + sol\_heu$" corresponds to the third incomplete formulation where the solver has been fed with the solution provided by the heuristic approach. As we can see, among the three first formulations, $\overline{F_{flow}}$, $\overline{F_{MTZ}}$ and $\overline{F2_{MTZ}}$, the third one provides much better results than the other ones. Regarding the branch and cut approach, although it does not provide a uniform improvement with respect to the running times, it slightly improves the gap in the instances where the optimal solution is not achieved within the limit time. With respect to the incomplete formulations, the third-incomplete formulation reports much better computational results than the remaining ones. Observe that the best computational results have been obtained when the solution obtained by the heuristic approach is fed to the solver using the third incomplete formulation. As a conclusion, we can see that the successive improvements from the first to the last formulation have provided a remarkable reduction of CPU time (around two orders of magnitude).

The heuristic algorithm g-Clique, in its version RR and VNS, has been coded in Julia 0.3.7. Julia is a programming language that is trying to blend the flexibility of high level languages like MATLAB and R with the fast execution times of the low-level languages like C and Fortran. At present, C and Fortran remain the fastest compilers, but Julia is close second with computational times that are reportedly only two or three times larger than the best ones.

Both versions RR and VNS start from the same initial random partition Π, the value t^{max}, that is the maximum number of starting solutions, has been fixed to $10n$. The computational results are reported in Table 2 for the medium size instances and Table 3 for the large size instances. Columns report the objective function and the number of iterations needed to reach the best known solution, with iterations calculated as the number of times that the matrix Δ is worked out in line 6 of Algorithm 2. In Table 2, the last column reports the difference between the best found heuristic solution and the optimal one, with empty spaces meaning that they are equal, "unknown" label meaning that the optimal solution is unknown.

Looking at the results of Table 2, there is a clear evidence that in medium size problem RR is better than VNS. In 26 out of 40

Table 2
Heuristic results for small/medium sized problems.

Name	RR	it	VNS	it	Optimal
20-1	−114	11	−114	64	
20-2	−74	28	−34	4	
20-3	−122	34	−122	67	
20-4	−112	14	−110	8	
20-5	−128	233	−102	8	
20-6	−102	112	−96	13	
20-7	−154	100	−102	442	
20-8	−94	895	−92	226	−96
20-9	−116	9	−116	28	
20-10	−140	11	−140	10	
30-1	−254	1210	−248	1661	
30-2	−152	171	−152	109	
30-3	−200	1265	−144	12	−210
30-4	−200	1378	−170	1003	
30-5	−288	1675	−276	296	
30-6	−260	382	−260	101	
30-7	−228	377	−222	280	
30-8	−122	906	−108	95	−126
30-9	−276	973	−136	16	
30-10	−168	2274	−154	21	−176
36-1	−296	3437	−296	1033	−300
36-2	−300	588	−300	33	−304
36-3	−356	1372	−340	15	−390
36-4	−326	1573	−304	1015	−340
36-5	v300	767	−300	420	
36-6	−286	121	−286	1045	
36-7	−310	1723	−324	452	−344
36-8	−230	1988	−204	25	−246
36-9	−260	1643	−242	518	−268
36-10	−290	1385	−290	115	−296
40-1	−294	2128	−284	15	unknown
40-2	−514	3038	−508	30	
40-3	−288	1144	−238	2330	unknown
40-4	−384	1005	−384	252	−412
40-5	−326	1231	−342	2335	
40-6	−292	912	−252	252	unknown
40-7	−272	964	−184	86	−330
40-8	−270	2453	−252	475	−314
40-9	−396	426	−376	417	
40-10	−420	2053	−372	34	−456

problems the RR objective function is better than the VNS one, in only 2 occurrences out of 40 VNS is better than RR. Considering only the 26 instances in which RR is better than VNS, one can see that in 23 out of 26 cases the number of iterations to calculate the optimum is larger for RR than for VNS: the average is 1197 iterations for RR and only 358 for VNS. One explanation of the VNS poor performance can be that the VNS search is too constrained: It takes fewer iterations because the local search starts from solutions that are too close to each other, so that one local optimum is found in the early phase of exploration, but then there is no sufficient diversification to jump to farther away feasible regions, where possibly the optimum is located.

