Formulation and Branch-and-cut algorithm for the Minimum Cardinality Balanced and Connected Clustering Problem

Alexandre Salles da Cunha*
Departamento de Ciência da Computação, Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
acunha@dcc.ufmg.br

ABSTRACT

Given a graph $G = (V, E)$, integer bounds $l, u : 0 \leq l \leq u$ and positive integer weights assigned to the vertices V of G, the Minimum Cardinality Balanced and Connected Clustering Problem (MCBCCP) consists of finding a minimum cardinality partitioning of the vertices of V so that (a) each set in the partition induces a connected subgraph of G and (b) the sum of the weights of the vertices in each set belongs to the interval $[l, u]$. In this paper, we present an integer programming formulation, valid inequalities and a Branch-and-cut algorithm for MCBCCP. Our computational experiments suggest that the inequalities investigated here help the overall performance of the algorithm. In addition to the one tested here, we discuss other formulations, following different modeling arguments. Some of them are based on MCBCCP's connections to other combinatorial optimization problems found in the literature.

KEYWORDS

Combinatorial Optimization, Clustering, Spanning forests, Branch-and-cut algorithms

1 INTRODUCTION

A K partition of the vertex set of an undirected graph $G = (V, E)$ ($n = |V|, m = |E|$) is a collection $\{V_1, \ldots, V_K\}$ of K non-empty pairwise disjoint subsets of V such that $\cup_{j=1}^{K} V_j = V$. The vertices in the same partition define a cluster. Assume that positive integer weights $\{w_i \in \mathbb{Z} : i \in V\}$ are assigned to the vertices of G and that integer bounds $l, u : 0 \leq l \leq u$ are given. Given $S \subseteq V$, define $w(S)$ as $\sum_{i \in S} w_i$. If the conditions

$$l \leq w(V_j) \leq u, \qquad j = 1, \ldots, K, \tag{1}$$

are satisfied, the clusters are balanced. Define $E(S) = \{\{i, j\} \in E : i, j \in S\}$ as the edges with both endpoints in S and denote by $\mathcal{C}$ the collection of all connected subgraphs of G, including those with just a single vertex and no edge incident to it. If $\{(V_j, E(V_j)) \in \mathcal{C} : j = 1, \ldots, K\}$, the clustering is connected. If $l \leq w(V_j) \leq u$ and $(S, E(V_j)) \in \mathcal{C}$ for every $j = 1, \ldots, K$ the clustering is balanced and connected. Accordingly, each cluster V_j is balanced and connected.

The Minimum Cardinality Balanced and Connected Clustering Problem (MCBCCP) consists of finding a balanced and connected clustering of G of minimum cardinality. MCBCCP is an NP-Hard optimization problem, even for series parallel graphs, but is solvable in linear time in case G is a path [13].

Our goal is to solve MCBCCP when G does not belong to a particular graph class. To that aim, we introduce an integer programming formulation, valid inequalities and an exact solution approach, of the Branch-and-cut type [21]. These topics are addressed in Sections 2 and 3. Our preliminary computational experiments reported in Section 4 suggest that the valid inequalities discussed here were of help to enhance the overall performance of the Branch-and-cut algorithm. In addition to that, we describe, at the last section of this paper, several other modeling strategies that may lead to effective MCBCCP exact algorithms. We also highlight connections between MCBCCP and some network design problems, that may be explored from a modeling and algorithmic perspective.

In the remaining of this section, we review some clustering problems, concentrating mostly on those that closely relate to MCBCCP, and thus, require clusters to be either balanced or to satisfy some kind of connectivity constraint.

Graph clustering is a central problem in Operations Research, Computer Science and Artificial Intelligence. They arise in applications as diverse as circuit board and micro-chip design, parallel computation, sparse matrix factorization and data mining [12]. Depending on the objective function and on the similarity criteria used to group vertices, different clustering problems arise. Not surprisingly, the literature on the topic is vast; a review of several clustering problems can be found, for instance, in [25].

Ito et al. [13] discussed three problems related to optimal choosing connected and balanced clusters. One of them consists of deciding whether or not G has a balanced and connected clustering of fixed size p. The other two are MCBCCP and the problem that maximizes the number of clusters. All of them arise in practical applications such as political districting, paging systems of operation systems and image processing. Ito et al. [13] was concerned with MCBCCPs defined on trees. To that particular input graph class, the authors presented the first polynomial time algorithm.

A common sense in graph clustering is that similar vertices should be grouped together, while dissimilar ones should be separated. Quite often, the similarity of a pair of vertices i, j is *measured* by a weight c_{ij}. Therefore, an objective function that arises frequently in practice, specially for cardinality constrained clustering problems, is the minimization of the sum of the weights of the edges connecting vertices in different clusters [1, 8, 11, 12, 15, 23, 24, 27]. Since $\sum_{\{i,j\} \in E} c_{ij}$ is a constant, an equivalent problem consists of maximizing the sum of the weights of the edges connecting vertices in the same clusters. In general, graph clustering is NP-Hard [9].

The problem of finding optimal balanced clusterings of fixed cardinality has received substantial attention in the literature. Jonhson et al. [14] investigated one such problem. In addition to integer bounds l, u and weights $\{w_i : i \in V\}$, costs $\{c_{ij} : \{i, j\} \in E\}$ are also assigned to the edges of E. The problem thus consists of finding a K balanced clustering of G that minimizes the function $\sum_{k=1}^{K} \sum_{\{i,j\} \in E(V_k)} c_{ij}$. In that particular problem, input graphs are not complete and clusters do not need to induce connected subgraphs of G.

