



# Constraint relaxation for the discrete ordered median problem

Luisa I. Martínez-Merino<sup>1,2</sup> · Diego Ponce<sup>2,3</sup>  · Justo Puerto<sup>2,3</sup>

Received: 18 July 2022 / Accepted: 17 November 2022  
© The Author(s) 2022

## Abstract

This paper compares different exact approaches to solve the Discrete Ordered Median Problem (DOMP). In recent years, DOMP has been formulated using set packing constraints giving rise to one of its most promising formulations. The use of this family of constraints, known as *strong order constraints* (SOC), has been validated in the literature by its theoretical properties and because their linear relaxation provides very good lower bounds. Furthermore, embedded in branch-and-cut or branch-price-and-cut procedures as valid inequalities, they allow one to improve computational aspects of solution methods such as CPU time and use of memory. In spite of that, the above mentioned formulations require to include another family of order constraints, e.g., the *weak order constraints* (WOC), which leads to coefficient matrices with elements other than  $\{0,1\}$ . In this work, we develop a new approach that does not consider extra families of order constraints and furthermore relaxes SOC -in a branch-and-cut procedure that does not start with a complete formulation- to add them iteratively using row generation techniques to certify feasibility and optimality. Exhaustive computational experiments show that it is advisable to use row generation techniques in order to only consider  $\{0,1\}$ -coefficient matrices modeling the DOMP. Moreover, we test how to exploit the problem structure. Implementing an efficient separation of SOC using callbacks improves the solution performance. This allows us to deal with bigger instances than using fixed cuts/constraints pools automatically added by the solver in the branch-and-cut for SOC, concerning both the formulation based on WOC and the row generation procedure.

**Keywords** Discrete ordered median problem · Branch-and-cut · Constraint relaxation · Row generation

**Mathematics Subject Classification** 90-04 · 90-08 · 90B80 · 90C10 · 90C57

---

Luisa I. Martínez-Merino, Diego Ponce, and Justo Puerto contributed equally to this work.

---

✉ Diego Ponce  
dponce@us.es

Extended author information available on the last page of the article

## 1 Introduction

At times, very hard combinatorial optimization problems contain easy combinatorial subproblems after relaxing some of their constraints. A paradigmatic example is the Traveling Salesman Problem: after the elimination of its subtour elimination constraints it turns into the Linear Assignment Problem which is polynomially solvable. This pattern calls for developing techniques that take advantage of this situation to solve some combinatorial problems based on their constraint relaxation. This approach is not new and the reader is referred to Focacci et al. (1999, 2002a, 2002b) and the references therein for further details.

This behavior is not only observed in problems where formulations include an exponential number of constraints. Actually, it also occurs in many polynomial size formulations. One of these cases is the Discrete Ordered Median Problem (DOMP) modeled with the strong order constraints formulation as introduced in Labbé et al. (2017). If one removes the family of strong order constraints, whose acronym is SOC, the resulting problem is the standard  $p$ -median problem that is known to be combinatorially friendly (Hakimi 1964; Marín and Pelegrín 2019; ReVelle and Swain 1970). The aim of this work is to develop solution techniques for DOMP based on constraint relaxations.

The DOMP is a discrete location model that allows to generalize several classical discrete location problems (see, e.g., Nickel and Puerto 2005). For instance, the discrete  $p$ -center and  $p$ -median are particular cases of DOMP. Assume that we are given a set of clients, a set of candidate locations for facilities, and the allocation costs from each candidate facility to each client. The objective of DOMP is to locate  $p$  facilities in such a way that a certain weighted function of the allocation costs is minimized. These weights are not assigned to specific costs but to their sorted values. Namely, the weighted average sorts the allocation costs in a nondecreasing order and then, it performs the scalar product of this so obtained sorted cost vector by the vector of weights.

In the literature, one can find different applications of the ordered median operator. For instance, it has been applied to facility location (Aouad and Segev 2019; Domínguez and Marín 2020; Espejo et al. 2009; Kalcsics et al. 2010; Martínez-Merino et al. 2017; Tamir 2001), multicast communication (Fourour and Lebbah 2020), multiobjective Markov decision processes (Ogryczak et al. 2011), voting problems (Ponce et al. 2018), supervised classification (Marín et al. 2022), tomography reconstruction (Calvino et al. 2022), and network design (Puerto et al. 2013), among other situations.

