A Branch-Price-and-Cut Procedure for the Discrete Ordered Median Problem

Samuel Deleplanque, Martine Labbé, Diego Ponce Lopez, Justo Puerto

▶ **To cite this version:**

HAL Id: hal-01954865

https://hal.inria.fr/hal-01954865

Submitted on 14 Dec 2018

A Branch-Price-and-Cut Procedure for the Discrete Ordered Median Problem

Samuel Deleplanque
Ifsttar, COSYS, ESTAS, Université Lille Nord de France,
Martine Labbé
Département d'Informatique, Faculté des Sciences,
Université Libre de Bruxelles,
Diego Ponce
Instituto de Matemáticas de la Universidad de Sevilla (IMUS),
Department of Mechanical, Industrial and Aerospace Engineering, Concordia University,
Centre interuniversitaire de recherche sur les réseaux d'enterprise, la logistique et le transport (CIRRELT),
Justo Puerto
Instituto de Matemáticas de la Universidad de Sevilla (IMUS).

December 13, 2018

Abstract

The Discrete Ordered Median Problem (DOMP) is formulated as a set partitioning problem using an exponential number of variables. Each variable corresponds to a set of demand points allocated to the same facility with the information of the sorting position of their corresponding costs. We develop a column generation approach to solve the continuous relaxation of this model. Then, we apply a branch-price-and-cut algorithm to solve small to large sized instances of DOMP in competitive computational time.

1 Introduction

Logistics is a very active domain in contemporary Operations Research and Location Analysis is one of its main building blocks. In the last years, motivated by the need of applying more flexible models in Logistics, a new family of location models that takes into consideration the order of the costs to be minimized has emerged in the literature.

The ordered weighted average of a vector is equal to the weighted sum of its coordinates after ranking them by non decreasing order (Nickel and Puerto (2005)). When the ordered weighted average is used in a location problem, the vector contains the distances or allocation costs from clients to service facilities. Ordered median location problems were first introduced in networks and continuous spaces by Nickel and Puerto (1999) and Puerto and Fernández (2000), respectively. Later, they were extended to the discrete setting by Nickel (2001) and Boland et al. (2006). The Discrete Ordered Median Problem (DOMP) has been widely studied since the 90's and there is a number of different formulations, solution approaches and applications available in the literature. To cite a few, DOMP has been applied to discrete facility location in Boland et al. (2006); Marín et al. (2009, 2010); Nickel (2001); Puerto (2008); Puerto et al. (2009); to location on networks in Nickel and Puerto (1999); to hub network design problems in Puerto et al. (2011, 2013, 2016); to determine values in cooperative game theory in Perea and Puerto (2013); to combinatorial optimization problems

1

with ordering in Fernández et al. (2013, 2014, 2017), and to voting problems in Ponce et al. (2018), etc. The reader is referred to the monographies by Nickel and Puerto (2005) and Puerto and Rodríguez-Chía (2015) for some other applications.

Given a set of clients, a set of candidate locations and assuming that the allocation costs of clients to facilities are known, DOMP consists in choosing p facility locations and assigning each client to a chosen facility with the smallest allocation cost in order to minimize the ordered weighted average of these costs. The ordered weighted average sorts the allocation costs in a non-decreasing sequence and then it performs the scalar product of this so-obtained sorted cost vector with a given vector of weights.

There are several valid formulations for DOMP that exploit specific features of the problem (see e.g. Boland et al. (2006); Marín et al. (2009); Labbé et al. (2017) and the references therein). In Labbé et al. (2017) a new formulation for DOMP has been proposed, based on a set packing approach, that is valid for general cost coefficients. This formulation gives rise to rather tight integrality gaps and was shown to be reasonably efficient to solve medium size instances when embedded in a branch-and-cut (**B&C**) scheme. In this paper we explore a different paradigm for solving DOMP based on an extended formulation using an exponential number of variables and that corresponds to a set partitioning model. Each variable represents a set of couples (client, position). These clients are served by the same facility and their position indicates the place of their allocation cost in the sorted list of allocation costs for the entire considered solution. To handle the exponential number of variables we use a column generation approach that is embedded in a branch-price-and-cut (**B&P&C**) algorithm. A recent similar approach to a different problem can be found in Doulabi et al. (2016). This scheme has never been applied to DOMP and it opens new avenues of research. Therefore, the contribution of this paper is to propose a new perspective in the resolution of DOMP based on formulations with an exponential number of variables and to develop an efficient **B&P&C** algorithm to handle them.

The remainder of this paper is organized as follows. Notation, models and algorithms are presented in Section 2. Subsection 2.2 introduces a new set partitioning formulation for DOMP. This formulation uses an exponential number of variables where each element of the partition is a set of clients that are assigned to the same facility together with their sorted positions. This formulation is theoretically compared with another valid formulation described in Subsection 2.1 and borrowed from Labbé et al. (2017). Subsection 2.3 describes the column generation algorithm that we have designed to overcome the large number of variables in the model. We prove that the pricing subproblem is solvable efficiently in polynomial time by using an *ad hoc* dynamic programming algorithm. We devote Section 3 to the implementation details of our **B&P&C** algorithm. We develop a GRASP heuristic, in Subsection 3.1, that is used to generate both a promising initial solution and a pool of variables to initialize the column generation routine. We also develop a stabilization routine, based on Pessoa et al. (2010), that reduces considerably the number of iterations of the column generation approach in Subsection 3.2. In addition, Subsection 3.3 is devoted to an additional improvement, namely a preprocessing. The next two subsections, 3.4 and 3.5, present our branching strategies and some families of valid inequalities that will be added to the branch-and-price algorithm. In the last section, namely Section 4, we report on the final computational experiments. We evaluate the performance of the **B&P&C** algorithm and compare it to the compact formulation in Section 2.1. The paper ends with some concluding remarks.

2 Problem definition and formulations

Let I be a set of n points which at the same time represent clients and potential uncapacitated facility locations and let c_{ij} denote the cost for serving client i's demand from facility j.

Given a set J of p open facilities, let $c_i(J)$ represent the cheapest cost for allocating client i to a facility in J, i.e. $c_i(J) := \min_{j \in J} c_{ij}$.

Now let us sort the costs $c_i(J)$, $i \in I$ by non-decreasing order of their values. The elements of the resulting vector of ordered costs are denoted by $c^{(k)}(J)$ and satisfy $c^{(1)}(J) \leq \cdots \leq c^{(n)}(J)$. We denote the set of all possible positions $1, \ldots, n$ in this ordered vector by K.

Given vector $\lambda = (\lambda^k)_{k \in K}$ satisfying $\lambda^k \geq 0, k \in K$, the objective function of DOMP, is defined as

$$z(J) := \sum_{k \in K} \lambda^k c^{(k)}(J). \tag{1}$$

Recall that this objective function provides a very general paradigm to encompass standard and new location models. For instance, if $\lambda^1 = \ldots = \lambda^n = 1$ we obtain the median objective, if $\lambda^1 = \lambda^2 = \ldots = \lambda^{n-1} = 0, \lambda^n = 1$ we obtain the center objective, if $\lambda^1 = \lambda^2 = \ldots = \lambda^{n-1} = \alpha, \lambda^n = 1$, where $\alpha = [0, 1]$, we obtain a convex combination of median and center objectives (centdian), etc.

The p-facility Discrete Ordered Median Problem looks for the subset J of p facilities to open in order to minimize the ordered median function:

$$\min_{J \subseteq I : |J| = p} z(J). \tag{DOMP}$$

Several formulations of DOMP have been proposed in the literature using different types of variables. Among them we mention those based on a combination of the p-median and permutation polytopes (Boland et al. (2006)) or on covering approaches by using radius variables (Puerto (2008), Marín et al. (2009, 2010)).

2.1 An explicit formulation for DOMP: The Weak Order Constraints

In the following, we recall the Weak Order Constraints formulation, WOC, introduced in Labbé et al. (2017), and that will be the starting point for the developments presented in this paper. This formulation uses two types of binary variables. Variable y_j assumes value 1 if facility $j \in I$ is open (i.e. $j \in J$) and 0 otherwise. Variable x_{ij}^k is equal to 1 if client $i \in I$ is allocated to facility $j \in I$ and the corresponding cost occupies position $k \in K$ in the allocation cost ranking (i.e. $c^{(k)}(J) = c_{ij}$). The choice of this formulation is motivated by its good performance in terms of integrality gap (see Labbé et al. (2017)). However, it requests important memory space since it needs $O(n^3)$ binary variables which may become prohibitive for moderate n.

We denote the rank of the allocation cost c_{ij} by r_{ij}, i.e. $r_{ij} = \ell$ if c_{ij} is the ℓ-th element in the list of the costs c_{ij}, for all $i, j \in I$, sorted by order of non decreasing values and where ties are broken arbitrarily. For the sake of readability the reader is referred to Example 1 in Section 2.3. The formulation is as follows:

$$\textbf{(WOC)} \min \quad \sum_{i \in I} \sum_{j \in I} \sum_{k \in K} \lambda^k c_{ij} x_{ij}^k \tag{2}$$

$$\text{s.t.} \quad \sum_{j \in I} \sum_{k \in K} x_{ij}^k = 1 \qquad i \in I \tag{3}$$

$$\sum_{i\in I}\sum_{j\in I} x_{ij}^k = 1 \qquad\qquad k \in K \qquad\qquad (4)$$

$$\sum_{k\in K} x_{ij}^k \le y_j \qquad\qquad i,j \in I \qquad\qquad (5)$$

$$\sum_{j\in I} y_j = p \qquad\qquad (6)$$

$$\sum_{i\in I}\sum_{j\in I}\left(\sum_{i'\in I}\sum_{\substack{j'\in I:\\ r_{i'j'}\le r_{ij}}} x_{i'j'}^k + \sum_{i'\in I}\sum_{\substack{j'\in I:\\ r_{i'j'}\ge r_{ij}}} x_{i'j'}^{k-1}\right) \le n^2 \qquad\qquad k \in K, k \ne 1 \qquad\qquad (7)$$

$$x_{ij}^k,\ y_j \in \{0,1\} \qquad\qquad i,j \in I, k \in K. \qquad\qquad (8)$$

By means of (3) we ensure that each location is served by exactly one facility. In the same way, in each position there must be exactly one allocation cost (4). Constraints (5) translate the fact that a client can be allocated to a facility only if this facility is open and that the allocation cost of a client to a facility can be placed in at most one position. The equality constraint (6) implies that there are exactly p open facilities.

Constraints (7), called *weak order constraints*, ensure that if client i is allocated to facility j and the corresponding costs c_{ij} occupies the k-th position in the cost ranking of the solution then in the $(k-1)$-th position there must be a smaller allocation cost. This property is enforced by the coefficients of each variable in the inequality. In each constraint there are two different positions, k and $k-1$, so that, by (4), only two variables must take value one and all the others will be equal to zero. If we do not take into account the variables assuming the value zero and we assume that the variables with value one for positions k and $k-1$ correspond to allocation pairs in sorted position s and t, respectively, the inequality reduces to the following expression: $(n^2 - (s-1))x_{i_s j_s}^k + t x_{i_t j_t}^{k-1} \le n^2$, which is valid if and only if $t < s$. Finally, the variables are binary, see (8).

WOC can be reinforced by adding the following valid inequalities:

$$\sum_{i'\in I}\sum_{\substack{j'\in I:\\ r_{i'j'}\le r_{ij}}} x_{i'j'}^k + \sum_{i'\in I}\sum_{\substack{j'\in I:\\ r_{i'j'}\ge r_{ij}}} x_{i'j'}^{k-1} \le 1,\ i,j \in I,\ k \in K, k \ne 1. \qquad\qquad (9)$$

Observe that constraints (7) are the aggregation over $i,j \in I$ of inequalities (9). These inequalities are the so called *strong order constraints*, See Labbé et al. (2017) for a detailed explanation.

