http://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING
Articles in Advance, pp. 1–18
ISSN 1091-9856 (print), ISSN 1526-5528 (online)

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

Samuel Deleplanque,[a] Martine Labbé,[b] Diego Ponce,[c,d,e] Justo Puerto[c]

[a] Ifsttar, COSYS, ESTAS, Université Lille Nord de France, 59000 Lille, France; [b] Département d'Informatique, Faculté des Sciences, Université Libre de Bruxelles, 1050 Bruxelles, Belgium; [c] Instituto de Matemáticas de la Universidad de Sevilla (IMUS), 41012 Sevilla, Spain; [d] Department of Mechanical, Industrial and Aerospace Engineering, Concordia University, Montreal, Quebec H3G 1M8, Canada; [e] Centre interuniversitaire de recherche sur les réseaux d'enterprise, la logistique et le transport (CIRRELT), Montreal, Quebec H3T 1J4, Canada

Contact: samuel.deleplanque@ifsttar.fr, http://orcid.org/0000-0003-4119-6006 (SD); mlabbe@ulb.ac.be, http://orcid.org/0000-0001-7471-2308 (ML); dponce@us.es, http://orcid.org/0000-0003-3380-6601 (DP); puerto@us.es, http://orcid.org/0000-0003-4079-8419 (JP)

Received: February 9, 2018
Revised: December 7, 2018; March 11, 2019
Accepted: June 2, 2019
Published Online in Articles in Advance: January 7, 2020

https://doi.org/10.1287/ijoc.2019.0915

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.

Funding: This research was supported by Spanish/FEDER [Grant MTM2016-74983-C02-01-R]. The research of the second and third authors was supported by the Interuniversity Attraction Poles Programme initiated by the Belgian Science Policy Office.

History: Accepted by Andrea Lodi, Area Editor for Design & Analysis of Algorithms—Discrete.

Keywords: discrete optimization • location theory • branch and price • ordered median problems

1. Introduction

Broadly speaking, a facility-location problem consists of locating one or several facilities to minimize an objective function of the assignment costs of clients to facilities. Median and center location problems constitute the most popular ones. The ordered median location problem is a flexible model that provides a common framework to cast most popular location problems. The objective function to be minimized is a weighted function of the allocation costs in which the weights are assigned to the ordered values of the costs rather than to specific costs. 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).

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

DOMP has been widely studied since the 1990s, and there are 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), and 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 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 monographs by Nickel and Puerto (2005) and Puerto and Rodríguez-Chía (2015) for some other applications.

There exist several valid formulations for DOMP that exploit specific features of the problem, for instance, free self-service, ties in the matrix of costs, or null elements in the vector of weights (see, e.g., Boland et al. (2006), Marín et al. (2009, 2010), Puerto et al. (2013), 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-sized instances when embedded in a branch-and-cut (B&C) scheme. For general cost coefficients (with no ties), all these formulations have a very large (cubic) number of

binary variables, and therefore, already for instances with 100 clients, they fail to be loaded by the solvers.

In this paper, we explore a different paradigm for solving DOMP based on column generation that avoids considering explicitly all the variables of the problem. Moreover, we introduce a new extended formulation using an exponential number of variables that corresponds to a set-partitioning model and provides even better linear relaxation lower bounds. Each variable represents a set of couples (client, position). In each element of the partition, its clients are served by the same facility, and their positions indicate the place of their allocation costs 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.

Branch-and-price algorithms to solve location problems have been proposed by du Merle and Vial (2002), Lorena and Senne (2004), Senne et al. (2005), Ceselli and Righini (2005), Avella et al. (2006), and Contreras et al. (2011) to cite a few. In most cases, the considered problem is the p-median although there are some exceptions, for instance, Contreras et al. (2011) for capacitated hub location or Doulabi et al. (2016) for a different problem.

A branch-and-price approach has never been applied to DOMP, and even more, our approach based on a set partition for couples is fully new. These two facts open new avenues of research in the field of location analysis. Therefore, the contributions of this paper are twofold: (1) methodological, 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, and (2) numerical, to provide solutions for large instances of DOMP. Moreover, in those cases in which optimality of a solution cannot be certified, our approach provides, at least, valid lower bounds that can be used to measure the quality of feasible solutions of DOMP given either by heuristic algorithms (Domínguez-Marín et al. 2005, Stanimirovic et al. 2007, Puerto et al. 2014, Olender and Ogryczak 2018).

The remainder of this paper is organized as follows. Notation, models, and algorithms are presented in Section 2. Section 2.2 introduces a new set-partitioning formulation for DOMP. This formulation uses an exponential number of variables in which 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 Section 2.1 and borrowed from Labbé et al. (2017). Section 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 Section 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 Section 3.2. In addition, Section 3.3 is devoted to an additional improvement, namely a preprocessing. The next two sections, 3.4 and 3.5, present our branching strategies and some families of valid inequalities that are 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 that, 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, that is, $c_i(J) := \min_{j\in J} c_{ij}$.

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

Given the vector $\lambda = (\lambda^k)_{k\in K}$ satisfying $\lambda^k \ge 0, k \in K$, 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 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 is the starting point for the developments presented in this paper. This formulation uses two types of binary variables. Variable y_j assumes value one if facility $j \in I$ is open (i.e., $j \in J$) and zero otherwise. Variable x_{ij}^k is equal to one 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 because 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}, that is, $r_{ij} = \ell$ if c_{ij} is the ℓth element in the list of the costs c_{ij} for all $i, j \in I$, sorted by order of nondecreasing values and for which 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:

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

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

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

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

$$\sum_{j\in I} y_j = p \tag{6}$$

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

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

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} occupy the kth 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 are 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:

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

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 preceding 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 solving 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 weights (λ^k) with many ties.

These reasons 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 (respectively, second) coordinates of its couples are different.