DOMP was first introduced in Nickel (2001) as an integer nonlinear problem. Then, in Boland et al. (2006), this problem was modeled as a mixed integer linear program. Some works on this problem take advantage of some particular characteristics. Specifically, Marín et al. (2009) introduce an efficient covering formulation for DOMP considering free self-service, ties in the cost matrix and a non-negative weighted order vectors in the objective function. Furthermore, Marín et al. (2010) present a covering reformulation for weighted order vectors containing zeros and an extended model for vectors even with negative elements.

In Labbé et al. (2017), a new three-index formulation based on set packing constraints is proposed. These set packing constraints are known as *strong order constraints* or SOC, and the number of these constraints appearing in the formulation is  $\mathcal{O}(n^3)$ . In addition, another new formulation, solved by an efficient branch-and-cut procedure that provides good results in terms of time, is introduced.

This second formulation is based on the aggregation of the SOC corresponding to the same position. The resulting order constraints are the *weak order constraints* (from now on WOC). This formulation includes SOC as valid inequalities. Both formulations present small integrality gaps.

Recently, in Deleplanque et al. (2020), a novel branch-price-and-cut algorithm has been proposed. This procedure is based on a formulation with an exponential number of variables that corresponds to a set partitioning model. The proposed approach allows to handle larger instances since it requires less memory to run the model.

In this paper, we want to explore different exact approaches to solve DOMP. The first one uses branch-and-cut techniques based on one of the most promising formulations, namely the formulation based on WOC proposed in Labbé et al. (2017), adding SOC as valid inequalities. Additionally, we compare the use of cut pools in the branch-and-cut with respect to the use of callbacks to implement an *ad hoc* separator proposed in Labbé et al. (2017). By setting up pools of cuts, all SOC are initially stored and then, solvers decide which cuts are included during the branch-and-cut process. In contrast, applying the callbacks using the separation algorithm introduced in Labbé et al. (2017), SOC are not initially stored and the implementation of the SOC separation is based on a sequential update of the left-hand side of the corresponding order constraints. This separation can be performed in  $\mathcal{O}(n^3)$ .

The second method is based on a constraint relaxation on the formulation using SOC to model the order. This procedure, to solve DOMP, starts with a relaxed formulation where all SOC are removed and feasibility is enforced adding model constraints from the SOC family in the searching tree. Although the number of SOC is polynomial,  $\mathcal{O}(n^3)$ , this number of constraints becomes too large to be handled when  $n$  increases. Consequently, it is interesting to study this approach since we could improve the time and memory performance by only including the necessary constraints in the solution process. Again, in this case, we compare the branch-and-cut procedure through callbacks with respect to the branch-and-cut based on constraint pools.

The contributions of this paper can be summarized as follows:

- (1) Comparing a branch-and-cut approach to solve DOMP based on the so called WOC formulation with a constraint relaxation approach for DOMP based on removing SOC.
- (2) Comparing the performance of the branch-and-cut and the constraint relaxation approach when using callbacks based on specific tailor made separation oracles with respect to the use of fixed pools of cuts/constraints.
- (3) Reporting intensive computational tests which show the limits of the different considered solution methods.

The remainder of this work is organized as follows. In Sect. 2, we introduce the notation and description of DOMP. Besides, we recall two formulations for DOMP that will be used along the paper (DOMP<sub>WOC</sub> and DOMP<sub>SOC</sub>). In Sect. 3, we present two solution methods for the problem. First, we describe the branch-and-cut procedure for DOMP<sub>WOC</sub> introduced in Labbé et al. (2017). Then, we propose a novel row generation procedure for DOMP<sub>SOC</sub>. Section 4 is devoted to the analysis of both solution methods. In addition, we present a comparison between the results of using pools of cuts/constraints and using callbacks for the branch-and-cut and the row generation techniques. Finally, in Sect. 5, we include some conclusions and future research lines.

## 2 Problem definition and formulations

This section is devoted to recall the definition and some formulations of DOMP that will be instrumental in our discussion. We shall follow the following notation. We denote by  $I = \{1, \dots, n\}$  the set of  $n$  clients and, at the same time and without loss of generality, the set of  $n$  potential facility locations. Facilities are assumed to be uncapacitated, i.e., they can supply as many clients as desired. Besides,  $c_{ij}$  denotes the cost for serving client  $i$  from facility  $j$ , for  $i, j \in I$ .