2.2 A set partitioning formulation

From a linear programming relaxation point of view the above formulation is not the strongest one but it provides a good compromise between the number of required constraints and the quality of its linear relaxation bound, see Labbé et al. (2017). Further, it allows to solve to optimality problems of moderate size. One of its drawbacks is the use of a cubic number of variables, which can be prohibitive for large n. A second important problem of most known formulations for DOMP is their high degree of symmetry in case of allocation costs c_{ij} or weighted (λ^k) with many ties.

The reasons above motivate the introduction of a new formulation based on a different rationale. We observe that a solution for DOMP is a partition of the clients together with their positions in the sorted vector of costs so that each subset of clients in the partition is allocated to the same facility.

Let us consider sets of couples (i, k) where the first component refers to a client i and the second to a position k, namely $S = \{(i, k) : \text{ for some } i \in I, k \in K\}$. Further, we denote by $\mathcal{P}(I \times K)$ the family of all sets S for which all first (resp. second) coordinates of its couples are different.

Associated with each set S and facility j, we define a variable y_S^j equal to 1 if the set S is part of a feasible solution $((i, k) \in S \text{ iff } x_{ij}^k = 1)$; and 0 otherwise.

Let S be the set of couples whose first coordinate corresponds to the clients allocated to a given facility j of a feasible solution. The positions of these clients in the solution, i.e. the second coordinates of couples in S must be compatible with the ranking of all the allocation costs involved in the solution. Hence, they must, in particular, be compatible with the ranking of the costs c_{ij} of the clients i allocated to j. This implies that for facility $j \in J$ we only need to consider couple subsets S belonging to $\mathcal{S}(j) = \{S \in \mathcal{P}(I \times K) : c_{ij} \leq c_{i'j}$ for all $(i, k), (i', k') \in S$ and $k < k'\}$.

Since in any feasible solution, each client i must be allocated to a unique facility j and its allocation cost must occupy a unique position k in the sorted list, the following relationship holds:

$$x_{ij}^k = \sum_{S \in \mathcal{S}(j):(i,k) \in S} y_S^j, i, j \in I, k \in K. \tag{10}$$

Next, we can evaluate the cost c_S^j induced by the set S provided that its clients are assigned to facility j in a feasible solution:

$$c_S^j = \sum_{(i,k) \in S} \lambda^k c_{ij}. \tag{11}$$

To simplify the presentation in the following we denote by $(i, \cdot)$ any couple whose first entry is i regardless of the value of the second entry. Analogously, $(\cdot, k)$ denotes any couple whose second entry is k regardless of the value of the first entry.

The following valid formulation uses variables y_S^j and constitutes our Master Problem (MP).

$$\textbf{(MP)} \quad \min \quad \sum_{j \in I} \sum_{S \in \mathcal{S}(j)} c_S^j y_S^j \tag{12}$$

$$\text{s.t.} \quad \sum_{j \in I} \sum_{\substack{S \in \mathcal{S}(j): \\ (i,\cdot) \in S}} y_S^j = 1 \qquad i \in I \tag{13}$$

$$\sum_{j \in I} \sum_{\substack{S \in \mathcal{S}(j): \\ (\cdot,k) \in S}} y_S^j = 1 \qquad k \in K \tag{14}$$

$$\sum_{S \in \mathcal{S}(j)} y_S^j \leq 1 \qquad j \in I \tag{15}$$

$$\sum_{j \in I} \sum_{S \in \mathcal{S}(j)} y_S^j \leq p \tag{16}$$

$$\sum_{i=1}^n \sum_{j=1}^n \left(\sum_{\substack{S \in \mathcal{S}(j): \\ (i',k) \in S \\ r_{i'j'} \leq r_{ij}}} y_S^{j'} + \sum_{\substack{S \in \mathcal{S}(j): \\ (i',k-1) \in S \\ r_{i'j'} \geq r_{ij}}} y_S^{j'} \right) \leq n^2 \qquad k \in K, k \neq 1 \tag{17}$$

$$y_S^j \in \{0, 1\} \qquad S \in \mathcal{S}(j), j \in I. \qquad (18)$$

The objective function (12) accounts for the sorted weighted cost of any feasible solution. Constraints (13) ensure that each client appears in exactly one set S. Constraints (14) ensure that each position is taken by exactly one client appearing in one set S. Constraints (15) guarantees that each facility j serves at most one set S of clients. Inequality (16) states that at most p facilities will be opened. By the following family of inequalities (17) we enforce the correct sorting of the costs in any feasible solution. Finally, the variables are binary.

One can relate MP and WOC. First, remark that for a given facility j, there is at most one cost c_{ij} which occupies a given position k. Hence, the following constraints are valid for WOC:

$$\sum_{i \in I} x_{ij}^k \le y_j, j \in I, k \in K. \qquad (19)$$

Let $WOC+$ denote the formulation given by (2)–(8) and (19) and consider the Dantzig-Wolfe reformulation of $WOC+$ in which constraints (5), (19) and (8) constitute the subproblem. The subproblem can be decomposed by facility.

On the one hand, the feasible points of the subproblem of a facility j correspond one to one to the sets $S \in \mathcal{P}(I \times K)$. Hence, this Dantzig-Wolfe reformulation of $WOC+$ is given by the above master problem in which we consider variables y_S^j for all $S \in \mathcal{P}(I \times K)$ (instead of only $S \in \mathcal{S}(j)$). More precisely, the variables of $WOC+$ are related to the variables y_S^j through the following two equations:

$$x_{ij}^k = \sum_{S \in \mathcal{P}(I \times K):(i,k) \in S} y_S^j \quad i, j \in I, k \in K$$

and

$$y_j = \sum_{S \in \mathcal{P}(I \times K)} y_S^j \quad j \in I.$$

Moreover, constraints (13) correspond to constraints (3), constraints (14) to (4), constraints (16) to (6), and constraints (17) to (7). Finally, constraints (15) constitute the "convexity" constraints for the subproblems.

On the other hand, it is easy to see that the polyhedron of each subproblem, defined by constraints (5) and (19) together with $x_{ij}^k \ge 0$ and $y_j \le 1$, is integer. This implies that the linear relaxations of $WOC+$ and MP in which all sets $S \in \mathcal{P}(I \times K)$ are considered provide the same bound. By restricting the subsets S to be considered for each facility j to belong to $\mathcal{S}(j)$, our formulation MP provides thus a stronger model. The computational experiments presented in Subsection 4.3 show that there exist instances for which the linear relaxation of MP provides a strictly better (higher) lower bound than the linear relaxation of WOC.

Formulation MP can be strengthened by adding valid inequalities borrowed from WOC. Indeed, one can translate valid inequalities (9) in terms of the y_S^j variables so that they can be used in the set partition formulation of DOMP. The translation of (9) results in:

$$\sum_{\substack{S \in \mathcal{S}(j): \\ (i',k) \in S \\ r_{i'j'} \le r_{ij}}} y_S^{j'} + \sum_{\substack{S \in \mathcal{S}(j): \\ (i',k-1) \in S \\ r_{i'j'} \ge r_{ij}}} y_S^{j'} \le 1, \; i, j \in I, k \in K, k \ne 1. \qquad (20)$$

2.3 Column generation to solve the linear relaxation of MP (**LRMP**)

Since the number of variables in MP is too large to be handled directly, in this section, we describe a column generation approach to solve it.

Let $(\alpha, \beta, \gamma, \delta, \epsilon)$ be the dual variables associated, respectively, to constraints (13), (14), (15), (16) and (17). The dual problem DP of LRMP is

$$\textbf{(DP)} \max \qquad \sum_{i \in I} \alpha_i + \sum_{k \in K} \beta_k - \sum_{j \in I} \gamma_j - p\delta - \sum_{\substack{k \in K: \\ k \neq 1}} n^2 \epsilon_k \qquad (21)$$

$$\text{s.t.} \qquad \sum_{\substack{i \in I: \\ (i,\cdot) \in S}} \alpha_i + \sum_{\substack{k \in K: \\ (\cdot,k) \in S}} \beta_k - \gamma_j - \delta - \sum_{i' \in I} \sum_{j' \in I} \left(\sum_{\substack{(i,k) \in S: \\ r_{i'j'} \geq r_{ij} \\ k \neq 1}} \epsilon_k + \sum_{\substack{(i,k) \in S: \\ r_{i'j'} \leq r_{ij} \\ k \neq n}} \epsilon_{k+1} \right) \leq c_S^j \qquad j \in I, S \in \mathcal{S}(j) \qquad (22)$$

$$\delta, \gamma_j, \epsilon_k \geq 0 \qquad j \in I, k \in K, k \neq 1. \qquad (23)$$

In order to apply the column generation procedure, let us assume that we are given a set of columns that define a restricted Master Problem, and denote its linear relaxation by ReLRMP. This problem is solved to optimality and $(\alpha^*, \beta^*, \gamma^*, \delta^*, \epsilon^*)$ represents its optimal dual solution. See Example 1. The reduced cost, $\bar{c}_S^j$, of column y_S^j, namely $\bar{c}_S^j = c_S^j - z_S^j$ is given by:

$$\bar{c}_S^j = c_S^j + \gamma_j^* + \delta^* + \sum_{i' \in I} \sum_{j' \in I} \left(\sum_{\substack{(i,k) \in S: \\ r_{i'j'} \geq r_{ij} \\ k \neq 1}} \epsilon_k^* + \sum_{\substack{(i,k) \in S: \\ r_{i'j'} \leq r_{ij} \\ k \neq n}} \epsilon_{k+1}^* \right) - \sum_{\substack{i \in I: \\ (i,\cdot) \in S}} \alpha_i^* - \sum_{\substack{k \in K: \\ (\cdot,k) \in S}} \beta_k^*. \qquad (24)$$

If $\bar{c}_S^j \geq 0$ for all $j, S \in \mathcal{S}(j)$ the current solution of ReLRMP is also optimal for the LRMP and the column generation procedure stops.

Otherwise, one has identified one (some) new column(s) to be added to the current reduced master problem to proceed further. In each iteration, ReLRMP and its reduced costs provide lower and upper bounds for the LRMP. Indeed it holds that (Desrosiers and Lübecke (2005))

$$z_{ReLRMP} + p \cdot \min_{j \in I, S \in \mathcal{S}(j)} \bar{c}_S^j \leq z_{LRMP} \leq z_{ReLRMP}, \qquad (25)$$

$$z_{ReLRMP} + \sum_{j \in I} \min_{S \in \mathcal{S}(j)} \bar{c}_S^j \leq z_{LRMP} \leq z_{ReLRMP}, \qquad (26)$$

where z_{ReLRMP} and z_{LRMP} denote the optimal value of $ReLRMP$ and $LRMP$ respectively.