Computational results for the large size graphs, e.g. graphs with $n = 80, 100,$ are reported in Table 3. Here the results of RR and VNS are similar, being the averages of the objective function the same. The best solution is found 8 out of 20 times by RR, 11 by VNS and one time by both. When we consider the instances in which RR is better, the iterations needed to find the best solution are 10276 on averages for RR, 3876 for VNS: Again, VNS takes less iterations. One possible explanation to the fewer iterations, but equal objective function is that, as before, VNS explores solutions that are close to a local optimum, but now the dimension of the problem is so large that it is worth exploring thoroughly local optima. Conversely, RR cannot find the best solution when it is close to the incumbent solution, as RR starts every local search from scratch.

Table 3
Heuristic results for the largest sized problems.

n	label	RR	it	VNS	it
80	1	−1030	2389	−1086	4245
80	2	−968	7601	−1042	2670
80	3	−1274	14858	−1260	141
80	4	−976	8314	−900	1035
80	5	−1234	1487	−1370	415
80	6	−936	10286	−818	4512
80	7	−1246	10875	−1286	3260
80	8	−998	10262	−904	1954
80	9	−1190	5695	−1196	1562
80	10	−1416	2514	−1440	1732
100	1	−1630	5027	−1482	10911
100	2	−1730	2097	−1908	7597
100	3	−1216	3138	−1266	2873
100	4	−2094	4527	−1966	3299
100	5	−1442	13965	−1386	6752
100	6	−2176	2844	−2176	3648
100	7	−1686	2289	−1756	3034
100	8	−1390	15593	−1484	408
100	9	−1934	14973	−1798	2405
100	10	−2136	4052	−2194	15048
mean		−1435.1	7139.3	−1435.9	3875.1

Table 4
Comparison between the exact and heuristic approaches.

n	t_exact	t_RR	t_VNS	#	GAP_exact/heur
20	0.8	1.45	0.59	9	0.2
30	25.0	2.67	2.00	7	1.2
36	487.2	4.70	3.56	2	3.3
40	2656.3	7.36	4.18	3	4.6
60		33.01	21.15		
80		86.99	56.74		
100		191.18	118.32		

Table 4 reports a comparison between the exact and the two heuristic methods. The first three columns report the size of the instances and the average solution times to solve these instances using $FI_3 + sol\_heu$, RR and VNS solution approaches, respectively. Column # gives the number of the ten tested instances in which g-Clique has been able to find the optimal solution. When these do not achieve the optimal solution, the last column reports the gap between the optimal and the heuristic solution. We can see that the heuristic approaches provide good results in those instances where we are solving to optimality. Indeed, the time to solve them is less than 8 s for instances of size 40 and the gap with respect to the optimal solution is less than five percent. In addition for instances of size 100, the solutions are obtained in less than 200 s.

The main motivation of the new clustering model proposed here is that connection data improves the quality of the clustering algorithms. The following experiment proves that it is so. The experiment has been run with fixed values of $n = 50$ and $m = 10$, $p_{out} = 0.04$ and with varying parameters $p_c = 0.65, 0.60, 0.55$, $p_{in} = 0.12, 0.16, 0.20$. Given those probabilities, the mode of Group 1 is a vector of ones and the mode of Group 2 is a vector of zeros. The expected number of arcs that from one node point outside its group is 1 and the expected number of arcs that point within the group are approximately 3, 4, or 5. These data are very difficult to cluster because, even though the groups modes are two vectors of all 0 and 1, those occurrences are seldom observed in practice. Rather, with the simulated values of p_c, there is even a non-negligible probability that units are actually closer to the mode of the other group than to its group mode. Consider a Group 1 unit, let Z be the random variable describing the number of occurrences $F_k = 1$ out of the simulated $m = 10$ features. Even knowing the groups modes, if $Z = 5$ then the researcher cannot

Table 5
ARI of 3 algorithms for clustering.