Given a set  $J$  composed by  $p$  open facilities,  $c_i(J)$  represents the cost of allocating client  $i$  to the facility set  $J$ , i.e.,  $c_i(J) = \min_{j \in J} c_{ij}$ . In addition, if the vector of costs  $c_i(J)_{i=1, \dots, n}$  for  $J \subset I$  is sorted in non-decreasing order, we denote by  $c^{(k)}(J)$  the allocation cost in position  $k \in K = I$  of this sorted vector. Thus, it holds that  $c^{(1)}(J) \leq c^{(2)}(J) \leq \dots \leq c^{(n)}(J)$ .

The aim of DOMP is to determine a subset of  $p$  facilities  $J \subset I$  to open, and to assign each client to an open facility in order to minimize the ordered median objective function. Given a vector  $\lambda = (\lambda^k)_{k \in K}$  such that  $\lambda^k \geq 0$ ,  $k \in K$ , the objective function of DOMP can be expressed as  $\sum_{k \in K} \lambda^k c^{(k)}(J)$ . Consequently, the definition of DOMP is

$$\min_{J \subset I: |J|=p} \sum_{k \in K} \lambda^k c^{(k)}(J) \quad (\text{DOMP}).$$

Observe that this formulation generalizes several standard discrete location problems. For instance: if  $\lambda^1 = \lambda^2 = \dots = \lambda^n = 1$ , this model is the  $p$ -median problem; if  $\lambda^1 = \lambda^2 = \lambda^{n-1} = 0$ ,  $\lambda^n = 1$ , one gets the  $p$ -center; if  $\lambda^1 = \lambda^2 = \dots = \lambda^{n-k} = 0$ ,  $\lambda^{n-k+1} = \dots = \lambda^n = 1$ , the resulting problem is the  $k$ -centrum; etc.

DOMP is known to be  $\mathcal{NP}$ -complete, see Nickel and Puerto (2005), and as mentioned in the introduction, different formulations have been proposed to deal with this problem. In this paper, we will elaborate on two of the most recent and promising formulations presented in Labbé et al. (2017). On the one hand, that paper introduces a three-index formulation where order is modeled by a family of set packing constraints, SOC. On the other hand, it also presents an aggregated version of that formulation where the order is ensured by a different set of constraints called WOC.

We will build on those two formulations, thus in order to be self-contained, in the following subsection we provide full details of them.

### 2.1 Strong order constraints formulation

For the formulation based on *strong order constraints*, the next families of variables are required:

$$y_j = \begin{cases} 1, & \text{if facility } j \text{ is open,} \\ 0, & \text{otherwise,} \end{cases} \quad \text{for } j \in I,$$

$$x_{ij}^k = \begin{cases} 1, & \text{if client } i \text{ is allocated to facility } j \text{ and the} \\ & \text{associated cost is in position } k \text{ of the} \\ & \text{sorted sequence of allocation costs,} \\ 0, & \text{otherwise,} \end{cases} \quad \text{for } i, j \in I, k \in K.$$

Besides, we denote the rank of the allocation cost  $c_{ij}$  by  $r_{ij}$ , i.e.,  $r_{ij} = \ell$  if  $c_{ij}$  is the  $\ell$ -th element in the list of the costs  $c_{ij}$ , for all  $i, j \in I$ , sorted in a non decreasing sequence and where ties are broken arbitrarily. Then, DOMP<sub>SOC</sub> formulation is the following.

$$(\text{DOMP}_{\text{SOC}}) \min \sum_{i \in I} \sum_{j \in I} \sum_{k \in K} \lambda^k c_{ij} x_{ij}^k \tag{1}$$

$$\text{s.t. } \sum_{j \in I} \sum_{k \in K} x_{ij}^k = 1, \quad i \in I, \tag{2}$$

$$\sum_{i \in I} \sum_{j \in I} x_{ij}^k = 1, \quad k \in K, \tag{3}$$

$$\sum_{k \in K} x_{ij}^k \leq y_j, \quad i, j \in I, \tag{4}$$

$$\sum_{j \in I} y_j = p, \tag{5}$$

$$\sum_{i' \in I} \sum_{\substack{j' \in I : \\ r_{i'j'} \leq r_{ij}}} x_{i'j'}^k + \sum_{i' \in I} \sum_{\substack{j' \in I : \\ r_{i'j'} \geq r_{ij}}} x_{i'j'}^{k-1} \leq 1, \quad i, j \in I, k \in K, k \neq 1, \tag{SOC}$$

$$x_{ij}^k, y_j \in \{0, 1\}, \quad i, j \in I, k \in K. \tag{6}$$

Constraints (2) ensure that each client is served by just one facility in one position. Similarly, constraints (3) are necessary to guarantee that only one allocation cost is