Example 1. *Consider the following vector* $\lambda = (4, 2, 1)$, *cost matrix* C *and precedence matrix* R:

$$C = \begin{pmatrix} 1 & 3 & 6 \\ 3 & 1 & 8 \\ 6 & 8 & 1 \end{pmatrix}, \qquad R = \begin{pmatrix} 1 & 4 & 6 \\ 5 & 2 & 8 \\ 7 & 9 & 3 \end{pmatrix}.$$

For $n = 3$, there are 33 different sets of couples (i, k) in $\mathcal{S}$.

$$S_1 = \{(1,1)\} \qquad S_{10} = \{(1,1),(2,2)\} \qquad S_{19} = \{(1,3),(2,2)\}$$
$$S_2 = \{(1,2)\} \qquad S_{11} = \{(1,1),(2,3)\} \qquad S_{20} = \{(1,3),(3,1)\}$$
$$S_3 = \{(1,3)\} \qquad S_{12} = \{(1,1),(3,2)\} \qquad S_{21} = \{(1,3),(3,2)\}$$
$$S_4 = \{(2,1)\} \qquad S_{13} = \{(1,1),(3,3)\} \qquad S_{22} = \{(2,1),(3,2)\}$$
$$S_5 = \{(2,2)\} \qquad S_{14} = \{(1,2),(2,1)\} \qquad S_{23} = \{(2,1),(3,3)\}$$
$$S_6 = \{(2,3)\} \qquad S_{15} = \{(1,2),(2,3)\} \qquad S_{24} = \{(2,2),(3,1)\}$$
$$S_7 = \{(3,1)\} \qquad S_{16} = \{(1,2),(3,1)\} \qquad S_{25} = \{(2,2),(3,3)\}$$
$$S_8 = \{(3,2)\} \qquad S_{17} = \{(1,2),(3,3)\} \qquad S_{26} = \{(2,3),(3,1)\}$$
$$S_9 = \{(3,3)\} \qquad S_{18} = \{(1,3),(2,1)\} \qquad S_{27} = \{(2,3),(3,2)\}$$

$$S_{28} = \{(1,1),(2,2),(3,3)\}$$
$$S_{29} = \{(1,1),(2,3),(3,2)\}$$
$$S_{30} = \{(1,2),(2,1),(3,3)\}$$
$$S_{31} = \{(1,2),(2,3),(3,1)\}$$
$$S_{32} = \{(1,3),(2,1),(3,2)\}$$
$$S_{33} = \{(1,3),(2,1),(3,2)\}.$$

The sets $\mathcal{S}(j)$ are the following:

$$\mathcal{S}(1) = \{S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10}, S_{11}, S_{12}, S_{13}, S_{15}, S_{17}, S_{22}, S_{23}, S_{25}, S_{28}\},$$
$$\mathcal{S}(2) = \{S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{12}, S_{13}, S_{14}, S_{17}, S_{18}, S_{19}, S_{22}, S_{23}, S_{25}, S_{30}\},$$
$$\mathcal{S}(3) = \{S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10}, S_{11}, S_{15}, S_{16}, S_{20}, S_{21}, S_{24}, S_{26}, S_{27}, S_{31}\}.$$

We consider as initial pool of columns the variables y_{18}^1 and y_8^3. With this set of variables, the ReLRMP is

$$
\begin{array}{rllll}
\textbf{(ReLRMP)}\min & +2y_5^2 & +10y_{13}^1 & & \\
\text{s.t.} & & +y_{13}^1 & \geq & 1 & i = 1 \\
& +y_5^2 & & \geq & 1 & i = 2 \\
& & +y_{13}^1 & \geq & 1 & i = 3 \\
& & +y_{13}^1 & \geq & 1 & k = 1 \\
& +y_5^2 & & \geq & 1 & k = 2 \\
& & +y_{13}^1 & \geq & 1 & k = 3 \\
& & -y_{13}^1 & \geq & -1 & j = 1 \\
& -y_5^2 & & \geq & -1 & j = 2 \\
& & & \geq & -1 & j = 3 \\
& -y_5^2 & -y_{13}^1 & \geq & -2 & \\
& -8y_5^2 & -y_{13}^1 & \geq & -9 & k = 2 \\
& -2y_5^2 & -3y_{13}^1 & \geq & -9 & k = 3 \\
& & y & \geq & 0 & .
\end{array}
$$

Actually, we are interested in its dual problem:

$$
\begin{array}{rccccccccccccccc}
\textbf{(DP)}\max & +\alpha_1 & +\alpha_2 & +\alpha_3 & +\beta_1 & +\beta_2 & +\beta_3 & -\gamma_1 & -\gamma_2 & -\gamma_3 & -2\delta & -9\epsilon_2 & -9\epsilon_3 & & & \\
\text{s.t.} & & +\alpha_2 & & & +\beta_2 & & & -\gamma_2 & & -\delta & -8\epsilon_2 & -2\epsilon_3 & \leq & 2 & (y_5^2) \\
& +\alpha_1 & & +\alpha_3 & +\beta_1 & & +\beta_3 & -\gamma_1 & & & -\delta & -\epsilon_2 & -3\epsilon_3 & \leq & 10 & (y_{13}^1) \\
& & & & & & & & & & \alpha, \beta, & \gamma, \delta, \epsilon & \geq & 0 & .
\end{array}
$$

Solving (DP), the solution is $\alpha_2 = 2$, $\beta_3 = 10$, $\alpha_1 = \alpha_3 = \beta_1 = \beta_2 = \delta = \epsilon_2 = \epsilon_3 = 0$ and the value of the objective function is $f = 12$.

2.4   Solving the pricing subproblem

Although any column y_S^j with negative reduced cost may be added to ReLRMP, we will follow a strategy that identifies the most negative reduced cost for each facility j. This approach

may give rise to several candidate columns (multiple pricing, see Chvátal (1983)), which is advantageous for this procedure.

In order to do that, for each facility $j \in I$, we solve a subproblem to find the column y_S^j, $S \in \mathcal{S}(j)$, with minimum reduced cost. This set S must be such that there is at most one couple $(i, \cdot)$ for each client i and one couple$(\cdot, k)$ for each position k. Furthermore, the set S must enjoy that the allocation costs of its couples are compatible. We solve this problem by the following dynamic programming algorithm.

Let d_{ij}^k be the contribution of the pair (i, k) to the reduced cost of any column y_S^j such that $(i, k) \in S$. Depending on the values of k, d_{ij}^k is given by

$$
d_{ij}^k = \begin{cases}
\lambda^k c_{ij} + \displaystyle\sum_{i' \in I} \sum_{\substack{j' \in I: \\ r_{i'j'} \leq r_{ij}}} \epsilon_{k+1} - \alpha_i - \beta_k & \text{if } k = 1, \\[2em]
\lambda^k c_{ij} + \displaystyle\sum_{i' \in I} \sum_{\substack{j' \in I: \\ r_{i'j'} \geq r_{ij}}}^{n} \epsilon_k + \sum_{i' \in I} \sum_{\substack{j' \in I: \\ r_{i'j'} \leq r_{ij}}} \epsilon_{k+1} - \alpha_i - \beta_k & \text{if } k = 2, \ldots, n-1, \\[2em]
\lambda^k c_{ij} + \displaystyle\sum_{i' \in I} \sum_{\substack{j' \in I: \\ r_{i'j'} \geq r_{ij}}} \epsilon_k - \alpha_i - \beta_k, & \text{if } k = n.
\end{cases} \tag{27}
$$

Then, for a facility j, the problem of finding the variable y_S^k with minimum reduced costs can be formulated as:

$$
\min_{S \in \mathcal{S}(j)} \bar{c}_S^j = \gamma_j^* + \delta^* + \sum_{(i,k) \in S} d_{ij}^k. \tag{28}
$$

Now for each facility j, we define a matrix D_j as follows:

$$
D_j = \begin{pmatrix}
d_{i_1 j}^1 & d_{i_1 j}^2 & \cdots & d_{i_1 j}^n \\
d_{i_2 j}^1 & & & \\
\vdots & & \ddots & \\
d_{i_n j}^1 & & & d_{i_n j}^n
\end{pmatrix} \tag{29}
$$

where $i_1, i_2, \ldots, i_n$ is a permutation of the indices $i = 1, \ldots, n$ such that $c_{i_1 j} \leq c_{i_2 j} \leq \cdots \leq c_{i_n j}$.

Example 2. *(Continuing from Example 1). We illustrate the procedure that computes the elements d_{ij}^k for all $i, k = 1, \ldots, n$ of the matrix D_1.* **(j=1)**

$d_{11}^1 = \lambda^1 c_{11} + r_{11}\epsilon_2 - \alpha_1 - \beta_1 = 4$ 　　　　 $d_{11}^2 = \lambda^2 c_{11} + (n^2 - r_{11} + 1)\epsilon_2 + r_{11}\epsilon_3 - \alpha_1 - \beta_2 = 2$

$d_{11}^3 = \lambda^3 c_{11} + +(n^2 - r_{11} + 1)\epsilon_3 - \alpha_1 - \beta_3 = -9$ 　　 $d_{21}^1 = \lambda^1 c_{21} + r_{21}\epsilon_2 - \alpha_2 - \beta_1 = 10$

$d_{21}^2 = \lambda^2 c_{21} + (n^2 - r_{21} + 1)\epsilon_2 + r_{21}\epsilon_3 - \alpha_2 - \beta_2 = 4$ 　 $d_{21}^3 = \lambda^3 c_{21} + +(n^2 - r_{21} + 1)\epsilon_3 - \alpha_2 - \beta_3 = -9$

$d_{31}^1 = \lambda^1 c_{31} + r_{31}\epsilon_2 - \alpha_3 - \beta_1 = 24$ 　　　　 $d_{31}^2 = \lambda^2 c_{31} + (n^2 - r_{31} + 1)\epsilon_2 + r_{21}\epsilon_3 - \alpha_3 - \beta_2 = 12$

$d_{31}^3 = \lambda^3 c_{31} + +(n^2 - r_{31} + 1)\epsilon_3 - \alpha_3 - \beta_3 = -4$

Since $r_{11} < r_{21} < r_{31}$ the valid permutation is $(1, 2, 3)$. This implies that

$$
D_1 = \begin{pmatrix}
4 & 2 & -9 \\
10 & 4 & -9 \\
24 & 12 & -4
\end{pmatrix} \quad \begin{matrix} i = 1 \\ i = 2 \\ i = 3 \end{matrix}
$$

By using D_j we obtain that a set S belongs to $\mathcal{S}(j)$ if and only if for every (i_1, k_1) and $(i_2, k_2) \in S$ such that $i_1 < i_2 : k_1 < k_2$.

Our dynamic programming algorithm to obtain the minimum reduced cost for each $j \in J$ builds upon this observation by constructing a solution to a reduced version of (28) in which only the first i_l clients and the first k positions are considered.

For each couple (i_l, k), we define a function:

$$g^j(i_l, k) = \min\{\bar{c}_S^j : S \in \mathcal{S}(j) \text{ and for all } (i_l', k') \in S : i_l' \leq i_l \text{ and } k' \leq k\} \qquad (30)$$

and we denote an optimal solution of this restricted optimization problem by $S^j(i_l, k)$.

Hence, the optimal value of problem (28) is equal to $g^j(i_n, n) + \delta + \gamma_j$ and a corresponding optimal solution by $S^j(i_n, n)$.

Our recursive procedure computes $g^j(i_l, k)$ and $S^j(i_l, k)$ for increasing values of l and k and exploits the following feasibility conditions on S:

(i) for each client i (resp. position k), at most one couple containing i (resp. position k) belongs to S.

(ii) if (i_{l_1}, k_1) and $(i_{l_2}, k_2) \in S$ and $k_1 < k_2$ then $r_{i_{l_1} j} < r_{i_{l_2} j}$.

More precisely, if (i_l, k) belongs to $S^j(i_l, k)$ then, from (i) it follows that $g^j(i_l, k) = g^j(i_{l-1}, k-1) + d_{i_l j}^k$. Otherwise, $S^j(i_l, k)$ may contain a couple (i_l, k') with $k' \leq k-1$ or a couple $(i_{l'}, k)$ with $l' \leq l-1$ but not both for otherwise condition (ii) would be violated. Hence, in this case, $g^j(i_l, k) = \min\{g^j(i_{l-1}, k-1), g^j(i_l, k-1), g^j(i_{l-1}, k)\}$. Combining the two cases, we obtain the following recurrence relation for $l, k = 2, \ldots, n$:

$$g^j(i_l, k) = \min\{g^j(i_{l-1}, k-1) + d_{i_l j}^k, g^j(i_{l-1}, k-1), g^j(i_l, k-1), g^j(i_{l-1}, k)\}. \qquad (31)$$

Obviously, if at the end of the procedure, $g^j(i_n, n) + \delta + \gamma_j$ is negative the variable $y_{S^j(i_n, n)}^j$ is a good candidate to be chosen in the next iteration of the column generation scheme.