Associated with each set S and facility j, we define a variable y_S^j equal to one if the set S is part of a feasible solution ($(i, k) \in S$ iff $x_{ij}^k = 1$) and zero 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, that is, 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 subsets of couples 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 $k < k'\}$.

Because, 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):

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

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

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

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

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

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

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

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} that occupies a given position k. Hence, the following constraints are valid for WOC:

$$\sum_{i \in I} x_{ij}^k \leq y_j, j \in I, k \in K. \tag{19}$$

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 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 $x_{ij}^k \geq 0$ and $y_j \leq 1$, 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 Section 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

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

2.3. Column Generation to Solve the Linear Relaxation of *MP* (LRMP)

Because 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)–(17). The dual problem *DP* of LRMP is

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

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

$$\delta,\gamma_j,\epsilon_k \ge 0 \quad j\in I, k\in K, k\neq 1. \tag{23}$$

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

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

If $\bar{c}_S^j \ge 0$ 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)

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

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

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:

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

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

$S_1 = \{(1,1)\}$
$S_2 = \{(1,2)\}$
$S_3 = \{(1,3)\}$
$S_4 = \{(2,1)\}$
$S_5 = \{(2,2)\}$
$S_6 = \{(2,3)\}$
$S_7 = \{(3,1)\}$
$S_8 = \{(3,2)\}$
$S_9 = \{(3,3)\}$
$S_{10} = \{(1,1),(2,2)\}$
$S_{11} = \{(1,1),(2,3)\}$
$S_{12} = \{(1,1),(3,2)\}$
$S_{13} = \{(1,1),(3,3)\}$
$S_{14} = \{(1,2),(2,1)\}$
$S_{15} = \{(1,2),(2,3)\}$
$S_{16} = \{(1,2),(3,1)\}$
$S_{17} = \{(1,2),(3,3)\}$
$S_{18} = \{(1,3),(2,1)\}$
$S_{19} = \{(1,3),(2,2)\}$
$S_{20} = \{(1,3),(3,1)\}$
$S_{21} = \{(1,3),(3,2)\}$
$S_{22} = \{(2,1),(3,2)\}$
$S_{23} = \{(2,1),(3,3)\}$
$S_{24} = \{(2,2),(3,1)\}$
$S_{25} = \{(2,2),(3,3)\}$
$S_{26} = \{(2,3),(3,1)\}$
$S_{27} = \{(2,3),(3,2)\}$
$S_{28} = \{(1,1),(2,2),(3,3)\}$
$S_{29} = \{(1,1),(2,3),(3,2)\}$
$S_{30} = \{(1,2),(2,1),(3,3)\}$
$S_{31} = \{(1,2),(2,3),(3,1)\}$
$S_{32} = \{(1,3),(2,1),(3,2)\}$
$S_{33} = \{(1,3),(2,1),(3,2)\}$.

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^1_{18} and y^3_8. With this set of variables, the ReLRMP is

$$
\begin{array}{llrrl}
\textbf{(ReLRMP)} & \min & +2y^2_5 + 10y^1_{13} & & \\
& \text{s.t} & +y^1_{13} & \geq 1 & i=1 \\
& & +y^2_5 & \geq 1 & i=2 \\
& & +y^1_{13} & \geq 1 & i=3 \\
& & +y^1_{13} & \geq 1 & k=1 \\
& & +y^2_5 & \geq 1 & k=2 \\
& & +y^1_{13} & \geq 1 & k=3 \\
& & -y^1_{13} & \geq -1 & j=1 \\
& & -y^2_5 & \geq -1 & j=2 \\
& & & \geq -1 & j=3 \\
& & -y^2_5 \; -y^1_{13} & \geq -2 & \\
& & -8y^2_5 \; -y^1_{13} & \geq -9 & k=2 \\
& & -2y^2_5 \; -3y^1_{13} & \geq -9 & k=3 \\
& & y & \geq 0. &
\end{array}
$$

Actually, we are interested in its dual problem:

$$
\begin{array}{lll}
\textbf{(DP)} & \max & +\alpha_1 +\alpha_2 +\alpha_3 +\beta_1 +\beta_2 +\beta_3 -\gamma_1 -\gamma_2 -\gamma_3 \\
& & -2\delta -9\epsilon_2 -9\epsilon_3 \\
& \text{s.t.} & +\alpha_2 +\beta_2 -\gamma_2 -\delta -8\epsilon_2 -2\epsilon_3 \leq 2 \quad \left(y^2_5\right) \\
& & +\alpha_1 +\alpha_3 +\beta_1 +\beta_3 -\gamma_1 -\delta -\epsilon_2 -3\epsilon_3 \leq 10 \\
& & \qquad\qquad\qquad\qquad\qquad\qquad \left(y^1_{13}\right) \\
& & \alpha, \beta, \gamma, \delta, \epsilon \geq 0.
\end{array}
$$

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^j_S with negative reduced cost may be added to ReLRMP, we 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.

To do that, for each facility $j \in I$, we solve a subproblem to find the column y^j_S, $S \in \mathcal{S}(j)$, 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 if its couples are compatible. We solve this problem by the following dynamic programming algorithm.