in each sorted position. Constraints (4) ensure that each client is allocated to an open facility and that the allocation cost of a client can only be in at most one position. Constraint (5) restricts that exactly  $p$  facilities must be open. Constraints (SOC) are the so-called *strong order constraints* which ensure the correct sorting of the allocation costs. These constraints are set packing constraints, i.e., at most one of the variables of the l. h. s. could take value one. The incompatibility of two or more variables taking value one is due to (4) and the fact that a variable  $x_{ij}^k$  cannot take value one if  $x_{i'j'}^{k-1} = 1$  when  $r_{ij} < r_{i'j'}$ , for  $i, j, i', j' \in I, k \in K, k \neq 1$ . We refer the reader to Labbé et al. (2017) for a more detailed explanation. Finally, constraints (6) are the domain of definition of the variables. The reader may note that removing (SOC) the formulation results in the  $p$ -median.

### 2.2 Weak order constraints formulation

Despite the good mathematical properties of  $\text{DOMP}_{\text{SOC}}$ , as the lower bound given by its linear relaxation, the number of (SOC) is  $\mathcal{O}(n^3)$ . Consequently, when the number of clients increases, the number of (SOC) becomes too large to be handled by a solver. For this reason, Labbé et al. (2017) introduce an alternative family of constraints to ensure the order of costs. These new constraints are based on the aggregation of (SOC) corresponding to the same position. (The reader is referred to Labbé et al. (2017) for further details.) This alternative formulation results in the following.

$$\begin{aligned}
 (\text{DOMP}_{\text{WOC}}) \min & \sum_{i \in I} \sum_{j \in I} \sum_{k \in K} \lambda^k c_{ij} x_{ij}^k \\
 \text{s.t.} & (2) - (6), \\
 & \sum_{i \in I} \sum_{j \in I} \left( \sum_{i' \in I} \sum_{j' \in I : r_{i'j'} \leq r_{ij}} x_{i'j'}^k + \sum_{i' \in I} \sum_{j' \in I : r_{i'j'} \geq r_{ij}} x_{i'j'}^{k-1} \right) \leq n^2, k \in K, k \neq 1. \quad (\text{WOC})
 \end{aligned}$$

Constraints labeled by (WOC) are known as *weak order constraints*. They ensure that if facility  $j$  serves client  $i$  and its cost  $c_{ij}$  is in position  $k$  of the sorted cost vector of the solution, then there must be a smaller or equal allocation cost in position  $k - 1$ . This is due to the coefficients corresponding to each variable in the constraints. In each inequality, there are represented two positions ( $k - 1$  and  $k$ ). By constraints (3), only two variables must take value 1, and the remaining ones take value 0. Assuming that the variables with value one for position  $k$  and  $k - 1$  correspond to positions  $s$  and  $t$  of the sorted costs, respectively, the inequality can be expressed as follows:

$$(n^2 - (s - 1))x_{i_s j_s}^k + t x_{i_t j_t}^{k-1} \leq n^2,$$

with  $i_s, j_s, i_t, j_t \in I$  such that  $c_{i_s j_s}$  and  $c_{i_t j_t}$  are the  $s$ -th and  $t$ -th smallest allocation costs in matrix  $(c_{ij})_{n \times n}$ . This is valid if and only if  $t < s$ .

In Labbé et al. (2017), it is shown that  $\text{DOMP}_{\text{SOC}}$  formulation provides a relevant improvement of the integrality gap with respect to  $\text{DOMP}_{\text{WOC}}$  formulation. In other words, for most of the instances, fractional solutions satisfying (WOC) could be cut including (SOC). Thus, it is recommended to use (SOC) as valid inequalities of  $\text{DOMP}_{\text{WOC}}$ .

### 3 Solution methods

Both formulations,  $\text{DOMP}_{\text{SOC}}$  and  $\text{DOMP}_{\text{WOC}}$ , can be solved by using standard MIP solvers as CPLEX, Gurobi, Xpress, or SoPlex. However, the good performance of these formulations is rather limited for large sizes of the problem as we will see in Sect. 4. The reader should observe that already for  $n = 100$  clients the number of (SOC) is almost  $10^6$ .

To improve the performance of  $\text{DOMP}_{\text{WOC}}$ , Labbé et al. (2017) propose a branch-and-cut procedure which starts by solving the linear program relaxation of  $\text{DOMP}_{\text{WOC}}$ , and then it includes (SOC) as valid inequalities whenever necessary. This procedure is described in Sect. 3.1.