If we solve this problem for all j, we get $\bar{c}_R^j = \min_S \bar{c}_S^j$ and if $\bar{c}_R^j < 0$, we can activate (at least) y_R^j. Next, we solve a new reduced master problem ReLRMP with this (these) new activated variable(s).

Remark 1. *Computing each matrix D_j can be done in $O(n^2)$. Next, obtaining $g^j(i_n, n)$ requires the evaluation of the function $g^j(i, k)$ for all $i \in I$ and $k \in K$. According to the above algorithm, the evaluation of each $g^j(i, k)$ is done in constant time. Solving the pricing subproblem amounts to evaluate $g^j(i_n, n)$ for all $j \in I$. Therefore, the entire pricing subproblem can be solved in $O(n^3)$ time.*

Example 3. *(Continuing from Example 2). We show the computation of the $g^j(i_n, n)$ and $S^j(i_n, n)$ for $j = 1$.*

$g^1(i_1, 1) = \min\{0, 4\} = 0, S^1(i_1, 1) = \emptyset$ $\qquad g^1(i_1, 2) = \min\{2, 0\} = 0, S^1(i_1, 2) = \emptyset$
$g^1(i_1, 3) = \min\{-9, 0\} = -9, S^1(i_1, 3) = \{(1, 3)\}$ $\qquad g^1(i_2, 1) = \min\{10, 0\} = 0, S^1(i_2, 1) = \emptyset$
$g^1(i_3, 1) = \min\{24, 0\} = 0, S^1(i_3, 1) = \emptyset$ $\qquad g^1(i_2, 2) = \min\{0 + 4, 0, 0, 0\}, S^1(i_2, 2) = \emptyset$
$g^1(i_3, 2) = \min\{0 + 12, 0, 0, 0\}, S^1(i_3, 2) = \emptyset$ $\qquad g^1(i_2, 3) = \min\{0 - 9, 0, -9, 0\}, S^1(i_2, 3) = \{(1, 3)\}$
$g^1(i_3, 3) = \min\{0 - 4, 0, -9, 0\}, S^1(i_3, 3) = \{(1, 3)\}$

We have obtained $g^1(i_3, 3) = -9$ and $S^1(i_3, 3) = S_3$ is the potential set to be used, since its reduced cost is negative. The corresponding reduced cost $\bar{c}_3^1 = g^1(i_3, 3) + \delta + \gamma_1 = -9 + 0 + 0 = -9 < 0$. Hence, we active variable y_3^1.

Algorithm 1 Pricing Subproblem Algorithm.

1: $g^j(i_1, 1) = \min\{0, d^1_{i_1 j}\}$;
2: **if** $g^j(i_1, 1) = d^1_{i_1 j} < 0$ **then**
3:     $S^j(i_1, 1) = \{(i_1, 1)\}$;
4: **else**
5:     $S^j(i_1, 1) = \emptyset$;
6: **end if**
7: **for** $k = 2, \ldots, n$ **do**
8:     $g^j(i_1, k) = \min\{d^k_{i_1 j}, g^j(i_1, k-1)\}$;
9:     **if** $g^j(i_1, k) = g^j(i_1, k-1)$ **then**
10:         $S^j(i_1, k) = S^j(i_1, k-1)$;
11:     **else**
12:         $S^j(i_1, k) = \{(i_1, k)\}$;
13:     **end if**
14: **end for**
15: **for** $l = 2, \ldots, n$ **do**
16:     $g^j(i_l, 1) = \min\{d^1_{i_l j}, g^j(i_{l-1}, 1)\}$;
17:     **if** $g^j(i_l, 1) = g^j(i_{l-1}, 1)$ **then**
18:         $S^j(i_l, 1) = S^j(i_{l-1}, k)$;
19:     **else**
20:         $S^j(i_l, 1) = \{(i_l, 1)\}$;
21:     **end if**
22: **end for**
23: **for** $k, l = 2, \ldots, n$ **do**
24:     $g^j(i_l, k) = \min\{g^j(i_{l-1}, k-1) + d^k_{i_l j}, g^j(i_{l-1}, k-1), g^j(i_l, k-1), g^j(i_{l-1}, k)\}$;
25:     **if** $g^j(i_l, k) = g^j(i_{l-1}, k-1)$ **then**
26:         $S^j(i_l, k) = S^j(i_{l-1}, k-1)$;
27:     **else if** $g^j(i_l, k) = g^j(i_l, k-1)$ **then**
28:         $S^j(i_l, k) = S^j(i_l, k-1)$;
29:     **else if** $g^j(i_l, k) = g^j(i_{l-1}, k)$ **then**
30:         $S^j(i_l, k) = S^j(i_{l-1}, k)$;
31:     **else**
32:         $S^j(i_l, k) = S^j(i_{l-1}, k-1) \cup \{(i_l, k)\}$;
33:     **end if**
34: **end for**

Next, the process continues with the following facilities, i.e. $j = 2, 3$. In this example the optimal solution can be certified after four complete iterations of the above process.

2.5   Dealing with infeasibility

One important issue when implementing a column generation procedure to solve a linear optimization problem is how to deal with infeasibility. This is specially crucial if the procedure is used within a branch-and-bound scheme to solve the linear relaxation of the problem at every node of the branching tree. To handle this, we resort to the so called Farkas pricing. This method was used previously, to the best of our knowledge, in Günlük et al. (2005) and Ceselli et al. (2008). The term Farkas pricing was coined in Gamrath (2010).

According to Farkas' Lemma (Farkas (1894)), a reduced master problem is infeasible if and only if its associated dual problem is unbounded. Thus, to recover feasibility in the ReLRMP we have to revoke the certificate of unboundedness in the dual problem. This can be done by adding constraints to it. Since we are only interested in recovering feasibility in ReLRMP, one can proceed in the same way that for the usual pricing, but with null coefficients in the objective function of the primal. In this way, the Farkas dual problem is

$$\max \qquad \sum_{i \in I} \alpha_i + \sum_{k \in K} \beta_k - \sum_{j \in I} \gamma_j - p\delta - \sum_{\substack{k \in K: \\ k \neq 1}} n^2 \epsilon_k \qquad (32)$$

$$s.t. \quad \sum_{\substack{i \in I: \\ (i,\cdot) \in S}} \alpha_i + \sum_{\substack{k \in K: \\ (\cdot,k) \in S}}^{n} \beta_k - \gamma_j - \delta - \sum_{i' \in I} \sum_{j' \in I} \left(\sum_{\substack{(i,k) \in S: \\ r_{i'j'} \geq r_{ij} \\ k \neq 1}} \epsilon_k + \sum_{\substack{(i,k) \in S: \\ r_{i'j'} \leq r_{ij} \\ k \neq n}} \epsilon_{k+1} \right) \leq 0 \quad j \in I, S \in \mathcal{S}(j) \qquad (33)$$

$$\delta, \gamma_j \epsilon_k \geq 0 \quad j \in I, k \in K, k \neq 1. \quad (34)$$

To identify new variables that make the reduced master problem feasible we use our dynamic programming approach in which we replace the column costs c_S^j by zeros.

Farkas pricing is an important element in our approach because it allows to start the column generation algorithm with an empty pool of columns, although this is not advisable. Furthermore, Farkas pricing will be crucial in the branching phase to recover feasibility (whenever possible) in those nodes of the branching tree where it is lost after fixing variables.

3   A branch-price-and-cut implementation

In this section, we precise several components of the implementation of our set partitioning formulation based on a column generation approach. **B&P&C** is a branch-and-cut scheme that solves the linear relaxation at each node of the branching tree with the column generation algorithm previously described and may apply cuts to improve the obtained lower bound. (The reader is referred to Doulabi et al. (2016) for another recent implementation of a **B&P&C**.)

Unless otherwise specified, to calibrate the best choice of the different parameters used in our **B&P&C**, we have performed a preliminary computational study based on a set of 60 instances with sizes $n = 20, 30$ and with a time limit of 1800 sec. Those are the smallest instances that we will eventually use in Section 4.

3.1   Upper bound for the Master Problem: A GRASP heuristic and an initialization stage

A heuristic algorithm that generates a good feasible solution for MP will provide a promising pool of initial columns as well as a good upper bound.

GRASP (Feo and Resende (1989), Feo and Resende (1995)) is a well-known heuristic technique that usually exhibits good performance in short computing time. In our case, it consists in a multistart greedy algorithm to construct a set of p facilities from a randomly generated set of facilities with smaller cardinality. Following Puerto et al. (2014) we have chosen, in a greedy manner, an initial set of $\lfloor p/2 \rfloor$ facilities. Next, we improve this initial solution by performing a fixed number of iterations of a local search procedure.

The greedy algorithm adds iteratively a new facility to the current set of open facilities, choosing the one with the maximum improvement of the objective value. The local search consists in an interchange heuristic between open and closed facilities. The pseudocode of the GRASP used to solve the problem is described in Algorithm 1.

Algorithm 2 GRASP for DOMP.

1: Input($n, p, C, \lambda, n_1, n_2, q$);
2: **for** n_1 replications **do**
3:     PartialSolution $\leftarrow$ ConstructRandomizedPartialSolution(q);
4:     Solution $\leftarrow$ ConstructGreedySolution(PartialSolution);
5:     **for** n_2 iterations **do**
6:         Solution $\leftarrow$ LocalSearch(Solution);
7:         BestSolution $\leftarrow$ UpdateSolution(Solution, BestSolution);
8:     **end for**
9: **end for**

First of all, we would like to point out the remarkable behavior of the GRASP heuristic for this problem. In order to illustrate the appropriateness of our heuristic we have solved to optimality a number of instances of the problem (using the MIP formulation) to be compared with those given by our GRASP. In all instances, up to a size of $n = 400$, the solution provided by GRASP is always as good as the one obtained by the any of our MIP formulations with a CPU time limit of 7,200 seconds, see Section 4.

Moreover, it is not only advisable to use the GRASP heuristic because it provides a very good upper bound thus helping the exploration of the searching tree by pruning many branches of the branch-and-bound tree, but in addition, the construction phase of the heuristic also provides a very promising pool of initial columns for the **B&P&C**, in combination with the technique described in the following.

Since we are solving the linear relaxation of our master problem, LRMP, without generating its entire set of variables, using the primal simplex algorithm, the goal of the initialization phase is to find an initial set of columns that allows solving the MP by performing a small number of iterations in the column generation routine. We create variables using a modification of the local search routine of the GRASP algorithm. Every time that we find a promising feasible solution in the heuristic, we create the variables that define that solution (CreateSetVariables(J)). Algorithm 3 presents the pseudocode of this process.

Function CreateSetVariables(J) determines the costs involved in the solution, i.e. the minimum for each client among the open facilities. Then those costs are ordered to determine the position of each client. Once we know the couples (i, k) assigned for each open facility, the corresponding variables are added to the pool.

Example 4. *(Continuing from Example 1) We illustrate the use of the function CreateSetVariables(J) with the following set $J = \{1, 3\}$ (open facilities). The allocation costs for this set J of open facilities are $c_{11} = 1, c_{21} = 3, c_{33} = 1$. According to R, the ranks of these costs are $r_{11} = 1 < r_{33} = 3 < r_{21} = 5$. Thus, we get the couples $(1, 1), (3, 2)$ and $(2, 3)$. This means that client 1 goes to facility 1 in position 1, client 3 goes to facility 3 in position 2 and client 2 goes to facility 1 in position 3. Therefore, the variables $y^1_{\{(1,1),(2,3)\}}$ and $y^3_{\{(3,2)\}}$ are added to the pool.*

Algorithm 3 Initial columns.