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

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

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

$$
\min_{S \in \mathcal{S}(j)} \bar{c}^j_S = \gamma^*_j + \delta^* + \sum_{(i,k) \in S} d^k_{ij}. \tag{28}
$$

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

$$
D_j = \begin{pmatrix}
d^1_{i_1 j} & d^2_{i_1 j} & \cdots & d^n_{i_1 j} \\
d^1_{i_2 j} & & & \\
\vdots & & \ddots & \\
d^1_{i_n j} & & & d^n_{i_n j}
\end{pmatrix}, \tag{29}
$$

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^k_{ij} for all $i, k = 1, \ldots, n$ of the matrix D_1 ($j = 1$).

$$
\begin{aligned}
d^1_{11} &= \lambda^1 c_{11} + r_{11}\epsilon_2 - \alpha_1 - \beta_1 = 4 \\
d^2_{11} &= \lambda^2 c_{11} + \left(n^2 - r_{11} + 1\right)\epsilon_2 + r_{11}\epsilon_3 - \alpha_1 - \beta_2 = 2 \\
d^3_{11} &= \lambda^3 c_{11} + +\left(n^2 - r_{11} + 1\right)\epsilon_3 - \alpha_1 - \beta_3 = -9 \\
d^1_{21} &= \lambda^1 c_{21} + r_{21}\epsilon_2 - \alpha_2 - \beta_1 = 10 \\
d^2_{21} &= \lambda^2 c_{21} + \left(n^2 - r_{21} + 1\right)\epsilon_2 + r_{21}\epsilon_3 - \alpha_2 - \beta_2 = 4 \\
d^3_{21} &= \lambda^3 c_{21} + +\left(n^2 - r_{21} + 1\right)\epsilon_3 - \alpha_2 - \beta_3 = -9 \\
d^1_{31} &= \lambda^1 c_{31} + r_{31}\epsilon_2 - \alpha_3 - \beta_1 = 24 \\
d^2_{31} &= \lambda^2 c_{31} + \left(n^2 - r_{31} + 1\right)\epsilon_2 + r_{21}\epsilon_3 - \alpha_3 - \beta_2 = 12 \\
d^3_{31} &= \lambda^3 c_{31} + +\left(n^2 - r_{31} + 1\right)\epsilon_3 - \alpha_3 - \beta_3 = -4
\end{aligned}
$$

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

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

By using D_j, we obtain that a set S belongs to $\mathcal{S}(j)$ if and only if 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

$$g^j(i_l, k) = \min\left\{\bar{c}^j_S : S \in \mathcal{S}(j) \text{ and for all } (i'_l, k') \in S : i'_l \le i_l \text{ and } k' \le k\right\} \tag{30}$$

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 (position k), at most one couple containing i (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 $g^j(i_l, k) = g^j(i_{l-1}, k-1) + d^k_{i_l j}$. Otherwise, $S^j(i_l, k)$ may contain a couple (i_l, k') with $k' \le k - 1$ or a couple $(i_{l'}, k)$ with $l' \le l - 1$ but not both; 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 $l, k = 2, \ldots, n$:

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

Algorithm 1 (Pricing Subproblem Algorithm)

$g^j(i_1, 1) = \min\{0, d^1_{i_1 j}\}$;
if $g^j(i_1, 1) = d^1_{i_1 j} < 0$, **then**
  $S^j(i_1, 1) = \{(i_1, 1)\}$;
else
  $S^j(i_1, 1) = \emptyset$;
end if
for $k = 2, \ldots, n$, **do**
  $g^j(i_1, k) = \min\{d^k_{i_1 j}, g^j(i_1, k-1)\}$;
  if $g^j(i_1, k) = g^j(i_1, k-1)$, **then**
    $S^j(i_1, k) = S^j(i_1, k-1)$;
  else
    $S^j(i_1, k) = \{(i_1, k)\}$;
  end if
end for
for $l = 2, \ldots, n$, **do**
  $g^j(i_l, 1) = \min\{d^1_{i_l j}, g^j(i_{l-1}, 1)\}$;
  if $g^j(i_l, 1) = g^j(i_{l-1}, 1)$, **then**
    $S^j(i_l, 1) = S^j(i_{l-1}, k)$;
  else
    $S^j(i_l, 1) = \{(i_l, 1)\}$;
  end if
end for
for $k, l = 2, \ldots, n$, **do**
  $g^j(i_l, k) = \min\{g^j(i_{l-1}, k-1) + d^k_{i_l j}, g^j(i_{l-1}, k-1), g^j(i_l, k-1), g^j(i_{l-1}, k)\}$;
  if $g^j(i_l, k) = g^j(i_{l-1}, k-1)$, **then**
    $S^j(i_l, k) = S^j(i_{l-1}, k-1)$;
  else if $g^j(i_l, k) = g^j(i_l, k-1)$, **then**
    $S^j(i_l, k) = S^j(i_l, k-1)$;
  else if $g^j(i_l, k) = g^j(i_{l-1}, k)$, **then**
    $S^j(i_l, k) = S^j(i_{l-1}, k)$;
  else
    $S^j(i_l, k) = S^j(i_{l-1}, k-1) \cup \{(i_l, k)\}$;
  end if
end for

Obviously, if, at the end of the procedure, $g^j(i_n, n) + \delta + \gamma_j$ is negative, the variable $y^j_{S^j(i_n, n)}$ 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 $\bar{c}^j_R = \min_S \bar{c}^j_S$, and if $\bar{c}^j_R < 0$, we can activate (at least) y^j_R. Next, we solve a new 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 algorithm, the evaluation of each $g^j(i, k)$ is done in constant time. Solving the pricing subproblem amounts to evaluating $g^j(i_n, n)$ 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 $j = 1$.

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

We have obtained $g^1(i_3,3) = -9$, and $S^1(i_3,3) = S_3$ is the potential set to be used because 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.

Next, the process continues with the following facilities, that is, $j = 2, 3$. In this example, the optimal solution can be certified after four complete iterations of the preceding 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 especially 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 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. Because we are only interested in recovering feasibility in ReLRMP, one can proceed in the same way as for the usual pricing but with null coefficients in the objective function of the primal. In this way, the Farkas dual problem is

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

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

$$\delta, \gamma_j \epsilon_k \geq 0 \quad j\in I, k\in K, \quad k\neq 1. \tag{34}$$

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 starting the column generation algorithm with an empty pool of columns although this is not advisable. Furthermore, Farkas pricing is 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 1,800 seconds. Those are the smallest instances that we 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* provides a promising pool of initial columns as well as a good upper bound.