*This research is partially funded by CNPq grants 303928/2018-2, 431369/2016-0 and FAPEMIG grants CEX-PPM-00164/17, APQ-02645-16.

Clustering under some sort of connectivity requirements has also been a topic of interest [3, 5, 7, 17]. In Lari et al. [17], the number of clusters K is defined beforehand. There is a subset S ($K = |S|$) of V that plays the role of clusters heads. The remaining vertices, those in the set $U = V \setminus S$, are denoted unit vertices. The problem asks for a K-centered connected partition of G: a partition of the vertices of G into K connected subgraphs $\{(V_j, E(V_j)) : j = 1, \ldots, K\}$, such that $|V_j \cap S| = 1$ for each $j = 1, \ldots, K$. One cluster head must be assigned to each cluster and unit vertices must be assigned to one vertex in S, the head of one cluster. The cost of assigning a unit vertex $u \in U$ to a cluster head $i \in S$ is denoted c_{ui}. The cost of a component (or cluster) V_j is $\sum_{i \in S \cap V_j} \sum_{u \in V_j \cap U} c_{ui}$. Additionally, weights $\{w_v : v \in V\}$ are assigned to each vertex of G. Lari et al. [17] investigated the problem of finding K centered connected partitions of G for different min-max objective functions involving either the assignment costs or the vertices' weights. The complexity class of each of these variants was also investigated.

Brucker [3] investigated the problem of clustering the vertices of V in K or fewer sets such that each vertex set induces a subgraph with diameter at most p; the diameter being the longest minimum shortest path between two vertices in the subgraph. The problem was proven to be NP-Complete and remains so even if $K = 3$ and all edges' lengths are taken from the set $\{0, 1\}$. Deogun et al. [6] investigated the same K clustering problem under binary edge costs. They introduced approximation results and algorithms, specialized for certain classes of input graphs.

Edachery et al. [7] introduced the Partition into Distance $p-$cliques Problem, another optimization problem whose decision version was proven to be NP-Complete. A vertex set $S \subseteq V$ is a distance $p-$clique if, for any pair of vertices $i, j \in S$, there is a path in the graph $(S, E(S))$ that involves at most p edges (or hops). The Partition into Distance $p-$cliques Problem thus consists of partitioning the vertex set of V in to the least number of distance $p-$cliques. For solving the problem, Edachery et al. [7] introduced heuristics and addressed their performance on various large scale graphs found in the telecommunication industry.

Nossack and Pesch [20] investigated the acyclic clustering problem, a problem defined in terms of a vertex and arc weighted directed graph $D = (V, A)$. A feasible solution to the problem is a K balanced clustering, with one additional property: the graph obtained by shrinking each cluster into a single vertex and by merging arcs that start and end at the same pair of inbound and outbound clusters must be a directed acyclic graph. The objective function consists of maximizing the sum of the weights of the arcs with both endpoints in the same cluster.

Finally, Aragão and Uchoa [5] investigated the γ-Connected Assignment Problem ($\gamma-$CAP). Given $G = (V, E)$, a set of colors $K = \{1, \ldots, k\}$, a vector $\gamma = (\gamma_1, \ldots, \gamma_k)$ of positive integers and costs $\{c_{iq} : i \in V, q \in K\}$, the γ-Connected Assignment Problem consists of finding a minimum cost assignment of colors to the vertices in such a way that no set of vertices assigned to the same color q induces a subgraph of G with more than γ_q connected components. Applications of the $\gamma-$CAP can be seen as variants of the Contiguity Constrained Clustering Problem [18, 19].

2 MCBCCP FORMULATION

Among other decision variables to be defined shortly, the model uses an integer variable K to denote the cardinality of the clustering we are looking for. Assuming that $l > 0$ holds, feasible values for K must satisfy $\left\lceil \frac{n}{\lfloor \frac{u}{\min\{w_i : i \in V\}} \rfloor} \right\rceil \leq K \leq \left\lfloor \frac{n}{\lceil \frac{l}{\max\{w_i : i \in V\}} \rceil} \right\rfloor$. Let $\overline{K}$ denote an upper bound on the maximum number of balanced clusters of G. If $l > 0$, $\overline{K}$ can be taken as the upper bound in the previous expression. If $l = 0$, $\overline{K} = n$.

The idea behind the formulation is to find a spanning forest of G with precisely $n - K$ edges, such that the vertices in each of its K trees define balanced clusters. Define $K_i = \min\{i, \overline{K}\}$. For a given $i \in V$, the set $\{1, \ldots, K_i\}$ gives the indexes of connected components where i can be placed in. In addition to K, the formulation uses the following decision variables:

- $\mathbf{x} = \{x_{ij} \in \mathbb{B} : \{i, j\} \in E\}$. Variable x_{ij} assumes value 1 if edge $\{i, j\}$ belongs to the forest and 0 if otherwise applies. For any subset $E' \subseteq E$, define $x(E') = \sum_{\{i,j\} \in E'} x_{ij}$.
- $\mathbf{y} = \{y_i^k \in \mathbb{B} : i \in V, k = 1, \ldots K_i\}$. Variable y_i^k assumes value 1 if vertex i belongs to the $k-$th connected component of the forest.
- $\mathbf{z} = \{z^k \in \mathbb{B} : k = 1, \ldots, \overline{K}\}$. Variable z^k assumes value 1 if and only if the forest has k or more connected components.