1: Input($|J| = p$);
2: $\bar{z} = z(J)$; CreateSetVariables(J);
3: **for** n_2 iterations,$j_1 \in J, j_2 \in \bar{J}$ **do**
4:     **if** $z((J \setminus \{j_1\}) \cup \{j_2\}) < \bar{z}$ **then**
5:         $\bar{z} = z((J \setminus \{j_1\}) \cup \{j_2\})$; $J = (J \setminus \{j_1\}) \cup \{j_2\}$; CreateSetVariables(J);
6:     **end if**
7: **end for**

In order to test the usefulness of GRASP in solving problem instances, Table 1 reports

results for the 60 instances of sizes $n = 20, 30$ enabling or not the use of the GRASP. It shows average results of CPU time (Time(s)), percent gap at termination, i.e. $100(z_{UB} - z_{LB})/z_{LB}$ (GAP(%)), and number of unsolved problems (in parentheses), number of nodes (#nodes) and number of variables ($|Vars|$).

GRASP	Time(s)	GAP(%)	#nodes	$|Vars|$
Disabled	1107.21	– (36)	158	12850
Enabled	965.31	0.89(30)	88	9907

Table 1: CPU-Time, Number of nodes and Number of variables with and without GRASP heuristic for $n = 20, 30$.

GRASP	Time(s)	#nodes	$|Vars|$
Disabled	386.19	272	10962
Enabled	147.77	79	6062

Table 2: CPU-Time, Number of nodes and Number of variables with and without GRASP heuristic for $n = 20, 30$. Summary of solved instances.

According to Table 1 it is clearly advisable to use the upper bound provided by the GRASP heuristic: it reduces the number of nodes, thus improving the size of the branch-and-bound tree.

In Table 2, the same information as in Table 1 is reported but only for the instances solved to optimality within the time limit. One can observe from this table that enabling the use of GRASP reduces the CPU time and number of nodes of the **B&B** tree and at the same time reduces the overall number of variables required by the **B&P&C**. In addition, by using the GRASP heuristic, **B&P&C** is able to solve 6 more instances. For those instances for which **B&P&C** does not certify optimality, GRASP provides an upper bound that leads to an average gap of 0.89 %. Finally, without the use of GRASP, in many cases, no feasible solutions are found within the time limit and thus no % gap ("–") can be reported.

Our results show that by using the GRASP heuristic, 2.03% of the final number of variables are generated when applying Algorithm 3. The combination of the incumbent solution (given by GRASP) and that initial pool of variables leads to solve the considered instances faster, requiring less number of nodes and variables to certify optimality.

Figure 1 reports the performance profile of GAP versus number of solved instances within a time limit of 1800 seconds, for the 60 instances with sizes $n = 20, 30$. The blue line reports results using GRASP and the orange one without it. It is interesting to point out that when GRASP is enabled the **B&P&C** is able to solve to optimality 30 instances and the GAP of the remaining never goes beyond 6.43%. On the other hand, if GRASP is disabled then **B&P&C** solves only 24 instances. In addition, it is capable to obtain a feasible solution for only 2 more instances whereas in the remaining 34 instances the gap is greater than 100% (no feasible solution is found).

3.2   Stabilization

When using a column generation procedure, the vector of dual variables may be quite different from an iteration to the next resulting in a slow convergence. For this reason, the stabilization is sometimes a critical step in order to reduce the number of variables and iterations needed to solve each reduced master problem (du Merle et al. (1999)). We follow the stabilization procedure of Pessoa et al. (2010) which depends on only one parameter. The idea consists in using a vector of dual variables which is a convex combination of the previous vector and the current solution of the dual problem.

Let $\pi = (\alpha, \beta, \gamma, \delta, \epsilon)$ be a generic vector of dual multipliers, $\overline{\pi}$ be the best known vector of dual multipliers (found so far) and π_{ReMP} be the current solution of the dual problem. Let $\overline{c}_S^j(\pi)$ be the reduced cost of y_S^j computed with the dual variable π and $LB(\pi)$ the lower
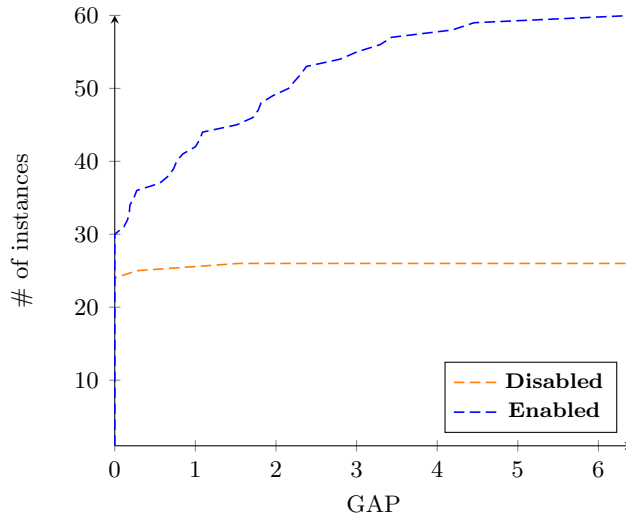
Figure 1: Performance profile graph with GRASP enabled or disabled after 1800 seconds, GAP / # of instances.

bound provided by the same vector of dual multipliers, namely π. Finally, let $z_D(\pi)$ be the value of the dual objective function of ReLRMP for the dual vector π, see (25). The stabilization algorithm that we have implemented is described by the following pseudocode:

Algorithm 4 Stabilization in $ReLRMP$.

1: $\Delta = \Delta_{init}$; $\overline{\pi} = 0$; $LB(\overline{\pi}) = 0$; $GAP = 1$;
2: **while** $GAP > \epsilon$ **do**
3:     Solve ReLRMP, obtaining z_{ReLRMP} and π_{ReLRMP}; $\pi_{st} = \Delta\pi_{ReLRMP} + (1 - \Delta)\overline{\pi}$;
4:     **for** $j = 1, \ldots, n$ **do**
5:         Solve the pricing using π_{st}, obtaining S;
6:         **if** $\overline{c}_S^j(\pi_{ReLRMP}) < 0$ **then** Add variable y_S^j; **end if**
7:     **end for**
8:     $LB(\pi_{st}) = z(\pi_{st}^t) + \displaystyle\sum_{S,j:y_S^j\,added} \overline{c}_S^j(\pi_{st})$;
9:     **if** At least one variable was added **then**
10:         **if** $LB(\pi_{st}) > LB(\overline{\pi})$ **then**
11:             $\overline{\pi} = \pi_{st}$; $LB(\overline{\pi}) = LB(\pi_{st})$;
12:         **end if**
13:     **else**
14:         $\overline{\pi} = \pi_{st}$; $LB(\overline{\pi}) = LB(\pi_{st})$;
15:     **end if**
16:     $GAP = \frac{z_{ReLRMP}-LB(\overline{\pi})}{z_{ReLRMP}}$;
17:     **if** $GAP < 1 - \Delta$ **then** $\Delta = 1 - GAP$; **end if**
18: **end while**

In words, the algorithm performs a while loop where in each iteration it makes a convex combination of the current vector of dual multipliers and the best vector of multipliers found so far. This loop ends whenever both vectors of multipliers are close enough based on the gap between the incumbent lower bound and the actual value of the reduced master problem. It is important to realize that the coefficient (importance), Δ, given in the convex combination to π_{ReLRMP} (the current solution of ReLRMP) increases with the number of iterations of the algorithm since $\Delta = 1 - GAP$ and GAP decreases with the number of iterations. Eventually in the very last iterations of the stabilization algorithm we will use the actual vector of dual multipliers since $\pi_{st} \approx \pi_{ReLRMP}$.

In order to check the efficiency of the stabilization and to determine the best value for parameter Δ_{init}, LRMP has been solved to optimality for 270 instances from $n = 20$ to $n = 100$. In our implementation, we have chosen $\Delta = 0.4$ based on the computational study shown in Figure 2. As one can observe in this figure, the best performance profile is obtained by $\Delta = 0.4$ (blue dashed line) because it is the configuration that solves the instances in less time. It is worth mentioning that LRMP can be solved in one third of the time required for solving the problem without stabilization. Detailed results of the linear relaxation for $\Delta = 0.4$ are reported in Table 4.
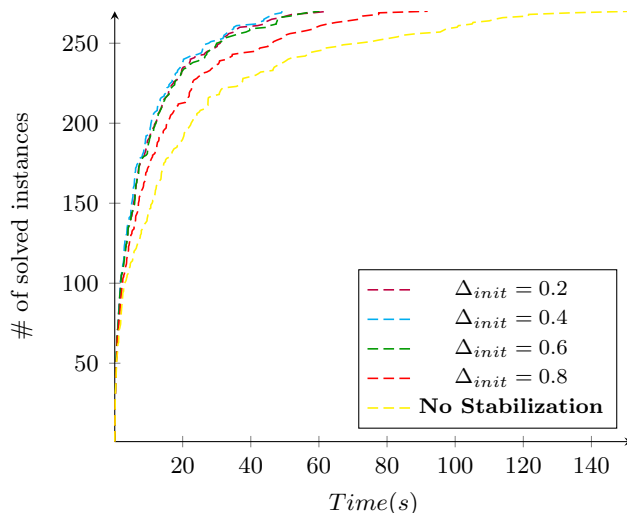
Figure 2: Performance profile graph with different combination of Δ_{init}, #solved instances / n.

We report in Figure 3 the evolution of the lower and upper bounds with respect to number of iterations for a single instance. Stabilization results in a better behavior: the dual bound is not infinite at iteration 0 and it does not improve for some iterations. The reason is because we start with a feasible solution of the problem.

The control over the dual variables significantly improves the necessary number of iterations and the number of variables used to certify optimality. Note that this improvement becomes more important when MP is solved using a branch-and-bound procedure because
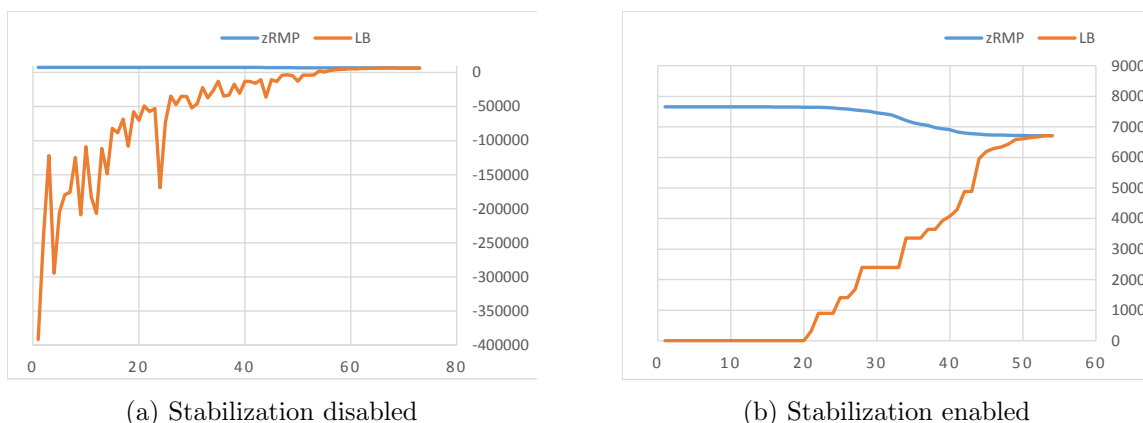
(a) Stabilization disabled

(b) Stabilization enabled

Figure 3: Bound's behavior at the root node in a particular instance on successive iterations.

the number of variables should be small at every node.