p_c	p_{in}	p_{out}	k-Means	Clique	g-Clique
0.60	0.20	0.04	0.038	0.058	0.136
0.60	0.20	0.04	0.021	0.013	0.121
0.60	0.20	0.04	0.255	0.275	0.374
0.60	0.20	0.04	0.020	−0.004	0.083
0.60	0.20	0.04	−0.006	0.002	0.131
0.60	0.20	0.04	−0.019	−0.018	0.075
0.60	0.20	0.04	−0.018	0.081	0.163
0.60	0.20	0.04	0.039	0.098	0.123
0.60	0.20	0.04	0.059	0.071	0.135
0.60	0.20	0.04	0.300	0.050	0.339
0.60	0.16	0.04	0.177	0.125	0.192
0.60	0.16	0.04	0.144	0.157	0.231
0.60	0.16	0.04	0.347	0.195	0.267
0.60	0.16	0.04	−0.014	0.019	0.084
0.60	0.16	0.04	−0.006	0.192	0.404
0.60	0.16	0.04	0.177	0.214	0.257
0.60	0.16	0.04	0.143	0.122	0.137
0.60	0.16	0.04	0.215	0.110	0.093
0.60	0.16	0.04	−0.018	0.017	0.091
0.60	0.16	0.04	−0.004	0.007	0.216
0.60	0.12	0.04	0.039	0.003	0.147
0.60	0.12	0.04	0.006	0.073	0.055
0.60	0.12	0.04	0.177	0.177	0.298
0.60	0.12	0.04	0.214	0.140	0.193
0.60	0.12	0.04	0.060	−0.019	0.225
0.60	0.12	0.04	0.085	0.215	0.203
0.60	0.12	0.04	0.215	0.301	0.342
0.60	0.12	0.04	0.060	0.138	0.130
0.60	0.12	0.04	−0.006	−0.012	0.145
0.60	0.12	0.04	0.143	0.017	0.161
0.65	0.20	0.04	0.020	0.263	0.336
0.65	0.20	0.04	0.300	0.193	0.328
0.65	0.20	0.04	0.347	0.372	0.541
0.65	0.20	0.04	0.509	0.451	0.472
0.65	0.20	0.04	0.451	0.422	0.490
0.65	0.20	0.04	−0.006	−0.022	0.060
0.65	0.20	0.04	−0.019	0.133	0.138
0.65	0.20	0.04	0.509	0.374	0.429
0.65	0.20	0.04	0.299	0.203	0.456
0.65	0.20	0.04	0.215	0.089	0.239
0.65	0.16	0.04	0.299	0.203	0.328
0.65	0.16	0.04	0.214	0.237	0.344
0.65	0.16	0.04	0.508	0.415	0.347
0.65	0.16	0.04	0.452	0.400	0.430
0.65	0.16	0.04	0.143	0.144	0.205
0.65	0.16	0.04	0.397	0.637	0.774
0.65	0.16	0.04	0.569	0.481	0.637
0.65	0.16	0.04	0.011	0.206	0.346
0.65	0.16	0.04	0.397	0.127	0.356
0.65	0.16	0.04	0.177	0.105	0.208
0.65	0.12	0.04	0.300	0.200	0.322
0.65	0.12	0.04	0.397	0.507	0.645
0.65	0.12	0.04	0.509	0.422	0.484
0.65	0.12	0.04	0.021	0.283	0.309
0.65	0.12	0.04	0.398	0.236	0.465
0.65	0.12	0.04	0.112	0.182	0.461
0.65	0.12	0.04	0.509	0.449	0.474
0.65	0.12	0.04	0.214	0.320	0.325
0.65	0.12	0.04	0.021	0.318	0.132
0.65	0.12	0.04	0.397	0.346	0.294
0.55	0.20	0.04	−0.017	−0.024	0.066
0.55	0.20	0.04	−0.006	−0.014	0.056
0.55	0.20	0.04	0.005	0.010	0.129
0.55	0.20	0.04	0.059	0.066	−0.015
0.55	0.20	0.04	0.039	−0.025	0.092
0.55	0.20	0.04	0.144	0.059	0.140
0.55	0.20	0.04	−0.006	0.014	0.048
0.55	0.20	0.04	0.038	0.105	0.265
0.55	0.20	0.04	−0.006	−0.024	−0.011
0.55	0.20	0.04	0.042	0.027	−0.017
0.55	0.16	0.04	−0.006	−0.024	0.017
0.55	0.16	0.04	−0.006	0.066	0.129
0.55	0.16	0.04	0.038	0.035	0.035
0.55	0.16	0.04	0.005	0.002	0.087
0.55	0.16	0.04	−0.014	0.056	0.168
0.55	0.16	0.04	−0.014	0.043	0.092
0.55	0.16	0.04	0.177	0.145	0.308
0.55	0.16	0.04	−0.019	−0.005	0.104
0.55	0.16	0.04	0.005	0.004	0.027
0.55	0.16	0.04	0.006	−0.024	0.064
0.55	0.12	0.04	0.007	0.020	0.201
0.55	0.12	0.04	0.040	−0.025	0.015
0.55	0.12	0.04	0.020	0.017	0.084
0.55	0.12	0.04	0.059	0.060	0.157
0.55	0.12	0.04	0.005	0.018	0.020
0.55	0.12	0.04	0.060	−0.011	0.010
0.55	0.12	0.04	−0.021	−0.012	−0.000
0.55	0.12	0.04	−0.014	0.002	0.036
0.55	0.12	0.04	0.041	−0.011	0.042
0.55	0.12	0.04	−0.019	0.018	0.145
mean			0.135	0.134	0.215