The formulation is:

$$\min \left\{ K : (K, \mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{P} \cap (\mathbb{R} \times \mathbb{B}^m \times \mathbb{B}^\sigma \times \mathbb{B}^{\overline{K}}) \right\}, \quad (2)$$

where $\sigma := \sum_{i=1}^{n} K_i$ and $\mathcal{P}$ is the polyhedral region defined by:

$$x(E) + K = n \quad (3)$$

$$x(E(S)) \leq |S| - 1 \qquad S \subset V, |S| \geq 2 \quad (4)$$

$$\mathbf{x} \geq 0 \quad (5)$$

$$\sum_{k=1}^{K_i} y_i^k = 1 \qquad i \in V \quad (6)$$

$$\sum_{i=k}^{n} y_i^k \geq z^k \qquad k = 1, \ldots, \overline{K} \quad (7)$$

$$\sum_{i=k}^{n} w_i y_i^k \leq u z^k \qquad k = 1, \ldots, \overline{K} \quad (8)$$

$$\sum_{i=k}^{n} w_i y_i^k \geq l z^k \qquad k = 1, \ldots, \overline{K} \quad (9)$$

$$x_{ij} + y_i^\tau + \sum_{k=1}^{\tau-1} y_j^k + \sum_{k=\tau+1}^{K_j} y_j^k \leq 2 \qquad \substack{\{i,j\} \in E, i < j \\ \tau = 1, \ldots, K_i} \quad (10)$$

$$\sum_{k=1}^{K_i} k y_i^k - \sum_{k=1}^{K_j} k y_j^k \geq -M_{ij}(1 - x_{ij}) \qquad \{i, j\} \in E \quad (11)$$

$$\sum_{k=1}^{K_i} k y_i^k - \sum_{k=1}^{K_j} k y_j^k \leq M_{ij}(1 - x_{ij}) \qquad \{i, j\} \in E \quad (12)$$

$$\sum_{k=1}^{\overline{K}} z^k = K \quad (13)$$

$$z^k \leq z^{k-1} \qquad k = 2, \ldots, \overline{K} \quad (14)$$

$$y_i^k \leq z^k \qquad \substack{i \in V \\ k = 1, \ldots, K_i} \quad (15)$$

$$y_i^k \geq 0 \qquad \substack{i \in V \\ k = 1, \ldots, K_i} \quad (16)$$

Aiming to cope with formulation symmetries, variables y_i^k for $k > K_i$ are not used; only y_i^k for $k = 1, \ldots K_i$ are in place.

Constraints (3)-(5) model the spanning forest with $n - K$ edges of G. Subtour elimination constraints (SECs) (4) avoid that K is

decreased to an infeasible value, by selecting edges in excess of what an acyclic subgraph $(S, E(S))$ of G allows.

Constraints (6) enforce that each vertex must be assigned to one connected component of G. These constraints make a clear distinction of the indexes of components that can be assigned to the vertices. More precisely, they state that vertex $i = 1$ can only belong to the first connected component ($K_1 = \{1\}$). Likewise, vertex $i = 2$ either belongs to the first connected component or to the second, and so on. Constraints (8)-(9) enforce that the sum of weights of each connected component lies in the desired interval $[l, u]$. Constraints (7) impose that each cluster must include at least one vertex of V.

Note that inequalities (10) are a lifting of the logical constraints

$$x_{ij} + y_i^{k_1} + y_j^{k_2} \leq 2, k_1 \in K_i, k_2 \in K_j, k_1 \neq k_2, \{i,j\} \in E. \quad (17)$$

The latter, as well as its stronger version (10), enforces that edge $\{i,j\}$ cannot belong to the forest if its endpoints are assigned to two different connected components. Disjunctive constraints (11) and (12) serve the same purpose, but use different modeling arguments; they enforce that an edge $\{i,j\}$ can only be included in the forest if its endpoints are assigned to the same connected component index. For these constraints, big-M parameter M_{ij} represents the maximum difference between the indexes of the clusters where vertices i and j are placed in and is given by

$$M_{ij} = \min\{\max\{i,j\}, \overline{K}\} - 1.$$

Note that if $x_{ij} = 0$, constraints (11) and (12) are trivially satisfied. However, if $x_{ij} = 1$, $\sum_{k=1}^{K_i} k y_i^k = \sum_{k=1}^{K_j} k y_j^k$ must hold. Because of the discreteness of $\mathbf{y}$ and due to (6), $y_i^k = y_j^k$ must hold, for a given $k = 1, \ldots, \min\{K_i, K_j\}$. Although we do not have numerical or theoretical evidence showing that constraints (11) and (12) improve the linear programming bounds, provided that (10) are in place, they were kept in the model. We decided to do so, since we empirically found that their inclusion helped the overall performance of the Branch-and-cut algorithm.

Constraints (13) state that the number of clusters is precisely the number of variables z^k activated. Constraints (15) impose that a vertex i cannot belong to a cluster k unless the forest has at least k components.

2.1 Additional valid inequalities

In the remaining of the text, assume that $U(S)$ denotes the minimum number of bins of size u, required to fit the vertices in $S \subseteq V$. Accordingly, let $\underline{U}(S)$ denote any valid lower bound on $U(S)$. A widely known lower bound on $U(S)$ is $\lceil \frac{w(S)}{u} \rceil$.