3.3 Preprocessing

In order to improve the performance of the algorithm we use two different preprocessings to set some variables to zero. Our approach is based on Claims 1 and 2 in Labbé et al. (2017). The reader may observe that although those results allow to fix some x_{ij}^k variables to zero, this variable-fixing can be translated to the new setting by using the relation (10) between the variables in formulations WOC and MP.

Therefore, the above results imply that those variables y_S^j such that $(i,k) \in S$ and $x_{ij}^k = 0$ will not be considered to be added to the ReLRMP. This can be simply enforced by setting the corresponding $d_{ij}^k = 0$ in every pricing subproblem.

3.4 Branching strategies

Branching on the original variables is a common option when the master problem involves set partition constraints. See for instance Johnson (1989). In spite of that, we have also considered other branching strategies as using the set partitioning variables or the Ryan and Foster branching, Ryan and Foster (1981); Barnhart et al. (1998). However, these two alternatives were discarded because our pricing subproblem is polynomially solvable when we branch on the original variables whereas using any of the other branching strategies mentioned above, makes it $\mathcal{NP}$-hard.

Recall that $x_{ij}^k = \sum_{S \in \mathcal{S}(j):(i,k) \in S} y_S^j$, thus, a way to branch using a fractional solution can be derived directly from the integrality conditions on the original variables.

Proposition 1. *If $x_{ij}^k \in \{0,1\}$ for $i,j \in I, k \in K$, then $y_S^j \in \{0,1\}$ for $j \in I, S \in \mathcal{S}(j)$.*

Proof. Suppose on the contrary there exists a variable with fractional value $y_{S'}^{j'}$. Since x_{ij}^k are binary for all i,j,k (in particular for i_1, j', k_1 where (i_1, k_1) is a pair of S'), there must be another fractional variable $y_{S''}^{j'}$ such that $(i_1, k_1) \in S''$.

Note that $S'' \neq S'$ since the column generation procedure never generates duplicate variables. Hence, there is a pair (i_2, k_2) such that either $(i_2, k_2) \in S'$ or $(i_2, k_2) \in S''$ but not both. Therefore, we obtain the following relationship

$$1 \geq \sum_{\substack{S \in \mathcal{S}(j'): \\ (i_1, k_1) \in S}} y_S^{j'} > \sum_{\substack{S \in \mathcal{S}(j'): \\ (i_2, k_2) \in S}} y_S^{j'} > 0. \tag{35}$$

The first inequality comes directly from the formulation. The second inequality is strict because the term $\sum_{S \in \mathcal{S}(j'):(i_2,k_2) \in S} y_S^{j'}$ has at least one fractional variable less than the term $\sum_{S \in \mathcal{S}(j'):(i_1,k_1) \in S} y_S^{j'}$. The third inequality is strict because of the choice of (i_2, k_2). Finally, a contradiction is found because $x_{i_2 k_2}^{j'}$ is not binary. $\square$

The reader may note that this branching can be seen as a SOS1 branching (Beale and Tomlin (1970)) since at most one of the above y_S^j variables can assume the value 1.

The way to implement this branching in the pricing subproblem is to set locally (in the current node) to zero the y_S^j variables which are in conflict with the condition implied by the branch $x_{ij}^k = 0$ or $x_{ij}^k = 1$.

In the case $x_{ij}^k = 0$ we set $y_S^j = 0$ for all sets S containing couples $(i,k) \in S$. Analogously, in the case $x_{ij}^k = 1$ we set $y_S^{j'} = 0$ for all sets S containing $(i,k) \in S$ such that $j \neq j'$, $(i', k) \in S$ such that $i \neq i'$ or $(i, k') \in S$ such that $k \neq k'$.

This condition can be transferred to the pricing subproblem modifying the d_{ij}^k coefficients accordingly. Specifically, this transformation is done as follows:

- If $x_{ij}^k = 0$ then $d_{ij}^k = 0$.

- If $x_{ij}^k = 1$ then $\begin{cases} d_{ij'}^k &=& 0, & j' \in I : j' \neq j. \\ d_{i'j'}^k &=& 0, & j', i' \in I : i' \neq i. \\ d_{ij'}^{k'} &=& 0, & j' \in I, k' \in K : k' \neq k. \end{cases}$

Moreover, it is also well-known that branching on SOS constraints (original variables) gives rise to more balanced branching trees (see e.g. Chapter 7 of Wolsey (1998)) than branching on the variables of MP.

Among the fractional original variables one has to decide which will be the next variable to branch on. One of the easiest techniques for this choice is to consider the *most fractional variable*. This is not difficult to implement but it is not better than choosing randomly (Achterberg et al. (2005)). Alternative techniques are *pseudocost branching* (Benichou et al. (1971)) or *strong branching* (Applegate et al. (1995)) although they are rather costly.

This issue has motivated us to propose another rule to select the variable to branch on, based on the improvement of the bounds in each of the new created nodes. We use the following indices corresponding to the down and up branches of the variable x_{ij}^k:

$$\varsigma_{ij}^{k,-} = \frac{\lambda^k c_{ij}}{x_{ij}^k} \text{ and } \varsigma_{ij}^{k,+} = \frac{\lambda^k c_{ij}}{1 - x_{ij}^k}. \tag{36}$$

They account, respectively, for the unitary contribution to the objective function due to fixing the variable x_{ij}^k either to zero (down branching) or to one (up branching). Branching down stimulates the improvement of the lower bound, whereas branching up helps the problem to find integer solutions.

We have tested several strategies that make use of the indices, ς, defined above.

Strategy 1: $\arg\min\{\theta\varsigma_{ij}^{k,-} + (1-\theta)\varsigma_{ij}^{k,+} : 0 < x_{ij}^k < 1\}$.

Strategy 2: $\arg\min\{\min\{\varsigma_{ij}^{k,-}, \varsigma_{ij}^{k,+}\} : 0 < x_{ij}^k < 1\}$.

Strategy 3: $\arg\min\{\max\{\varsigma_{ij}^{k,-}, \varsigma_{ij}^{k,+}\} : 0 < x_{ij}^k < 1\}$.

Based on our computational experience (see Figure 4), we have concluded that the best strategy to choose the following variable to branch on corresponds to strategy 1 with $\theta = 0.5$.

Each node of the branching tree can be fathomed before it is fully processed comparing lower bounds, as given by (25) and (26), with the current incumbent solution. This strategy implies reducing the number of calls to the pricing subproblem and as a result savings in the number of variables added to the restricted master problem.

3.5   Valid inequalities

Clearly, the addition of valid inequalities (20) to MP modifies the structure of the master problem and thus the pricing must be modified accordingly. Let us denote by ζ_{ij}^k the dual variable associated with valid inequality (20) for indices i, j, k. After some calculation, one
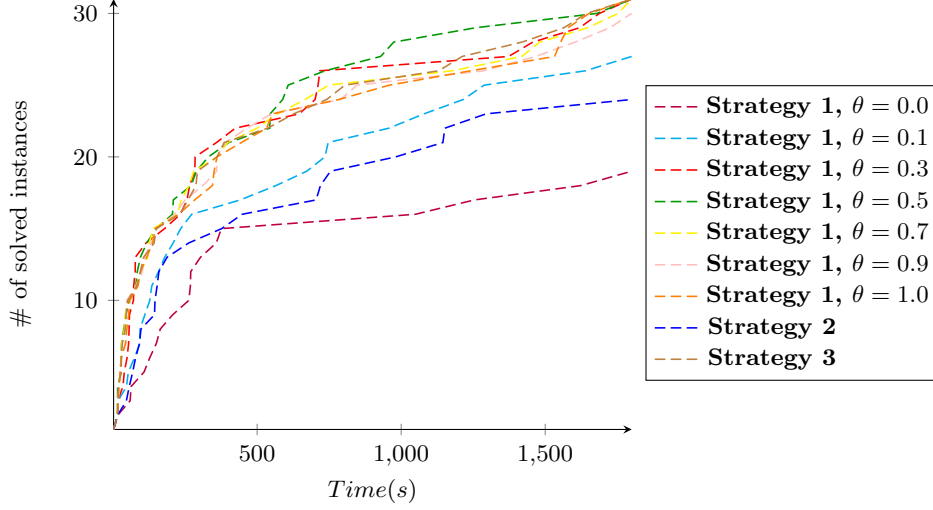
Figure 4: Performance profile graph of #solved instances using different branching strategies.

obtains the following expression of the reduced costs of variable y_S^j:

$$\bar{c}_S^j = c_S^j + \gamma_j^* + \delta^* + \sum_{i' \in I} \sum_{j' \in I} \left(\sum_{\substack{(i,k) \in S: \\ r_{i'j'} \geq r_{ij} \\ k \neq 1}} (\epsilon_k^* + \zeta_{i'j'}^{k*}) + \sum_{\substack{(i,k) \in S: \\ r_{i'j'} \leq r_{ij} \\ k \neq n}} (\epsilon_{k+1}^* + \zeta_{i'j'}^{(k+1)*}) \right) - \sum_{\substack{i \in I: \\ (i,\cdot) \in S}} \alpha_i^* - \sum_{\substack{k \in K: \\ (\cdot,k) \in S}} \beta_k^*. \tag{37}$$

Furthermore, solving the pricing subproblem to find a new column or to certify optimality of the column generation algorithm requires to adapt the dynamic programming algorithm that computes the $g(i_l, k)$ terms using the new dual multipliers. This implies to modify the D_j matrices. Once again, after some calculations the modified d_{ij}^k elements are now given by:

$$d_{ij}^k = \begin{cases} \lambda^k c_{ij} + \sum_{i' \in I} \sum_{\substack{j' \in I: \\ r_{i'j'} \leq r_{ij}}} (\epsilon_{k+1} + \zeta_{i'j'}^{k+1}) - \alpha_i - \beta_k & \text{if } k = 1, \\ \lambda^k c_{ij} + \sum_{i' \in I} \sum_{\substack{j' \in I: \\ r_{i'j'} \geq r_{ij}}} (\epsilon_k + \zeta_{i'j'}^k) + \sum_{i' \in I} \sum_{\substack{j' \in I: \\ r_{i'j'} \leq r_{ij}}} (\epsilon_{k+1} + \zeta_{i'j'}^{k+1}) - \alpha_i - \beta_k & \text{if } k = 2, \ldots, n-1, \\ \lambda^k c_{ij} + \sum_{i' \in I} \sum_{\substack{j' \in I: \\ r_{i'j'} \geq r_{ij}}} (\epsilon_k + \zeta_{i'j'}^k) - \alpha_i - \beta_k, & \text{if } k = n. \end{cases} \tag{38}$$

These new elements allow us to apply the adapted column generation algorithm to solve LRMP, reinforced with valid inequalities (20).

To justify the use of the mentioned cuts we have done some preliminary computational experiments with instances of sizes $n = 50$ and 60. Table 3 compares the behavior of the standard branch-and-price without cuts, (**B&P**(MP)), against the strategy with cuts, **B&P&C**(MP).

From Table 3, we conclude that it is always better to add cuts because the final gap is always smaller with this strategy. This solution scheme has been implemented and the results are reported in the next section.

			$n = 50$			$n = 60$	
		$p = 12$	$p = 16$	$p = 25$	$p = 15$	$p = 20$	$p = 30$
B&P	$Time(s)$	7200.03	7200.01	7200.01	7200.07	7200.02	7202.00
(MP)	$|Vars|$	48648	38368	21558	49990	34262	24730
	$|Nodes|$	4828	11768	49523	5329	12526	30305
	$\#unsolved$	10	10	10	10	10	10
	$Gap(\%)$	6.26	7.57	9.57	8.29	8.97	12.07
B&P&C	$Time(s)$	7200.39	7200.49	6860.93	7200.08	7200.77	7200.20
(MP)	$|Vars|$	14807	14977	13407	17131	16691	16838
	$|Nodes|$	1	6	9	2	1	11
	$|Cuts|$	3526	3066	2709	3489	4192	2864
	$\#unsolved$	10	10	9	10	10	10
	$Gap(\%)$	2.83	2.89	1.87	5.04	4.52	4.47

Table 3: Numerical results with and without cuts.