GRASP (Feo and Resende 1989, 1995) is a well-known heuristic technique that usually exhibits good performance in short computing time. In our case, it consists of 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 of an interchange heuristic between open and closed facilities. The pseudocode of the GRASP used to solve the problem is described in Algorithm 2.

Algorithm 2 (GRASP for DOMP)

```
Input(n, p, C, λ, n1, n2, q);
for n1 replications, do
  PartialSolution ← ConstructRandomizedPartialSolution(q);
  Solution ← ConstructGreedySolution(PartialSolution);
  for n2 iterations, do
    Solution ← LocalSearch(Solution);
    BestSolution ← UpdateSolution(Solution, BestSolution);
  end for
end for
```

First of all, we would like to point out the remarkable behavior of the GRASP heuristic for this problem. To illustrate the appropriateness of our heuristic, we have solved to optimality a number of instances of the problem using the mixed integer programming (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.

Because we are solving the 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, that is, 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**

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)), percentage gap at termination, i.e., $100(z_{UB} - z_{LB})/z_{LB}$ (Gap(%)), and number of unsolved problems (in parentheses), number of nodes $|\text{Nodes}|$, and number of variables ($|Vars|$).

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 six 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 percentage 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 solving the considered instances faster, requiring a fewer 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 1,800 seconds for the 60 instances with sizes $n = 20, 30$. The dashed line reports results using GRASP and the solid 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

Table 1. CPU Time, Number of Nodes, and Number of Variables with and without GRASP Heuristic for $n = 20, 30$

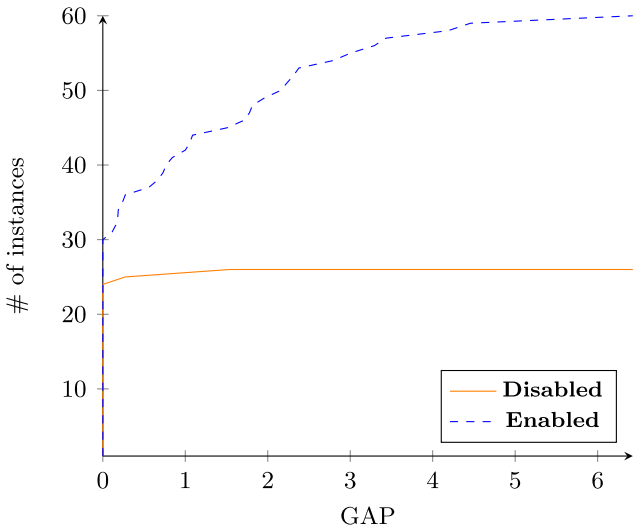
GRASP	Time(s)	Gap(%)	$\lvert\text{Nodes}\rvert$	$\lvert Vars\rvert$
Disabled	1,107.21	— (36)	158	12,850
Enabled	965.31	0.89 (30)	88	9,907

Table 2. CPU Time, Number of Nodes, and Number of Variables with and without GRASP Heuristic for $n = 20, 30$

GRASP	Time(s)	$\lvert\text{Nodes}\rvert$	$\lvert Vars\rvert$
Disabled	386.19	272	10,962
Enabled	147.77	79	6,062

Note. Summary of solved instances.

Figure 1. (Color online) Performance Profile Graph with GRASP Enabled or Disabled after 1,800 Seconds, *GAP*/# *of instances*



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 obtaining a feasible solution for only two 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 one iteration to the next, resulting in a slow convergence. For this reason, the stabilization is sometimes a critical step 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 of 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 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*)

$\Delta = \Delta_{init}$; $\overline{\pi} = 0$; $LB(\overline{\pi}) = 0$; $GAP = 1$;
while $GAP > \epsilon$, **do**
  Solve ReLRMP, obtaining z_{ReLRMP} and π_{ReLRMP};
    $\pi_{st} = \Delta\pi_{ReLRMP} + (1-\Delta)\overline{\pi}$;
  for $j = 1, \ldots, n$, **do**
    Solve the pricing using π_{st}, obtaining S;
    if $\overline{c}_S^j(\pi_{ReLRMP}) < 0$, **then** add variable y_S^j; **end if**
  end for
  $LB(\pi_{st}) = z(\pi_{st}^t) + \sum_{S,j:y_S^j added} \overline{c}_S^j(\pi_{st})$;
  if at least one variable was added, **then**
    if $LB(\pi_{st}) > LB(\overline{\pi})$, **then**
      $\overline{\pi} = \pi_{st}$; $LB(\overline{\pi}) = LB(\pi_{st})$;
    end if
  else
    $\overline{\pi} = \pi_{st}$; $LB(\overline{\pi}) = LB(\pi_{st})$;
  end if
  $GAP = \frac{z_{ReLRMP} - LB(\overline{\pi})}{z_{ReLRMP}}$;
  if $GAP < 1 - \Delta$, **then** $\Delta = 1 - GAP$; **end if**
end while

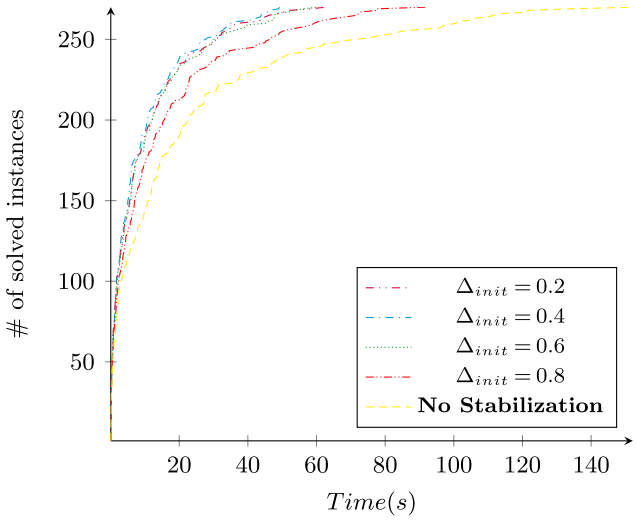
In words, the algorithm performs a while loop in which, in each iteration, it makes a convex combination of the current vector of dual multipliers and the best vector of multipliers found so far. This loop ends whenever both vectors of multipliers are close enough based on the gap between the incumbent 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 because $\Delta = 1 - GAP$ and GAP decreases with the number of iterations. Eventually, in the very last iterations of the stabilization algorithm, we use the actual vector of dual multipliers because $\pi_{st} \approx \pi_{ReLRMP}$.