Subtour elimination constraints (4) can be lifted to the *capacity constraints* (CC)

$$x(E(S)) \leq |S| - \underline{U}(S), \qquad \forall S : S \subset V, |S| \geq 2. \quad (18)$$

To the best of our knowledge, capacity constraints (18) were introduced for the Capacitated Vehicle Routing Problem in [16]. In that context, weights $\{w_i : i \in V\}$ represent demands that must be collected by a vehicle of capacity u and $U(S)$ denotes the minimum number of vehicles of capacity u needed to collect the total demand $w(S)$ associated to S. Here, these inequalities avoid that clusters with weights exceeding u induce trees of the forest. Although such conditions are already granted by constraints (8), the inclusion of CCs (18) improves linear programming relaxation bounds. From now on, assume that $\mathcal{P}^+$ denotes the intersection of polytope $\mathcal{P}$ with capacity constraints (18).

The formulation can also be strengthened by the capacity cutset constraints (CCC)

$$x(\delta(S)) \geq 1 \quad (19)$$

defined by sets $S \neq \emptyset, S \subset V$, satisfying the following conditions:

(1) $(S, E(S)) \in C$;
(2) $w(S) < l$.

In inequalities (19), $\delta(S) = \{\{i,j\} \in E : i \in S, j \notin S\}$ stands for the subset of edges with exactly one endpoint in $S \subset V$. Validity of inequalities (19) for MCBCCP comes from the following observations. Let $V_1 \subset V$ be a balanced and connected cluster such that $S \cap V_1 \neq \emptyset$. Since $w(S) < l$, it is clear that $V_1 \setminus S \neq \emptyset$ since otherwise $w(V_1) < l$ would hold. Since $(V_1, E(V_1))$ is connected, at least one edge connecting S to $V_1 \setminus S$ must be chosen.

CCCs are also valid for subsets $S : w(S) < l$ whose subgraphs $(S, E(S))$ are not connected. However, for this case, the right hand side can be lifted to the number of connected components of $(S, E(S))$ and the inequality can be seen as a (weaker) surrogate version of those defined by the connected subsets contained in S. For the remaining of the text, assume that $\mathcal{P}^{++}$ denotes the intersection of $\mathcal{P}^+$ with CCCs (19).

3 BRANCH-AND-CUT ALGORITHM

In this section, we describe the main features of the MCBCCP Branch-and-cut algorithm BC^{++} based on formulation $\mathcal{P}^{++}$. The algorithm dynamically separates two classes of MCBCCP valid inequalities, CCs (18) and CCCs (19). BC^{++} first solves the Linear Program (LP)

$$\min \left\{ K : (K, \mathbf{x}, \mathbf{y}, \mathbf{z}) \in \hat{\mathcal{P}} \right\}, \quad (20)$$

where $\hat{\mathcal{P}}$ is the polytope given by the intersection of constraints (3), SECs (4) defined by sets $S = \{i,j\} \in E$, (5)-(16) and

$$x(\delta(S)) \geq 1, \ S \in \mathcal{S}, \quad (21)$$

where $\mathcal{S} = \{S \subset V : w(S) < l, 1 \leq |S| \leq Max, (S, E(S)) \in C\}$. We found advantageous to include all CCCs defined for sets $S : |S| \leq Max$ in the very first linear programming relaxation (in our implementation, $Max = 4$). CCCs defined by sets with more than Max vertices are identified on-the-fly, as described in the sequence.

Assume that the LP (20) is feasible and denote by $(\hat{K}, \hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}})$ an optimal solution to it. BC^{++} then attempts to strengthen the relaxation $\hat{\mathcal{P}}$ by appending to it some CCs and CCCs, violated by $\hat{\mathbf{x}}$. The separation procedures for these two sets of valid inequalities makes use of a separation engine designed for the SEC (4) separation problem, outlined in [2]. The engine comprises heuristic and exact algorithms for the identification of violated SECs. The idea is to use these SEC separation procedures to provide candidate sets of vertices for which the violation of CCCs and CCs can be checked.

More precisely, the SEC separation engine involves a Kruskal like heuristic and the exact SEC separation algorithm introduced in [22]. The heuristic first sorts the edges in $\hat{E} = \{\{i,j\} \in E : \hat{x}_{ij} > 0\}$, in a non increasing order of their linear programming relaxation values $\{\hat{x}_{ij} : \{i,j\} \in \hat{E}\}$. In turn, edges in the list are used to find a maximum cardinality spanning forest of G, in a Kruskal like fashion. Assuming that $e = \{i,j\}$ is the edge to be processed, the heuristic merges the connected components where i and j are placed into a single subset S of vertices. The lower bound $\underline{U}(S) = \lceil \frac{w(S)}{u} \rceil$ is computed and the corresponding CC (18) is checked for violation. The same set S is checked for the

identification of a violated CCC (19): whenever $w(S) < l$, we check if $\hat{x}(\delta(S)) < 1$ applies.

Violated CCs and CCCs are stored in two separate lists. At the end of the application of the heuristic, the most violated inequality in each list reinforces the relaxation $\hat{\mathcal{P}}$. Remaining inequalities in the list are only included in $\hat{\mathcal{P}}$ if they are sufficiently orthogonal to the most violated inequality of its class, found at that separation round. To be more specific, assume that $\mathbf{a}_1^T\mathbf{x} \geq \mathbf{b}_1$ and $\mathbf{a}_2^T\mathbf{x} \geq \mathbf{b}_2$ are two violated valid inequalities of the same class, CC or CCC, stored in the list, at that separation round. Assume as well that the first is the most violated inequality of that class. Inequality $\mathbf{a}_2^T\mathbf{x} \geq \mathbf{b}_2$ is only added to $\hat{\mathcal{P}}$ if $\frac{|\langle \mathbf{a}_1, \mathbf{a}_2 \rangle|}{\|\mathbf{a}_1\|_2 \|\mathbf{a}_2\|_2} \leq \epsilon$, where $\epsilon \in [0,1)$ is an implementation parameter that controls the desired level of orthogonality (in our implementation, $\epsilon = 0.5$). Inequalities that do not satisfy the orthogonality criteria are discarded.