Table 6
ARI averages controlling for p_c.

	0.55	0.6	0.65
k-means	0.021	0.095	0.289
clique	0.019	0.094	0.290
g-Clique	0.083	0.185	0.379

Table 7
ARI averages controlling for p_{in}.

	0.12	0.16	0.20
k-means	0.135	0.150	0.120
clique	0.146	0.147	0.110
g-Clique	0.217	0.233	0.197

determine the unit group membership and if $Z \leq 4$ the unit membership is inferred to the wrong group. Working out the calculation with $m = 10$ and $p_c = 0.65$, $\Pr[Z = 5] \approx 0.15$ and $\Pr[Z \leq 4] \approx 0,095$, while with $p_c = 0.55$, $\Pr[Z \leq 4] \approx 0.25$ and $\Pr[Z = 5] \approx 0.24$.

Three clustering methods are compared. The first two methods are the k-means and the clique-partition, two methods that do not consider the network structure of the data. The third method is the g-Clique presented in this paper. The k-means algorithm is run with k equal to the exact value of 2, an assumption that seldom can be made in practice. The methods are compared in terms of the ARI, the adjusted rand index, (Hubert & Arabie, 1985). The ARI is an index that compares the true and the estimated partitions of a population. It is equal to 1 if the true and the estimated partition are equal, it is close to 0 (with the possibility of being negative) if the estimated partition is equivalent to the random one.

The results of the experiment are reported in Table 5. Looking at the averages, it can be seen that the best ARI has been obtained by the g-Clique, with a value of approximately 0.21, while the k-means and the clique partition gave approximately the same result of 0.13. The ARI of the g-Clique algorithm is almost always better than the one of the other methods, the only exceptions are a few cases in which clusters are so hard to detect that all the ARIs are approximately equal to 0. Tables 6 and 7 reports the ARI averages controlling on values of p_{in} and p_c. It can be seen that the ARI decreases as the values of p_c decrease, while connectivity do not have a monotone behavior: increasing the values of p_{in}, the ARI of g-Clique first increases but then decreases, still remaining well above the ARIs of the other methods.

6. Conclusions

In this paper a new combinatorial model has been proposed to detect clusters for all those cases in which individual and relational

data are available. Two solution approaches have been pursued. The first has been to cast the problem as an integer linear programming model and take advantage of the constraints structure. As it was seen, a new family of valid inequalities to represent connection within a block with an exponential number of constraints is introduced and a separation procedure to handle them in an efficient way is also presented. This approach allowed to solve optimally problems up to 40 units. The second approach has been to implement local search heuristic algorithms to take advantage of the problem combinatorial structure and the local optimum definition. By this way, instances of up to 100 units were approximately solved in a satisfactory way, that is, the ARI of the resulting clustering is much better than any other clustering method.

The results on the ARI of the previous section shows that we introduced a class of clustering methods that are worth exploring further. Our future research will be devoted to improve the methodology introduced here. First, exact algorithms can be implemented using column generation. This methodology has been previously proposed for clique partitioning but it should be tailored to the model in which units must be connected. The second stream of research will be devoted to improve the heuristic algorithms proposed here. Both RR and VNS can be improved trading off accuracy with time. We may want faster methods to cope with larger data sets, in that case constructive heuristics (like hierarchical trees) may serve this goal. But it can also be the case that the most accurate calculation of the objective function is needed, so that VNS or RR can be sophisticated with all the tricks that are common to local search heuristics. The third stream of research can be devoted to model enhancement, to test whether the information provided by the connectivity can be improved by other graph topologies, like the modularity constraints, in the stream of the community detection research rooted in Newman and Girvan (2004).