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 $\Delta = 0.4$ (dash-dotted 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 Section 4.2.

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

Figure 2. (Color online) Performance Profile Graph with Different Combination of Δ_{init}, Time(s)/#*solved instances*



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 the number of variables should be small at every node.

3.3. Preprocessing

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 fixing 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 preceding results imply that those variables y_S^j such that $(i,k) \in S$ and $x_{ij}^k = 0$ are not 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, such 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 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 $j \in I, S \in \mathcal{S}(j)$.*

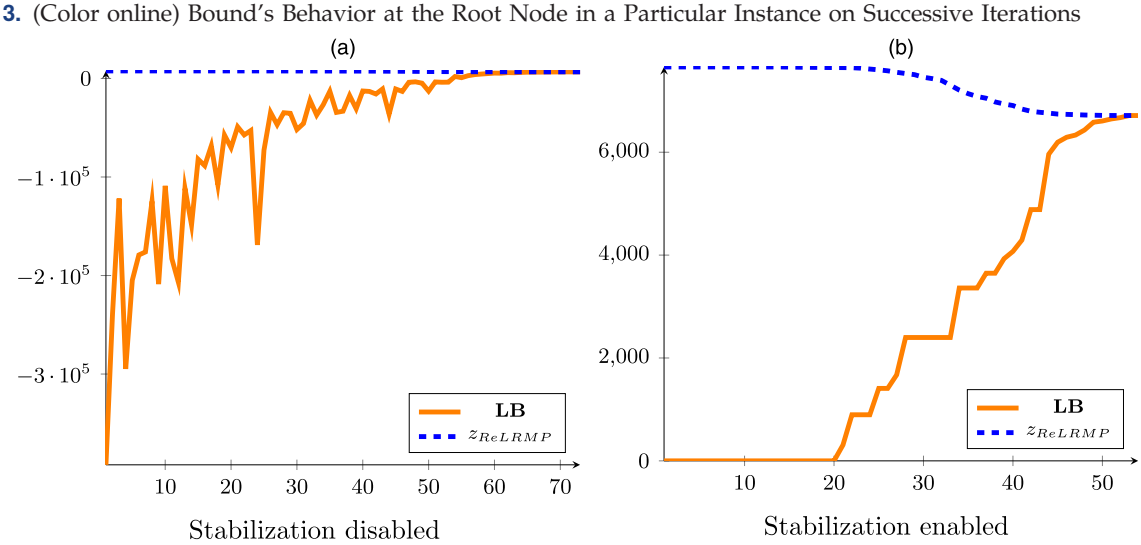
Proof. Suppose, on the contrary, there exists a variable with fractional value $y_{S'}^{j'}$. Because x_{ij}^k are binary for all i,j,k (in particular for i_1, j', k_1, where (i_1,k_1) is a pair of S'), there must be another fractional variable $y_{S''}^{j'}$ such that $(i_1,k_1) \in S''$.

Note that $S'' \neq S'$ because 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

Figure 3. (Color online) Bound's Behavior at the Root Node in a Particular Instance on Successive Iterations



$\sum_{S\in\mathcal{S}(j'):(i_2,k_2)\in S} y_S^{j'}$ 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 $x_{i_2k_2}^{j'}$ is not binary. □

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

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 that 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 on which to branch 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 resulting from 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.

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.

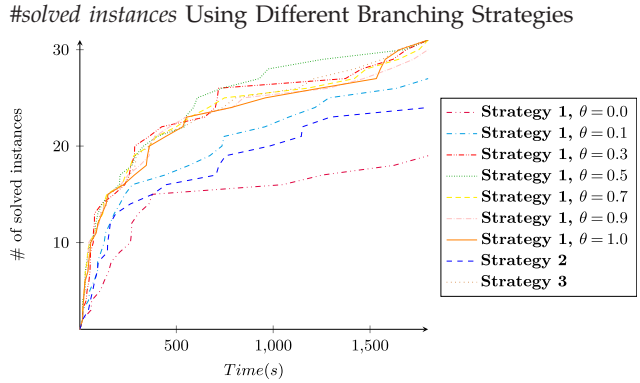
3.5. Valid Inequalities

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

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

Furthermore, solving the pricing subproblem to find a new column or to certify optimality of the column-generation algorithm requires adapting the dynamic programming algorithm that computes the $g(i_l, k)$ terms using the new dual multipliers. This implies

Figure 4. (Color online) Performance Profile Graph of *#solved instances* Using Different Branching Strategies



modifying the D_j matrices. Once again, after some calculations, the modified d_{ij}^k elements are now given by

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

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.

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 as follows. 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 4 GB of RAM. Each implementation has a maximum of 7,200 seconds (two 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

Because 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 refer to 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 instance 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.