The exact SEC separation algorithm in [22] is called next, only if no violated inequalities were found by the SEC separation heuristic. Without giving much of the details of that procedure, it suffices to mention that the algorithm solves a series of $n-2$ max-flow (min-cut) problems in a conveniently defined directed network, obtained from $\hat{\mathbf{x}}$ and $\hat{E}$, in order to find violated SECs. The optimal set of vertices in each of these min-cut problems is checked for the violation of CCs and CCCs. Again, the same orthogonality criteria is applied in order to decide which inequalities stored in the lists are discarded or used to reinforce the relaxation $\hat{\mathcal{P}}$.

The process outlined above is carried out for each BC^{++} node, until no violated inequalities are found by the separation engine. If the solution to that linear programming relaxation is integer feasible, the optimal solution to the node under investigation was found, and the node is pruned by optimality. Being fractional, BC^{++} branches on variables, giving preference to branch first on variables $\mathbf{x}$.

BC^{++} makes use of the `XPRESS` mixed integer optimization package (release 27.01.02) in charge of managing the search tree. All pre-processing and cutting plane generation procedures embedded in the solver are turned off. No multi-threading is allowed. Since we still have not implemented MCBCCP primal heuristics to speed up the search, preference is given to find feasible solutions as early as possible. Thus, BC^{++} implements a depth-first search and the solver's linear programming heuristics are turned on.

4 COMPUTATIONAL EXPERIMENTS

Our computational experiments were conduced with two sets of test instances: `cb` and `g`. The first one, `cb`, comprises benchmark instances for the min-cut problem addressed by Johnshon et al. [14]. They represent real compiler construction problems, where each vertex in the (typically very sparse) input graph represent modules of code that need to be combined to form clusters. In total, five graphs representing the compiler construction problem were used. Each of these instances are indicated by `cb_`*id*, where *id* represents one of the five graphs. In that application, edges' weights represent the communication cost between modules of code. In the MCBCCP case, these weights are not applicable, being simply ignored. In the application described in [14], clusters are restricted in their total memory storage. We used the same values of $u \in \{450, 512\}$ considered in [14]. Since in that reference l was assumed to be zero, we used $l = \lfloor 10\%u \rfloor$ and $l = \lfloor 20\%u \rfloor$.

The second set of instances considered here, `g`, represent grid graphs. These instances were generated here, in an attempt to address another application of MCBCCP, that of clustering geographical contiguous areas into political districts. Each grid has $s \in \{8, 12\}$ rows and columns. For each value of s, five instances were generated. Each instance is identified by the word `g_s_id`, where $id \in \{1, \ldots, 5\}$ is an integer representing a particular instance of size s. One vertex, placed at the center of each of these s^2 grid cells, represents the cell. The set E includes one edge $\{i, j\}$ for each pair of neighboring cells i and j. Integer weights $\{w_i : i \in V\}$ were randomly chosen in the interval $[10, 100]$, with uniform probability. Depending on the values of s, different values of u were chosen: for $s = 8$, $u = 350$ and for $s = 12$, $u = 650$. These values of u account for about 10% of $w(V)$. The values of l were chosen as before: $l = \lfloor 10\%u \rfloor$ and $l = \lfloor 20\%u \rfloor$. Considering the different values of u and l involved in our experiments, 40 MCBCCP instances were tested. Data for the instances used in our testings are given in Table 1. That table provides, for each instance, the corresponding values of n, m, the minimum and the maximum values of w_i and, finally, $w(V)$.

The algorithms outlined in this paper were implemented in `C` and compiled with `gcc` with optimization flags `-O3` turned on, under Linux OS. Experiments were conducted with a Intel i7-5820K processor, running at 3.3GHz, with 32Gbytes of RAM memory.

In Table 2, we present computational results of two Branch-and-cut implementations: BC^{++} and BC^{+}. The first one is based on formulation $\mathcal{P}^{++}$ and, thus, makes use of CCCs (19). The other one does not, being based on the (weaker) model $\mathcal{P}^{+}$. Therefore, BC^{+} neither includes inequalities (21) in its first linear programming relaxation nor calls the procedures for the indentification of violated constraints (19), within the separation engine we described earlier. Apart from that, the two algorithms share every other implementation strategy.

The first two columns of the table provide the instance and the value of u under consideration. The table is divided in two sets of rows; the upper part of the table is dedicated to the $l = \lfloor 20\%u \rfloor$ instances, while the bottom reports results for $l = \lfloor 10\%u \rfloor$. For each BC implementation, the table presents: the root node lower bound (LB), the best lower (BLB) and upper bound (BUB) found at the end of the search or when the time limit of 1800 seconds was

Table 1: Data for the instances used in the computational experiments.

Inst.	n	m	min w_i	max w_i	$w(V)$
cb_1	30	47	19	298	2497
cb_2	45	98	14	298	3325
cb_3	47	99	14	298	3425
cb_4	47	101	14	278	3890
cb_5	61	187	15	165	3704
g8_1	64	112	11	99	3472
g8_2	64	112	13	100	3732
g8_3	64	112	10	100	3272
g8_4	64	112	10	100	3411
g8_5	64	112	12	100	3757
g12_1	144	264	10	100	8000
g12_2	144	264	10	99	7999
g12_3	144	264	11	100	8013
g12_4	144	264	10	100	8153
g12_5	144	264	10	100	7997

hit, the CPU times (in seconds) needed to obtain these bounds, $t(s)$, and finally the number of nodes explored by the search trees. An indication "-" is provided in the CPU time columns, whenever the time limit was hit and the corresponding BC implementation either did not prove the instance infeasibility or did not solve it to proven optimality. Whenever an algorithm did not find a feasible solution within the time limit, ∞ is reported in the corresponding BUB column entry.