4   Computational Experiments

The **B&P&C** implementation of the formulation MP has been experimentally compared with the **B&C** implementation of the formulation WOC on the instances detailed below. The **B&P&C** algorithm considered in these experiments is based on the description in the previous section.

The computer used for these tests has an Intel Core i7 CPU clocked at 2.8 GHz with 4GB of RAM. Each implementation has a maximum of 7,200 seconds (2 hours) to solve each individual instance.

Both implementations are using the SCIP 4.0.1's API (see Gamrath et al. (2016)) and both are calling the LP solver of IBM ILOG CPLEX 12.7.

4.1   Instances

Since no standard libraries of instances for DOMP are available in public repositories we generate our own instances with the *pseudorandom* number generator from the C random library. In this work we consider that the sets of clients and potential facilities coincide, thereby we will refer both as points.

We consider 20 sets of 30 instances. Each set has a different number of points such that $n \in \{20, 30, \ldots, 90, 100, 120, 140, \ldots, 280, 300, 400\}$. For a given n, we generate one subset of 10 instances for each value of p, where $p \in \{\lfloor (n/4) \rfloor, \lfloor (n/3) \rfloor, \lfloor (n/2) \rfloor\}$.

For a given n, we first randomly generate the Cartesian coordinates of the points in the square $[0, 400]^2$. Then, we calculate the cost for each pair of points with the Euclidean distance between the two related nodes in the square. We round each distance to the nearest integer to build the cost matrices. We also fix the values of the matrix diagonal to the smallest admissible cost to avoid free self service.

Finally, we randomly generate the weight vector λ such that $\lambda^k \in [n/4, n]$ for $k \in K$. All these instances, with n up to 400, are available at `https://gom.ulb.ac.be/gom/wp-content/uploads/2018/12/DOMP_Repository.zip`. Detailed information about the instances generation can be found in Deleplanque et al. (2018).

4.2 MP vs WOC linear relaxations

We assess experimentally the linear relaxation of MP by comparing with WOC on all the instances generated. For these experiments, neither cuts nor preprocessing have been applied.

In Table 4, we report averages of the numerical results of the linear relaxation for both formulations. The value $GapLP(\%)$ represents the gap percentage between the optimal integer value z^* (alternatively the best known solution) and the linear relaxation optimal value z_{LP}^* : $GapLP(\%) = 100(z^* - z_{LP}^*)/z_{LP}^*$. The computational times in the column $Time(s)$ are given in seconds. Table 4 also includes average number of variables ($|Vars|$) and required memory ($Memory(MB)$). We highlight the small number of variables that are generated to certify optimality with the column generation approach applied to MP, besides the time and memory saving, which is likewise significant.

		(WOC)				(MP)			
n	p	$GapLP(\%)$	$Time(s)$	$|Vars|$	$Memory(MB)$	$GapLP(\%)$	$Time(s)$	$|Vars|$	$Memory(MB)$
20	10	15.35	0.12	8020	35	14.78	0.09	450	2
30	15	16.77	0.62	27030	101	16.16	0.29	1067	5
40	20	18.22	2.09	64040	235	17.98	0.74	2005	10
50	25	15.02	6.25	125050	451	14.74	1.36	2922	15
60	30	16.94	12.27	216060	764	16.69	2.74	4777	23
70	35	15.97	29.40	343070	1214	15.80	4.61	6327	30
80	40	7.67	47.63	512080	1830	7.63	5.63	7161	34
90	45	7.19	82.19	729090	2561	7.15	9.14	8867	42
100	50	–	–	1000100	>4096	7.05	14.18	11255	54
20	6	9.56	0.14	8020	35	8.79	0.14	548	3
30	10	11.68	0.68	27030	101	10.96	0.39	1257	7
40	13	12.39	2.34	64040	235	12.08	1.17	2371	14
50	16	9.65	6.37	125050	451	9.34	2.09	3641	21
60	20	11.11	13.30	216060	764	10.82	4.18	5652	32
70	23	10.16	35.65	343070	1214	9.98	7.54	7684	44
80	26	8.32	58.14	512080	1830	8.11	11.32	9699	55
90	30	7.08	96.74	729090	2561	7.02	17.58	12185	69
100	33	–	–	1000100	>4096	8.01	28.76	15656	87
20	5	9.53	0.14	8020	35	8.61	0.16	574	4
30	7	10.61	0.70	27030	101	9.36	0.59	1278	9
40	10	10.44	2.51	64040	235	10.05	1.61	2536	17
50	12	7.96	7.35	125050	451	7.54	3.01	3955	27
60	15	9.74	15.98	216060	764	9.41	5.56	5697	36
70	17	8.80	40.78	343070	1214	8.50	10.19	7828	53
80	20	9.53	67.42	512080	1830	9.38	17.38	10740	69
90	22	9.58	128.70	729090	2561	9.41	28.66	13581	86
100	25	–	–	1000100	>4096	8.65	44.20	17084	106

Table 4: Numerical results on linear relaxation for WOC and MP.

As expected, the integrality gap of formulation MP is smaller than the one of WOC. Moreover, formulation MP also outperforms WOC in the number of required variables (see Figure 5) which results in much smaller memory requirements (see Figure 6). Indeed, the implementation of WOC fails to solve, already for sizes of $n = 100$, the linear relaxation of all instances by lack of RAM memory; whereas with the same parameter configuration, formulation MP is relatively far from experiencing that problem. Figure 6 shows the performance profile of the memory requirement of both formulations. As one can see MP outperforms WOC with respect to this factor for all instance sizes.

4.3 B&P&C(MP) vs B&C (WOC)

We now compare the **B&P&C** implementation of MP with the **B&C** implementation of WOC. The former follows the procedure explained in Section 3 and the latter consists of WOC formulation with (9) as valid inequalities.

The results are reported in Table 5. In that table, we denote by $Time(s)$ the average computational time (in seconds) required by each method to obtain an optimal solution for a given set of 10 instances defined by number of clients (n) and number of open facilities (p).
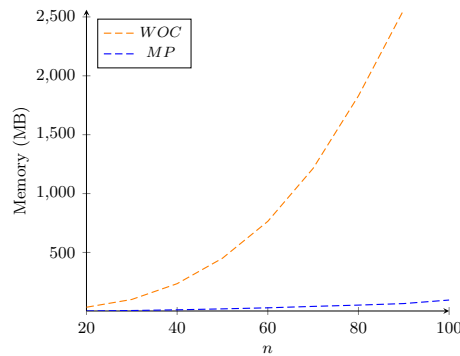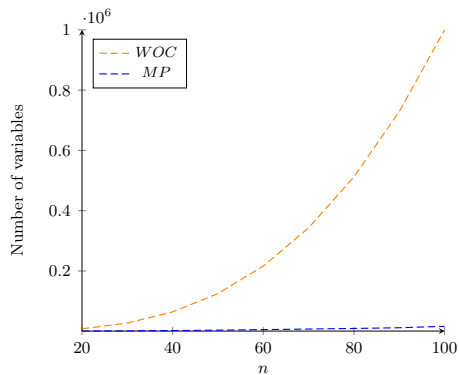
Figure 5: Graph of Number of Variables versus size n for WOC and MP.

Figure 6: Graph of Memory usage (MB versus size n for WOC and MP.

		B&C(WOC)						**B&P&C**(MP)					
n	p	$Time(s)$	$|Vars|$	$|Nodes|$	$|Cuts|$	$\#unsolved(T/M)$	$Gap(\%)$	$Time(s)$	$|Vars|$	$|Nodes|$	$|Cuts|$	$\#unsolved$	$Gap(\%)$
20	10	4.48	4211	38	689	0/0	0.00	58.70	4272	111	373	0	0.00
30	15	131.89	13952	19197	2519	0/0	0.00	3670.21	16369	392	1078	3	0.45
40	20	6202.85	32820	605812	4727	8/0	2.54	6751.27	14591	41	1819	9	3.33
50	25	6575.59	63776	355560	10131	9/0	1.35	6860.93	13407	9	2709	9	1.87
60	30	6707.38	109804	85723	15676	8/2	1.82	7200.20	16838	11	2864	10	4.47
70	35	2474.98	173955	835	19238	2/8	7.72	7200.31	17758	2	4150	10	4.95
80	40	3428.13	259186	1	12406	0/10	3.38	7201.36	18902	2	4680	10	2.13
90	45	6243.49	368560	1	12157	7/3	4.24	7200.38	20028	7	4127	10	2.37
20	6	11.50	5706	440	1249	0/0	0.00	952.76	10959	97	615	0	0.00
30	10	1578.21	18245	305595	3056	1/0	0.12	6270.41	17502	81	1503	8	1.61
40	13	7061.36	43664	628962	6559	8/2	2.38	7200.72	11186	2	3073	10	3.69
50	16	7116.54	85630	284028	10423	9/1	1.14	7200.49	14977	6	3066	10	2.89
60	20	3306.54	144983	20330	19887	2/8	3.02	7200.77	16691	1	4192	10	4.52
70	23	2119.13	231680	1	23603	0/10	6.20	7200.97	19307	2	4365	10	4.51
80	26	2886.25	346926	1	25187	0/10	5.59	7201.35	21675	1	5449	10	3.69
90	30	5214.89	488316	1	32406	0/10	4.62	7201.66	24507	1	6116	10	3.17
20	5	16.54	6054	1215	1537	0/0	0.00	1989.43	14699	97	731	1	0.09
30	7	1807.41	20643	198424	4789	1/1	0.65	6840.10	14934	29	2093	8	1.85
40	10	7050.93	48065	602685	7939	7/3	1.68	7200.90	10730	1	3455	10	3.95
50	12	7200.00	94784	270959	12579	10/0	0.91	7201.39	14807	1	3526	10	2.83
60	15	2768.88	161807	1	18081	0/8	2.90	7201.08	17131	2	3489	10	5.04
70	17	1842.00	259406	1	16115	0/10	6.04	7201.78	19454	1	4649	10	4.51
80	20	2902.00	383199	1	27129	0/10	6.95	7201.52	25278	3	4320	10	4.89
90	22	5999.16	549561	1	46216	0/10	6.82	7201.75	27418	1	5549	10	5.50

Table 5: Numerical results for **B&C**(WOC) and **B&P&C**(MP).

With $|Vars|$ we refer to the average of the numbers of variables used by MP or WOC. We also denote by $|Nodes|$ and $|Cuts|$ the average of the number of nodes explored and the average of the number of cuts used, respectively, in the corresponding methodology. The column $\#unsolved(T/M)$ in the case of **B&C**(WOC) reports the number of unsolved instances out of the 10 in each group. It distinguishes between those instances not solved by exceeding the maximum running time (T) or the memory limits (M). Observe that in the similar column within the blocks **B&P&C**(MP) no distinction is shown since the memory limit is never reached and instances may be not solved only because of time limitation. Finally, we also include the gap at termination as $Gap(\%) = 100(z_{UB} - z_{LB})/z_{LB}$, where z_{LB} and z_{UB} are the lower and upper bound, respectively.