Table 3. Numerical Results with and Without Cuts

		$n = 50$			$n = 60$		
		$p = 12$	$p = 16$	$p = 25$	$p = 15$	$p = 20$	$p = 30$
B&P (MP)	*Time*(s)	7,200.03	7,200.01	7,200.01	7,200.07	7,200.02	7,202.00
	|*Vars*|	48,648	38,368	21,558	49,990	34,262	24,730
	|*Nodes*|	4,828	11,768	49,523	5,329	12,526	30,305
	#unsolved	10	10	10	10	10	10
	Gap(%)	6.26	7.57	9.57	8.29	8.97	12.07
B&P&C (MP)	*Time*(s)	7,200.39	7,200.49	6,860.93	7,200.08	7,200.77	7,200.20
	|*Vars*|	14,807	14,977	13,407	17,131	16,691	16,838
	|*Nodes*|	1	6	9	2	1	11
	|*Cuts*|	3,526	3,066	2,709	3,489	4,192	2,864
	#unsolved	10	10	9	10	10	10
	Gap(%)	2.83	2.89	1.87	5.04	4.52	4.47

Table 4. Numerical Results on Linear Relaxation for *WOC* and *MP*

		(WOC)				(MP)			
n	p	*GapLP*(%)	*Time*(*s*)	|*Vars*|	*Memory*(*MB*)	*GapLP*(%)	*Time*(*s*)	|*Vars*|	*Memory*(*MB*)
20	10	15.35	0.12	8,020	35	14.78	0.09	450	2
30	15	16.77	0.62	27,030	101	16.16	0.29	1,067	5
40	20	18.22	2.09	64,040	235	17.98	0.74	2,005	10
50	25	15.02	6.25	125,050	451	14.74	1.36	2,922	15
60	30	16.94	12.27	216,060	764	16.69	2.74	4,777	23
70	35	15.97	29.40	343,070	1,214	15.80	4.61	6,327	30
80	40	7.67	47.63	512,080	1,830	7.63	5.63	7,161	34
90	45	7.19	82.19	729,090	2,561	7.15	9.14	8,867	42
100	50	–	–	1,000,100	>4,096	7.05	14.18	11,255	54
20	6	9.56	0.14	8,020	35	8.79	0.14	548	3
30	10	11.68	0.68	27,030	101	10.96	0.39	1,257	7
40	13	12.39	2.34	64,040	235	12.08	1.17	2,371	14
50	16	9.65	6.37	125,050	451	9.34	2.09	3,641	21
60	20	11.11	13.30	216,060	764	10.82	4.18	5,652	32
70	23	10.16	35.65	343,070	1,214	9.98	7.54	7,684	44
80	26	8.32	58.14	512,080	1,830	8.11	11.32	9,699	55
90	30	7.08	96.74	729,090	2,561	7.02	17.58	12,185	69
100	33	–	–	1,000,100	>4,096	8.01	28.76	15,656	87
20	5	9.53	0.14	8,020	35	8.61	0.16	574	4
30	7	10.61	0.70	27,030	101	9.36	0.59	1,278	9
40	10	10.44	2.51	64,040	235	10.05	1.61	2,536	17
50	12	7.96	7.35	125,050	451	7.54	3.01	3,955	27
60	15	9.74	15.98	216,060	764	9.41	5.56	5,697	36
70	17	8.80	40.78	343,070	1,214	8.50	10.19	7,828	53
80	20	9.53	67.42	512,080	1,830	9.38	17.38	10,740	69
90	22	9.58	128.70	729,090	2,561	9.41	28.66	13,581	86
100	25	–	–	1,000,100	>4,096	8.65	44.20	17,084	106

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.

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 the *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

Figure 5. (Color online) Graph of Number of Variables vs. Size n for *WOC* and *MP*

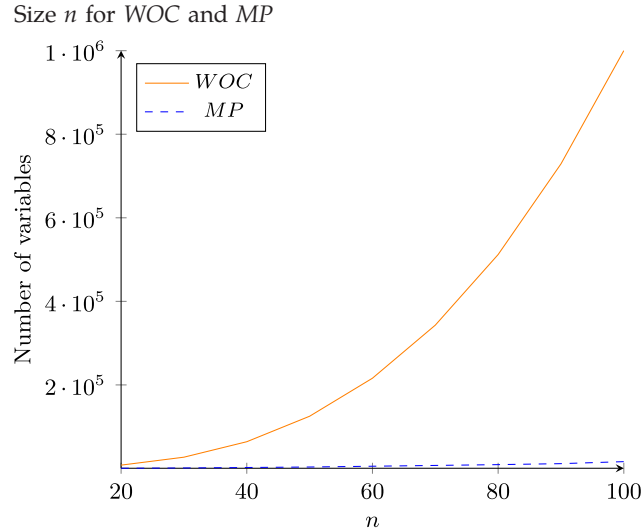
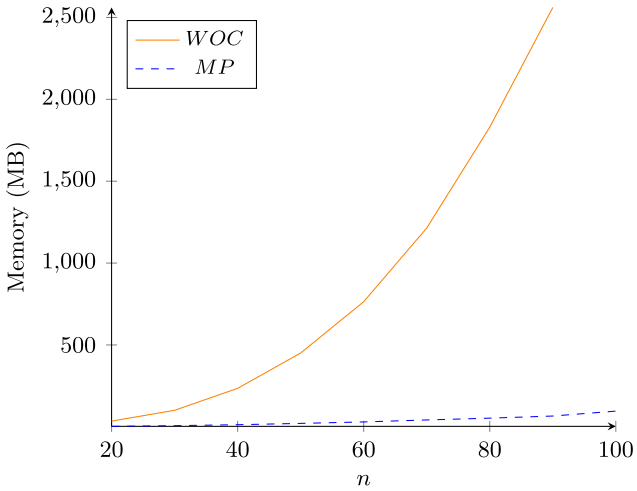


Figure 6. (Color online) Graph of Memory Usage (MB) vs. Size n for *WOC* and *MP*



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).

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 because the memory limit is never reached, and instances may be not solved only because of the 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-sized 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 7,200 seconds are always smaller than 8% for B&C(WOC) and smaller than 6% for B&P&C(MP), the latter being clearly better from instances of $n = 70$. Because 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.