The values of l and u involved in our testings led to 39 feasible instances out of the 40 available. BC^{++} managed to prove the infeasibility of that instance quite fast, right at the root node, in contrast to BC^{+}, that spent the entire time limit without concluding so. Considering the 39 feasible instances of our study, better results were also obtained by BC^{++}. While it managed to solve 24 out of the 39 feasible instances to proven optimality within the 1800 seconds time limit, BC^{+} solved only 12. Considering the 11 instances solved to optimality by both methods, computational results also lean in favor of BC^{++}. It was faster than BC^{+} in 7 out of these 11 cases. BC^{++} was also capable of delivering higher quality integer feasible solutions, if our attention now moves to the 14 instances left unsolved by both methods. In all these cases, BC^{++} found sharper upper bounds than BC^{+}. The latter did not find a feasible solution within the time limit for 14 out of the 39 feasible instances of our test set.

It is worthwhile mentioning that the inclusion of inequalities (19) resulted in small (if any) increases in the root node linear programming bounds, for the lowest values of l. That comes as a result of two aspects. The first is the nature of the objective function, that involves the minimization of K, while every other variable has the same null cost. The second is the heuristic nature of our separation engine for the separation problem associated to CCs (18) and to CCCs (19). As a consequence of that, there is no guarantee that the cutting plane algorithm of BC^{++} delivers a stronger bound even if, for a particular instance, the optimal solution $\mathbf{x}^{++}$ to the linear programming relaxation defined by $\mathcal{P}^{++}$ does not belong to $\mathcal{P}^{+}$. That explains why, for two cases, BC^{+} delivered root node lower bounds slightly stronger than those provided by BC^{++}. Nevertheless, the inclusion of (19) as well as the lifting (17) of (10) had a positive impact on the performance of the full search tree. In general, fewer nodes are investigated in less CPU time.

Table 2: Branch-and-cut algorithm: Computational results with and without inequalities (19)

$l = \lfloor 20\%u \rfloor$

		BC^{++}					BC^{+}				
Inst.	u	LB	BLB	BUB	$t(s)$	nodes	LB	BLB	BUB	$t(s)$	nodes
cb_1	450	infeasible			0.2	1	7	9	∞	-	57002
cb_2	450	9	10	10	163.0	7376	8.5	9	∞	-	26262
cb_3	450	9	10	10	106.6	3517	8.2	9	∞	-	30337
cb_4	450	9	9	9	439.6	8556	9	9	10	-	16424
cb_5	450	9	9	9	44.9	818	9	9	∞	-	20373
cb_1	512	6	6	6	0.6	9	5.3	6	6	33.7	7028
cb_2	512	7.5	8	8	80.6	3564	7.3	8	∞	-	28250
cb_3	512	7	8	8	124.2	3145	7	7	∞	-	20071
cb_4	512	8	8	8	79.8	2404	8	8	8	119.9	5621
cb_5	512	8	8	8	31.7	468	8	8	8	216.5	9204
g8_1	350	11	11	11	23.4	521	11	11	11	206.4	8891
g8_2	350	11	12	12	254.3	6165	11	12	13	-	16150
g8_3	350	10	10	10	34.1	766	10	10	11	-	24266
g8_4	350	10	11	11	831.3	7625	10	11	11	434.1	4161
g8_5	350	11	11	12	-	8041	11	12	12	207.0	4161
g12_1	650	13	13	15	-	8902	13	13	∞	-	9912
g12_2	650	13	13	19	-	8558	13	13	∞	-	9336
g12_3	650	13	13	16	-	6591	13	13	∞	-	9954
g12_4	650	13	13	15	-	12252	13	13	19	-	11618
g12_5	650	13	13	15	-	6196	13	13	∞	-	9493

$l = \lfloor 10\%u \rfloor$

		BC^{++}					BC^{+}				
Inst.	u	LB	BLB	BUB	$t(s)$	nodes	LB	BLB	BUB	$t(s)$	nodes
cb_1	450	6.5	8	8	1.5	167	7	8	8	6.1	1069
cb_2	450	9	9	10	-	25546	8.4	9	12	-	23702
cb_3	450	8.3	9	11	-	27518	8.2	9	13	-	23303
cb_4	450	9	9	9	234.3	6198	9	9	9	208.0	5374
cb_5	450	9	9	9	61.5	1531	9	9	12	-	19561
cb_1	512	5.3	6	6	4.0	639	5.2	6	6	12.3	3869
cb_2	512	7.3	8	9	-	29238	7.2	8	10	-	23258
cb_3	512	7	7	8	142.7	6899	7	7	8	-	28586
cb_4	512	8	8	8	24.5	1340	8	8	8	92.2	5426
cb_5	512	8	8	8	29.9	340	8	8	8	42.9	610
g8_1	350	11	11	11	42.8	1275	11	11	11	1775.8	13153
g8_2	350	11.0	12	12	881.8	9273	11.1	12	13	-	14082
g8_3	350	10	10	11	-	9296	10	10	11	-	14329
g8_4	350	10	11	11	1245.9	9267	10	10	11	-	16581
g8_5	350	11	12	12	1399.0	8919	11	12	12	201.8	3141
g12_1	650	13	13	17	-	5432	13	13	∞	-	4079
g12_2	650	13	13	16	-	7043	13	13	∞	-	3938
g12_3	650	13	13	18	-	3129	13	13	∞	-	3723
g12_4	650	13	13	18	-	3747	13	13	∞	-	4445
g12_5	650	13	13	16	-	4302	13	13	∞	-	4941

5 CONCLUSIONS AND FUTURE WORK

In this paper, we investigated the Minimum Cardinality Balanced and Connected Clustering Problem (MCBCCP). We introduced an integer programming formulation along with valid inequalities for the problem. Additionally, we implemented and tested a Branch-and-cut algorithm based on that model.