Analyzing further the results in Table 5 we conclude that on average **B&C**(WOC) uses less variables than **B&P&C**(MP). This allows us to solve larger sized instances that were not affordable for the original WOC. We also observe that the number of required cuts for **B&P&C**(MP) is smaller than for **B&C**(WOC). This could be explained by the tightness

of **B&P&C**(MP) with respect to **B&C**(WOC). After adding cuts **B&P&C**(MP) is able to solve the problem with a smaller branch-and-bound tree. The number of instances solved to optimality, for small size instances up to $n = 40$, is slightly better for **B&C**(WOC). As the size increases this number is similar in both cases. Gaps at termination, after 7200 seconds, are always smaller than 8% for **B&C**(WOC) and smaller than 6% for **B&P&C**(MP), being the later clearly better from instances of $n = 70$. Since **B&C**(WOC) is not able to handle any instance with $n = 100$ (reporting out of memory flags) we continue our study in Table 6 without this formulation.

n	p	$Time(s)$	$|Vars|$	$|Nodes|$	$|Cuts|$	$Memory(MB)$	$Gap(\%)$
100	50	7201.10	23057	2	5221	311	2.43
120	60	7200.54	29209	1	6992	424	2.54
140	70	7202.11	34222	2	6805	449	2.87
160	80	7201.08	41811	1	8069	574	3.13
180	90	7201.24	50523	1	7961	656	3.48
200	100	7201.61	59931	1	9050	805	3.96
220	110	7201.39	57685	1	11097	806	4.39
240	120	7200.94	63710	1	11487	874	4.43
260	130	7201.83	73143	1	9772	910	5.05
280	140	7200.98	83892	1	9592	1037	6.19
300	150	7201.74	87444	1	10811	1076	6.83
100	33	7201.10	29974	2	5864	562	3.81
120	40	7201.12	37204	5	5433	621	4.80
140	46	7200.98	48286	2	7953	894	4.47
160	53	7201.55	57060	3	7159	926	5.01
180	60	7203.85	70077	1	8997	1188	5.99
200	66	7200.55	67417	1	11416	1199	6.16
220	73	7201.41	71731	2	9356	1073	6.88
240	80	7202.02	84466	2	10093	1324	9.80
260	86	7200.74	92401	1	10505	1390	9.54
280	93	7201.68	105535	1	13265	1728	9.01
300	100	7200.28	114427	1	15595	2034	9.41
100	25	7202.69	31218	1	7034	801	4.73
120	30	7200.98	40074	1	6818	922	5.91
140	35	7203.38	49539	1	8466	1162	5.83
160	40	7201.35	58464	1	10688	1465	7.29
180	45	7203.43	72146	1	10329	1674	8.70
200	50	7201.88	62568	1	10252	1364	11.27
220	55	7203.56	74359	1	9985	1515	10.34
240	60	7200.19	80228	1	9745	1422	11.57
260	65	7200.54	99628	1	7944	1346	11.77
280	70	7203.63	109544	1	4187	1247	11.32
300	75	7200.79	128462	1	3844	1261	12.26
400	200	[1]86400.42	158287	1	9308	1183	7.07
400	133	86401.23	178265	1	13652	2764	11.26
400	100	86401.11	236973	1	7582	3125	10.29

Table 6: Numerical results for **B&P&C**(MP) for bigger instances.

Table 6 contains the results within the time limit of 2 hours for bigger instances of DOMP. This table has the same layout as Table 5 except that we replace column $\#unsolved$ by $Memory(MB)$. This new column shows the average required memory to solve the corresponding set of instances. In that table extensive computational experiments are reported for instances up to 400 points. We would like to remark that the increase of the complexity, with respect to the instance sizes, of $|Vars|$, $|Cuts|$, $Memory(MB)$ and $Gap(\%)$ is moderate (almost linear) which allows one to handle DOMP problems of larger size. Moreover the $Gap(\%)$ are similar to those reported in Table 5.

To conclude, the results show that the overall performance of **B&P&C**(MP) in solving DOMP is systematically better than the branch-and-cut formulation **B&C**(WOC) for instances for $n \geq 70$. In addition, it is worth noting that **B&C**(WOC) is not even able to solve

the linear relaxation of DOMP problems of sizes $n \geq 100$. This fact shows the usefulness of our new approach.

5   Conclusions

This paper presents a first branch-price-and-cut, **B&P&C**(MP), algorithm for solving DOMP. This approach is based on an extended formulation using an exponential number of variables coming from a set partitioning model. Elements in the partitions are couples containing information about a client and its sorted position in the sorted sequence of allocation costs. To address the solution of this formulation we develop a column generation algorithm and we prove that the pricing routine is polynomially solvable by a dynamic programming algorithm. We embed the column generation algorithm within a branch-and-price framework. Furthermore, we adapt preprocessing and incorporate families of valid inequalities that improve its performance. Extensive computational results compare the performance of our **B&P&C**(MP) against the most recent algorithm in the literature for DOMP, **B&C**(WOC), showing that for the largest considered instances **B&P&C**(MP) performs better and it requires less memory to upload and run the models. The methodology presented in this paper is able to solve sized instances for DOMP that had never been solved in the literature.

Acknowledgements

References

Achterberg T, Koch T, Martin A (2005) Branching rules revisited. *Operations Research Letters* 33:42–54.

Applegate D, Bixby R, Chvátal V, Cook W (1995) Finding cuts in TSP. *DIMACS TR* 95-05.

Barnhart C, Johnson E, Nemhauser G, Savelsbergh M, Vance P (1998) Branch-and-price: column generation for solving huge integer programs. *Operations Research* 46:316–329.

Beale E, Tomlin J (1970) Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. Lawrence J, ed., *Proceedings of the Fifth International Conference on Operational Research* (Tavistock Publications, London).

Benichou M, Gauthier J, Girodet P, Hentges G, Ribiere G, Vincent O (1971) Experiments in mixed-integer programming. *Mathematical Programming* 1:71–94.

Boland N, Domínguez-Marín P, Nickel S, Puerto J (2006) Exact procedures for solving the discrete ordered median problem. *Computers & Operations Research* 33(11):3270–3300.

Ceselli A, Gatto M, Lübbecke ME, Nunkesser M, Schilling H (2008) Optimizing the Cargo Express Service of Swiss Federal Railways. *Transportation Science* 42(4):450–465.

---

[1]For instances with $n = 400$ the time limit was set to 24 hours.

Chvátal V (1983) *Linear Programming* (W. H. Freeman and Company).

Deleplanque S, Labbé M, Ponce D, Puerto J (2018) An extended version of a Branch-Price-and-Cut Procedure for the Discrete Ordered Median Problem. *Arxiv 2158317-v1*.

Desrosiers J, Lübbecke M A primer in column generation. Desaulniers G, Desrosiers J, Solomon MM, eds., *Column Generation*, Springer, 2005.

Doulabi SHH, Rousseau LM, Pesant G (2016) A constraint-programming-based branch-and-price-and-cut approach for operating room planning and scheduling. *INFORMS Journal on Computing* 28(3):432–448.

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

Farkas G (1894) A Fourier-féle mechanikai elv alkalmazásai. *Mathematikai és Természettudományi Érstesitö* (12): 457–472.

Feo TA, Resende MGC (1989) A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters* 8:67–71.

Feo TA, Resende MGC (1995) Greedy ramdomized adaptive search procedures. *Journal of Global Optimization* 6:109–133.

Fernández E, Puerto J, Rodríguez-Chía AM (2013) On Discrete Optimization with Ordering. *Annals of Operations Research* 207(1): 83–96.

Fernández E, Pozo MA., Puerto J (2014) Ordered Weighted Average Combinatorial Optimization: Formulations and their properties. *Discrete Applied Mathematics* 169(31): 97–118.

Fernández E, Pozo MA., Puerto J, Scozzari A (2017) Ordered Weighted Average Optimization in Multiobjective Spanning Tree Problems. *European Journal of Operational Research* 260(31): 886–903.

Gamrath G (2010) Generic Branch-Cut-and-Price. *Diploma Thesis*

Gamrath G, Fischer T, Gally T, Gleixner AM, Hendel G, Koch T, Maher SJ, Miltenberger M, Müller B, Pfetsch ME, Puchert C, Rehfeldt D, Schenker S, Schwarz R, Serrano F, Shinano Y, Vigerske S, Weninger D, Winkler M, Witt JT, Witzig J (2016) The SCIP optimization suite 3.2. Technical Report 15-60, ZIB, Takustr.7, 14195 Berlin.

Günlük O, Ladányi L, de Vries S (2005) A Branch-and-Price Algorithm and New Test Problems for Spectrum Auctions. *Management Science* 51(3):391–406.

Johnson EL (1989) Modeling and Strong Linear Programs for Mixed Integer Programming. Wallace S, ed., *Algorithms and Model Formulations in Mathematical Programming*, volume 51 of *NATO ASI Series*, 1–43 (Springer Berlin Heidelberg), ISBN 978-3-642-83726-5.

Labbé M, Ponce D, Puerto J (2017) A comparative study of formulations and solution methods for the discrete ordered p-median problem. *Computers & Operations Research* 78:230 – 242.

Marín A, Nickel S, Puerto J, Velten S (2009) A flexible model and efficient solution strategies for discrete location problems. *Discrete Applied Mathematics* 157(5):1128–1145.

Marín A, Nickel S, Velten S (2010) An extended covering model for flexible discrete and equity location problems. *Mathematical Methods of Operations Research* 71(1):125–163.

Nickel S (2001) Discrete ordered weber problems. *Operations Research Proceedings 2000*, 71–76 (Springer).

Nickel S, Puerto J (1999) A unified approach to network location problems. *Networks* 34:283–290.

Nickel S, Puerto J (2005) *Location Theory: A Unified Approach.* (Springer).

Perea F, Puerto J (2013) Finding the nucleolus of any $n-$person cooperative game by a single linear program. *Computers & Operations Research* 40(10):2308–2313.

Pessoa A, Uchoa E, Aragão MP, Rodrigues R (2010) Exact Algorithm over an Arctime-Indexed Formulation for Parallel Machine Scheduling Problems. *Mathematical Programming Computation* 2:259–290.

Ponce D, Puerto J, Ricca F, Scozzari A (2018) Mathematical programming formulations for the efficient solution of the k-sum approval voting problem. *Computers & Operations Research* 98:127 – 136.

Puerto J (2008) A new formulation of the capacitated discrete ordered median problem with $\{0, 1\}$-assignment. *Operations Research Proceedings 2007*, 165–170 (Springer), ISBN 978-3-540-77902-5.

Puerto J, Fernández F (2000) Geometrical properties of the symmetrical single facility location problem. *Journal of Nonlinear and Convex Analysis* 1(3):321–342.

Puerto J, Pérez-Brito D, García-González CG (2014) A modified variable neighborhood search for the discrete ordered median problem. *European Journal of Operational Research* 234:61–76.

Puerto J, Rodríguez-Chía AM, Tamir A (2009) Minimax regret single-facility ordered median location problems on networks. *INFORMS Journal on Computing* 21(1):77–87.

Puerto J, Ramos AB, Rodríguez-Chía AM (2011) Single-allocation ordered median hub location problems. *Computers & Operations Research* 38(2):559–570.

Puerto J, Ramos AB, Rodríguez-Chía AM (2013) A specialized Branch & Bound & Cut for Single-Allocation Ordered Median Hub Location problems. *Discrete Applied Mathematics* 161(16–17):2624–2646.

Puerto J, Ramos AB, Rodríguez-Chía AM, Sánchez-Gil MC (2016) Ordered Median Hub Location Problems with Capacity Constraints. *Transportation Research Part C: Emerging Technologies* 70:142–156.

Puerto J, Rodríguez-Chía AM Ordered Median Location Problems. Laporte G, Nickel S, Saldanha da Gama F, eds., *Location Science*, Springer, 2015.

Ryan DM, Foster A (1981) An integer programming approach to scheduling. Wren A, ed., *Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling*, 269–280 (Amsterdan: North-Holland).

Wolsey LA (1998) *Integer programming* (New York (N.Y.), Chichester, Weinheim: J. Wiley & sons).