Acknowledgments

This research has been partially supported by Spanish Ministry of Education and Science/FEDER grants numbers MTM2013-46962-C02-(01–02), MTM2016-74983-C2-(1–2)-R, Junta de Andalucía grant number FQM 05849.

References

Bansal, N., Blum, A., & Chawla, S. (2004). Correlation clustering. *Machine Learning, 56*(1–3), 89–113.

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. (2008). Categorical data fuzzy clustering: An analysis of local search heuristics. *Computers and Operations Research, 35*(3), 766–775.

Bertsimas, D., & King, A. (2016). Or forum- an algorithmic approach to linear regression. *Operations Research, 64*(1), 2–16.

Bertsimas, D., & Shioda, R. (2007). Classification and regression via integer optimization. *Operations Research, 55*(2), 252–271.

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

Cafieri, S., Hansen, P., & Mladenovic, N. (2014). Edge-ratio network clustering by variable neighborhood search. *European Physical Journal B, 87*(5). doi:10.1140/epjb/e2014-50026-4.

Charikar, M., Guruswami, V., & Wirth, A. (2005). Clustering with qualitative information. *Journal of Computer and System Sciences, 71*(3), 360–383. doi:10.1016/j.jcss.2004.10.012.

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* (pp. 1248–1253).

Gavish, B. (1983). Formulations and algorithms for the capacitated minimal directed tree problem. *Journal of the Association for Computing Machinery, 30*(1), 118–132.

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.

Grötschel, M., & Wakabayashi, Y. (1990). Facets of the clique partitioning polytope. *Mathematical Programming, 47*(3, (Ser. A)), 367–387.

Hansen, P., Mladenovic, N., & Moreno Perez, J. (2010). Variable neighbourhood search: Methods and applications. *Annals of Operations Research, 175*(1), 367–407.

Hansen, P., Ruiz, M., & Aloise, D. (2012). A VNS heuristic for escaping local extrema entrapment in normalized cut clustering. *Pattern Recognition, 45*(12), 4337–4345.

Hubert, L., & Arabie, P. (1985). Comparing partitions. *Journal of Classification, 2*(1), 193–218.

Inglehart, R., & Baker, W. (2000). Modernization, cultural change, and the persistence of traditional values. *American Sociological Review, 65*(1), 19–51.

Jaehn, F., & Pesch, E. (2013). New bounds and constraint propagation techniques for the clique partitioning problem. *Discrete Applied Mathematics, 161*(13–14), 2025–2037.

Johnson, E. L., Mehrotra, A., & Nemhauser, G. L. (1993). Min-cut clustering. *Mathematical Programming, 62*(1, Ser. B), 133–151.

Landete, M., & Marín, A. (2014). Looking for edge-equitable spanning trees. *Computers and 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.

Marcotorchino, F., & Michaud, P. (1982). Agrégation de similarités en classification automatique. *Review of Statistics and Its Application, 30*(2), 21–44.

Martí, R., Resende, M. G. C., & Ribeiro, C. C. (2013). Multi-start methods for combinatorial optimization. *European Journal of Operational Research, 226*(1), 1–8.

McPherson, M., Smith-Lovin, L., & Cook, J. (2001). Birds of a feather: Homophily in social networks. *Annual Review of Sociology, 27*, 415–444.

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

Miyauchi, A., & Sukegawa, N. (2015). Redundant constraints in the standard formulation for the clique partitioning problem. *Optimization Letters, 9*(1), 199–207.

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, 18th international joint conference on artificial intelligence. Acapulco, Mexico*.

Newman, M., & Girvan, M. (2004). Finding and evaluating community structure in networks. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics, 69*(2 2), 026113–1–026113–15.

Swamy, C. (2004). Correlation clustering: Maximizing agreements via semidefinite programming. In *Proceedings of the fifteenth annual ACM-SIAM symposium on discrete algorithms* (pp. 526–527(electronic)). ACM, New York.

Wasserman, M., & Faust, K. (1994). *Social networks analysis: Methods and applications*. Cambridge University Press.

Xu, Z., Ke, Y., Wang, Y., Cheng, H., & Cheng, J. (2014). GBAGC: A general Bayesian framework for attributed graph clustering. *ACM Transactions on Knowledge Discovery from Data, 9*(1), 5:1–5:43.

Zhou, Y., Hao, J.-K., & Goëffon, A. (2016). A three-phased local search approach for the clique partitioning problem (English) Zbl 06620838. *J. Comb. Optim., 32*(2), 469–491.