Our preliminary numerical experiments indicated that the valid inequalities introduced here, the lifting (10) of the logical constraints (17) and the capacity cutset constraints (19), had a positive impact on the computational results. In particular, the Branch-and-cut algorithm that separates capacitated cutset inequalities (19) obtained more optimality certificates than its version that does not. It also found higher quality solutions for those instances left unsolved when the time limit is hit.

The capacity cutset constraints (19) introduced here, alongside capacity constraints (18), allows the problem to be formulated without the need of variables $\mathbf{y}, \mathbf{z}$. To be more specific, denote by $\mathbf{P}_x \subset \mathbb{R}^{m+1}$ the intersection of (3) and (5) alongside the capacity constraints (18) and cutset constraints (19). MCBCCP can be formulated as $\min\{K : (K, \mathbf{x}) \in \mathcal{P}_x \cap (\mathbb{R} \times \mathbb{B}^m)\}$.

A drawback of the formulation introduced here is its symmetry. To alleviate that, we restricted the indexes of the clusters that can be assigned to the vertices of G. The lifting (10) of (17) also helps in that matter. From an algorithmic perspective, we enforced that the Branch-and-cut algorithm first branches on $\mathbf{x}$ variables.

A promising alternative to deal with symmetry is to use the concept of *representatives* [4] to formulate the problem. For the MCBCCP case, such a formulation involves binary decision variables $\mathbf{v} = \{v_{ij} \in \mathbb{B} : i, j \in V\}$. Variable v_{ij} assumes value 1 if

vertex j is the representative (or cluster head) of the connected component where i is placed. Following the strategy to reduce symmetry and the number of $\mathbf{y}$ variables, these variables can be restricted to $\{v_{ij} : i,j \in V, j \leq i\}$, indicating that the representative of a cluster is always the vertex with the least index among those in the same tree of the forest. This formulation does not involve decision variables $\mathbf{y}$ and $\mathbf{z}$ and the objective function, to be minimized, is $\sum_{i \in V} v_{ii}$. In addition to constraints $\{v_{ij} \leq v_{jj}, i,j \in V, j < i\}$, the model includes constraints akin to the inequalities defining $\mathcal{P}^{++}$, except to (14) to (7) that have no meaning in the new variable setting. Preprocessing these $\mathbf{v}$ variables could be carried out as follows. Define $D = (V, A)$ as the directed graph obtained by duplicating the edges of E, into two arcs of opposite directions. Assume that the length of an arc $(i,j) \in A$ is $c_{ij} = w_j$. Whenever the length of the shortest path connecting i and j exceeds u, the representative of i and j cannot be the same, and assuming that $j < i$ applies, variable v_{ij} would not be required.

We also plan to investigate set partitioning formulations for MCBCCP and Branch-and-cut-and-price algorithms based on them. One possible formulation along these lines is similar to the one introduced in [14] for a fixed cardinality clustering problem. In addition to edge variables $\mathbf{x}$, it uses exponentially many binary decision variables associated to all balanced subsets of vertices of V. The master problem is defined by set partitioning constraints enforcing that every vertex must be included in one of such subset of vertices, constraints (19) and (18), as well as another type of constraints that couple $\mathbf{x}$ and the exponentially many variables of sets of vertices. Such coupling constraints express the following idea: *an edge $\{i,j\}$ cannot be selected by the master program unless its endpoints i and j are included in the same balanced set.* Because of that type of coupling constraints, the associated pricing problem consists of solving a Constrained Quadratic Binary Problem, a variation of the Maximum Weight Binary Knapsack Problem (QKP), where not only knapsack constraints $w(S) \leq u$ but also covering constraints $w(S) \geq l$ must be enforced. A nice feature of this problem is that, due to the sign of the dual variables in the master program, diagonal entries of the quadratic cost matrix should be negative while off-diagonal ones should be positive.

MCBCCP also has connections with network design problems found in the literature, for instance, the Capacited Minimum Spanning Tree Problem (CMSTP) [10, 26]. Actually, we can reformulate MCBCCP as a variation of the CMSTP. To that aim, consider a directed graph $D = (V \cup \{r\}, A)$, where the arc set A involves two arcs, in opposite directions, for every edge in E, as well as one arc pointing from the artificial root vertex r to every vertex $i \in V$. Arcs connecting r to $i \in V$ cost one while the remaining ones cost zero. The goal is to find a minimum cost spanning arborescence of D, rooted out of r, such that the sum of the weights of the vertices in every tree rooted in $i \in V$ satisfy (1). We plan to investigate models and algorithms for MCBCCP based on such an idea, including those where the desired arborescence topology is enforced by network flow and set partitioning constraints.

So far, we have not investigated primal heuristics for the problem. One possible approach to fill that gap benefits from the Dynamic Programming algorithm in [13], capable of exactly solving MCBCCP in polynomial time, when G is a spanning tree. The idea is to build a spanning tree of G, driven by the linear programming relaxations $\{\hat{x}_{ij} : \{i,j\} \in \hat{E}\}$ provided by our BC algorithms. In turn, that spanning tree is used as an input graph for the exact MCBCCP algorithm in [13].