Table 6 contains the results within the time limit of two hours for bigger instances of DOMP. This table has the same layout as Table 5 except that we replace

Table 5. Numerical Results for B&C(WOC) and B&P&C(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	4,211	38	689	0/0	0.00	58.70	4,272	111	373	0	0.00
30	15	131.89	13,952	19,197	2,519	0/0	0.00	3,670.21	16,369	392	1,078	3	0.45
40	20	6,202.85	32,820	605,812	4,727	8/0	2.54	6,751.27	14,591	41	1,819	9	3.33
50	25	6,575.59	63,776	355,560	10,131	9/0	1.35	6,860.93	13,407	9	2,709	9	1.87
60	30	6,707.38	109,804	85,723	15,676	8/2	1.82	7,200.20	16,838	11	2,864	10	4.47
70	35	2,474.98	173,955	835	19,238	2/8	7.72	7,200.31	17,758	2	4,150	10	4.95
80	40	3,428.13	259,186	1	12,406	0/10	3.38	7,201.36	18,902	2	4,680	10	2.13
90	45	6,243.49	368,560	1	12,157	7/3	4.24	7,200.38	20,028	7	4,127	10	2.37
20	6	11.50	5,706	440	1,249	0/0	0.00	952.76	10,959	97	615	0	0.00
30	10	1,578.21	18,245	305,595	3,056	1/0	0.12	6,270.41	17,502	81	1,503	8	1.61
40	13	7,061.36	43,664	628,962	6,559	8/2	2.38	7,200.72	11,186	2	3,073	10	3.69
50	16	7,116.54	85,630	284,028	10,423	9/1	1.14	7,200.49	14,977	6	3,066	10	2.89
60	20	3,306.54	144,983	20,330	19,887	2/8	3.02	7,200.77	16,691	1	4,192	10	4.52
70	23	2,119.13	231,680	1	23,603	0/10	6.20	7,200.97	19,307	2	4,365	10	4.51
80	26	2,886.25	346,926	1	25,187	0/10	5.59	7,201.35	21,675	1	5,449	10	3.69
90	30	5,214.89	488,316	1	32,406	0/10	4.62	7,201.66	24,507	1	6,116	10	3.17
20	5	16.54	6,054	1,215	1,537	0/0	0.00	1,989.43	14,699	97	731	1	0.09
30	7	1,807.41	20,643	198,424	4,789	1/1	0.65	6,840.10	14,934	29	2,093	8	1.85
40	10	7,050.93	48,065	602,685	7,939	7/3	1.68	7,200.90	10,730	1	3,455	10	3.95
50	12	7,200.00	94,784	270,959	12,579	10/0	0.91	7,201.39	14,807	1	3,526	10	2.83
60	15	2,768.88	161,807	1	18,081	0/8	2.90	7,201.08	17,131	2	3,489	10	5.04
70	17	1,842.00	259,406	1	16,115	0/10	6.04	7,201.78	19,454	1	4,649	10	4.51
80	20	2,902.00	383,199	1	27,129	0/10	6.95	7,201.52	25,278	3	4,320	10	4.89
90	22	5,999.16	549,561	1	46,216	0/10	6.82	7,201.75	27,418	1	5,549	10	5.50

Table 6. Numerical Results for B&P&C(MP) for Bigger Instances

n	p	*Time*(*s*)	|*Vars*|	|*Nodes*|	|*Cuts*|	*Memory*(*MB*)	*Gap*(%)
100	50	7,201.10	23,057	2	5,221	311	2.43
120	60	7,200.54	29,209	1	6,992	424	2.54
140	70	7,202.11	34,222	2	6,805	449	2.87
160	80	7,201.08	41,811	1	8,069	574	3.13
180	90	7,201.24	50,523	1	7,961	656	3.48
200	100	7,201.61	59,931	1	9,050	805	3.96
220	110	7,201.39	57,685	1	11,097	806	4.39
240	120	7,200.94	63,710	1	11,487	874	4.43
260	130	7,201.83	73,143	1	9,772	910	5.05
280	140	7,200.98	83,892	1	9,592	1,037	6.19
300	150	7,201.74	87,444	1	10,811	1,076	6.83
100	33	7,201.10	29,974	2	5,864	562	3.81
120	40	7,201.12	37,204	5	5,433	621	4.80
140	46	7,200.98	48,286	2	7,953	894	4.47
160	53	7,201.55	57,060	3	7,159	926	5.01
180	60	7,203.85	70,077	1	8,997	1,188	5.99
200	66	7,200.55	67,417	1	11,416	1,199	6.16
220	73	7,201.41	71,731	2	9,356	1,073	6.88
240	80	7,202.02	84,466	2	10,093	1,324	9.80
260	86	7,200.74	92,401	1	10,505	1,390	9.54
280	93	7,201.68	105,535	1	13,265	1,728	9.01
300	100	7,200.28	114,427	1	15,595	2,034	9.41
100	25	7,202.69	31,218	1	7,034	801	4.73
120	30	7,200.98	40,074	1	6,818	922	5.91
140	35	7,203.38	49,539	1	8,466	1,162	5.83
160	40	7,201.35	58,464	1	10,688	1,465	7.29
180	45	7,203.43	72,146	1	10,329	1,674	8.70
200	50	7,201.88	62,568	1	10,252	1,364	11.27
220	55	7,203.56	74,359	1	9,985	1,515	10.34
240	60	7,200.19	80,228	1	9,745	1,422	11.57
260	65	7,200.54	99,628	1	7,944	1,346	11.77
280	70	7,203.63	109,544	1	4,187	1,247	11.32
300	75	7,200.79	128,462	1	3,844	1,261	12.26
400	200	86,400.42	158,287	1	9,308	1,183	7.07
400	133	86,401.23	178,265	1	13,652	2,764	11.26
400	100	86,401.11	236,973	1	7,582	3,125	10.29

Note. For instances with n = 400, the time limit was set to 24 hours.

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 of $n \geq 70$. 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.