These ideas, in full or in part, should complement the material presented in this paper and should be presented at the next International Network Optimization Conference.

REFERENCES

[1] E.R. Barnes. 1982. An algorithm for partitioning the nodes of a graph. *SIAM Journal on Algebraic and Discrete Mathematics* 3 (1982), 541–555.

[2] Bicalho, L., da Cunha, A.S and Lucena, A. 2016. Branch-and-cut-and-price algorithms for the Degree Constrained Minimum Spanning Tree Problem. *Computational Optimization and Applications* 63 (2016), 755–792.

[3] P. Brucker. 1978. *Optimization and Operations Research.* Lecture Notes in Economics and Mathematical Sciences, Vol. 157. Springer, Chapter On the complexity of clustering problems, 45–54.

[4] Manoel Campêlo, Ricardo Corrêa, and Yuri Frota. 2004. Cliques, holes and the vertex coloring polytope. *Inform. Process. Lett.* 89, 4 (2004), 159 – 164.

[5] Marcus Poggi de Aragão and Eduardo Uchoa. 1999. The $\gamma-$connected assignment problem. *European Journal of Operational Research* 118 (1999), 127–138.

[6] Jitender S. Deogun, Dieter Kratsch, and George Steiner. 1997. An approximation algorithm for clustering graphs with dominating diametral path. *Inform. Process. Lett.* 61, 3 (1997), 121 – 127.

[7] J. Edachery, A. Sen, and F. J. Brandenburg. 1999. *Graph drawing.* Lecture Notes in Computer Science, Vol. 1731. Springer, Chapter Graph Clustering Using Distance-k cliques, 98–106.

[8] J. Falkner, F. Rendl, and H. Wolkowicz. 1994. A computational study of graph partitioning. *Mathematical Programming* 66 (1994), 211–224.

[9] M.R. Garey, D.S. Johnson, and L. Stockmeyer. 1976. Some simplified NP-complete graph problems. *Theoretical Computer Science* 1, 3 (1976), 237–267.

[10] Bezalel Gavish. 1983. Formulations and Algorithms for the Capacitated Minimal Directed Tree Problem. *J. ACM* 30, 1 (Jan. 1983), 118–132.

[11] W. Hager and Y. Krulyuk. 2002. Multiset graph partitioning. *Mathematical Methods of OR* 55 (2002), 1–10. Issue 1.

[12] William W. Hager, Dzung T. Phan, and Hongchao Zhang. 2013. An exact algorithm for graph partitioning. *Mathematical Programming* 137 (2013), 531–556.

[13] Takehiro Ito, Takao Nishizeki, Michael Schröeder, Takeakin Uno, and Xiao Zhou. 2012. Partitioning a Weighted Tree into Subtrees with Weights in a Given Range. *Algorithmica* 62 (2012), 823–841.

[14] Ellis L. Johnson, Anuj Mehrotra, and George L. Nemhauser. 1993. Min-cut clustering. *Mathematical Programming* 62 (1993), 133–151.

[15] N. P. Kruyt. 1997. A conjugate gradient method for the spectral partitioning of graphs. *Parallel Comput.* 22 (1997), 1493–1502.

[16] Gilbert Laporte, Yves Nobert, and Martin Desrochers. 1985. Optimal Routing under Capacity and Distance Restrictions. *Operations Research* 33 (1985), 1050–1073. Issue 5.

[17] Isabella Lari, Federica Ricca, Justo Puerto, and Andrea Scozzari. 2016. Partitioning a Graph into Connected Components with Fixed Centers and Optimizing Cost-Based Objective Functions or Equipartition Criteria. *Networks* 67 (2016), 69–81. Issue 1.

[18] F. Murtagh. 1985. A Survey of Algorithms for Contiguity-constrained Clustering and Related Problems. *Comput. J.* 28 (1985), 82–88. Issue 1.

[19] F Murtagh. 2003. Maximum Split Clustering Under Connectivity Constraints. *Journal of Classification* 20 (2003), 143–180.

[20] Jenny Nossack and Erwin Pesch. 2014. A branch-and-bound algorithm for the acyclic partitioning problem. *Computers & Operations Research* 41 (2014), 174–184.

[21] M. W. Padberg and G. Rinaldi. 1991. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review* 33, 1 (1991), 60–100.

[22] M. W. Padberg and L. A. Wolsey. 1983. Trees and cuts. *Annals of Discrete Mathematics* 17 (1983), 511–517.

[23] A. Pothen, H. D. Simon, and K. P. Liou. 1990. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.* 11 (1990), 430âĂŞ452.

[24] F. Rendl and H. Wolkowicz. 1995. A projection tecnhique for partitioning the nodes of a graph. *Ann. Oper. Res.* 581 (1995), 172–191.

[25] Satu Elisa Schaeffer. 2007. Graph clustering. *Computer Science Review* 1, 1 (2007), 27 – 64.

[26] E. Uchoa, R. Fukasawa, J. Lysgaard, A. Pessoa, M Poggi de Aragão, and D. Andrade. 2008. Robust branch-cut-and-price for the Capacitated Minimum Spanning Tree problem over a large extended formulation. *Mathematical Programming* 112 (2008), 446–472.

[27] H. Wolkowicz and Q. Zhao. 1999. Semidefinite programming relaxations for the graph partitioning problem. *Discrete Appl. Math.* 96/97 (1999), 467–547.