Acknowledgments

The authors thank the SCIP team (Gamrath et al. 2016) for the helpful advice. They also thank the helpful reports from two anonymous referees that led to improving the quality of the paper.

References

Achterberg T, Koch T, Martin A (2005) Branching rules revisited. *Oper. Res. Lett.* 33(1):42–54.

Applegate D, Bixby R, Chvátal V, Cook W (1995) Finding cuts in the TSP (a preliminary report). DIMACS Technical Report 95-05, DIMACS, Rutgers University, New Brunswick, NJ.

Avella P, Sassano A, Vasil'ev I (2006) Computational study of large-scale p-median problems. *Math. Programming* 109(1):89–114.

Barnhart C, Johnson E, Nemhauser G, Savelsbergh M, Vance P (1998) Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.* 46(3):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. *Proc. 5th Internat. Conf. Oper. Res.* (Tavistock Publications, London), 447–454.

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

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

Ceselli A, Righini G (2005) A branch-and-price algorithm for the capacitated *p*-median problem. *Networks* 45(3):125–142.

Ceselli A, Liberatore F, Righini G (2008) A computational evaluation of a general branch-and-price framework for capacitated network location problems. *Ann. Oper. Res.* 167(1):209–251.

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

Contreras I, Díaz J, Fernández E (2011) Branch and price for large-scale capacitated hub location problems with single assignment. *INFORMS J. Comput.* 23(1):41–55.

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. Preprint, submitted February 9, https://arxiv.org/abs/1802.03191.

Desrosiers J, Lübecke M (2005) A primer in column generation. Desaulniers G, Desrosiers J, Salomon MM, eds. *Column Generation* (Springer, Boston), 1–32.

Domínguez-Marín P, Nickel S, Hansen P, Mladenovic N (2005) Heuristic procedures for solving the discrete ordered median problem. *Ann. Oper. Res.* 136(1):145–173.

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

du Merle O, Vial J (2002) Proximal ACCPM, a cutting plane method for column generation and Lagrangean relaxation: Application to the *p*-median problem. Technical report, HEC Genève, Université de Genève, Geneva, Switzerland.

du Merle O, Villenueve D, Desrosiers J, Hansen P (1999) Stabilized column generation. *Discrete Math.* 194(1–3):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. *Oper. Res. Lett.* 8(2):67–71.

Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. *J. Global Optim.* 6(2):109–133.

Fernández E, Pozo MA, Puerto J (2014) Ordered weighted average combinatorial optimization: Formulations and their properties. *Discrete Appl. Math.* 169(31):97–118.

Fernández E, Puerto J, Rodríguez-Chía AM (2013) On discrete optimization with ordering. *Ann. Oper. Res.* 207(1):83–96.

Fernández E, Pozo MA, Puerto J, Scozzari A (2017) Ordered weighted average optimization in multiobjective spanning tree problems. *Eur. J. Oper. Res.* 260(31):886–903.

Gamrath G (2010) Generic branch-cut-and-price. Master's thesis, Institut für Mathematik, Technische Universität Berlin, Berlin.

Gamrath G, Fischer T, Gally T, Gleixner AM, Hendel G, Koch T, Maher SJ, Miltenberger M, Müller B, Pfetsch ME, et al. (2016) The SCIP optimization suite 3.2. Technical report 15-60, ZIB, Berlin.

Günlük O, Ladányi L, de Vries S (2005) A branch-and-price algorithm and new test problems for spectrum auctions. *Management Sci.* 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*, NATO ASI Series, vol. 51 (Springer, Berlin Heidelberg), 1–43.

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

Lorena L, Senne E (2004) A column generation approach to capacitated p-median problems. *Comput. Oper. Res.* 31(6):863–876.

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

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

Nickel S (2001) Discrete ordered Weber problems. *Oper. Res. Proc.* 2000: 71–76.

Nickel S, Puerto J (1999) A unified approach to network location problems. *Networks* 34(4):283–290.

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

Olender P, Ogryczak W (2018) A revised variable neighborhood search for the discrete ordered median problem. *Eur. J. Oper. Res.* 274(2):445–465.

Perea F, Puerto J (2013) Finding the nucleolus of any *n*–person cooperative game by a single linear program. *Comput. Oper. Res.* 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. *Math. Programming Comput.* 2(3–4): 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. *Comput. Oper. Res.* 98:127–136.

Puerto J (2008) A new formulation of the capacitated discrete ordered median problem with {0, 1}-assignment. Kalcsics J, Nickel S, eds. *Oper. Res. Proc. 2007* (Springer, Berlin Heidelberg), 165–170.

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

Puerto J, Rodríguez-Chía AM (2015) Ordered median location problems. Laporte G, Nickel S, Saldanha da Gama F, eds. *Location Science* (Springer) Heidelberg: 249–288.

Puerto J, Pérez-Brito D, García-González CG (2014) A modified variable neighborhood search for the discrete ordered median problem. *Eur. J. Oper. Res.* 234(1):61–76.

Puerto J, Ramos AB, Rodríguez-Chía AM (2011) Single-allocation ordered median hub location problems. *Comput. Oper. Res.* 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 Appl. Math.* 161(16–17):2624–2646.

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

Puerto J, Ramos AB, Rodríguez-Chía AM, Sánchez-Gil MC (2016) Ordered median hub location problems with capacity constraints. *Transportation Res., Part C Emerging Tech.* 70:142–156.

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* (North-Holland, Amsterdam), 269–280.

Senne E, Lorena L, Pereira MA (2005) A branch-and-price approach to *p*-median location problems. *Comput. Oper. Res.* 32(6):1655–1664.

Stanimirovic Z, Kratica J, Dugosija D (2007) Genetic algorithms for solving the discrete ordered median problem. *Eur. J. Oper. Res.* 182(3):983–1001.

Wolsey LA (1998) *Integer Programming* (John Wiley & Sons, New York).