In this paper, we propose an alternative solution method which exploits the good properties of  $\text{DOMP}_{\text{SOC}}$  avoiding the use of the complete family of strong order constraints. This solution method consists in a row generation procedure which initially considers  $\text{DOMP}_{\text{SOC}}$  without (SOC), and then it iteratively includes these order constraints. As far as we know, this row generation method has not been considered before for DOMP. Section 3.2 is devoted to the description of this procedure.

#### 3.1 Branch-and-cut for $\text{DOMP}_{\text{WOC}}$

The branch-and-cut procedure has become an efficient method for solving large instances of models where the number of constraints is intractable for solvers. For instance, it has been successfully applied to the matching problem with blossoms (Edmonds and Johnson 1973; Grötschel and Holland 1985; Letchford et al. 2004), problems related to trees (Fernández et al. 2017; Magnanti and Wolsey 1995), clustering (Benati et al. 2017), and the orienteering arc routing problem (Archetti et al. 2016), to name a few. In Labbé et al. (2017), a branch-and-cut method for (SOC) in  $\text{DOMP}_{\text{WOC}}$  formulation is proposed.

This branch-and-cut procedure could be handled by two different perspectives. On the one hand, most of the current solvers have some options to define a fixed pool of cuts that are added automatically when necessary in the cut generation. This means that, changing some parameters in the solver, we can remove some families of constraints from the formulations (thus avoiding to include them initially), which are later added when necessary in the solution process. This automatic feature of solvers is interesting when the constraint separation must be done by enumeration due to the efficient implementation of the solvers in this case.

In our particular case, the use of cut pools in the branch-and-cut method for (SOC) in formulation  $\text{DOMP}_{\text{WOC}}$  seems to require  $\mathcal{O}(n^6)$  operations, since there

are  $\mathcal{O}(n^3)$  (SOC) and  $\mathcal{O}(n^3)$   $x$ -variables in the formulation. Actually, it is  $\mathcal{O}(n^5)$  since each constraint has only  $\mathcal{O}(n^2)$  variables to check. Nevertheless, based on the knowledge of the structure of the problem, an efficient separation method of (SOC) constraints can be developed. This *ad hoc* separation can be included by callbacks allowing a more efficient implementation of the branch-and-cut.

Focusing in DOMP, Labbé et al. (2017) propose an algorithm to separate (SOC) with complexity  $\mathcal{O}(n^3)$ . It is a remarkable quadratic improvement with respect to the pure enumerative approach. Algorithm 1 shows in detail this separation procedure, based on the calculation of left-hand sides of the possible cuts adding and subtracting two values in each iteration.

In Sect. 4, we will develop a complete analysis of the branch-and-cut method using pools of cuts and the branch-and-cut method using callbacks. We will determine whether or not it is advisable to use a separation algorithm that takes advantage of the knowledge of the problem through callbacks.

**Remark 1** According to previous experiences (Labbé et al. 2017; Deleplanque et al. 2020), when valid inequalities (SOC) are embedded in a branch-and-cut procedure over  $\text{DOMP}_{\text{WOC}}$  formulation, they should be added at the root node, but not deeper, in order to find a compromise between the integrality gap and the size of the problem. Hence, for our computational study, we will use this cut-and-branch procedure to check the performance of the solution method proposed in this section.

### 3.2 Row generation procedure for $\text{DOMP}_{\text{SOC}}$

Since  $\text{DOMP}_{\text{WOC}}$  formulation presents coefficients that are not zero-one, in this work we explore the use of a (SOC) relaxation of  $\text{DOMP}_{\text{SOC}}$  adding iteratively these constraints whenever they are necessary. Therefore, the solution method of this section starts with a formulation which has a zero-one coefficient matrix to later add set packing constraints. Hence, we provide a well-behaved (from the solvers point of view) formulation of DOMP without using a huge number of constraints.

The initial formulation which is considered in this row generation procedure is the following.

$$\begin{aligned} (\text{DOMP}_{\text{relax}}) \min \quad & \sum_{i \in I} \sum_{j \in I} \sum_{k \in K} \lambda^k c_{ij}^k x_{ij}^k \\ \text{s.t.} \quad & (2) - (6). \end{aligned}$$

Observe that this formulation corresponds to the DOMP model without imposing the order constraints. Therefore, we are dealing with a relaxation of  $\text{DOMP}_{\text{SOC}}$ . In this case, the proposed relaxation results in the  $p$ -median problem.

For each obtained solution in the branch-and-bound of  $\text{DOMP}_{\text{relax}}$ , (SOC) are checked by using Algorithm 1 and added to the model when necessary. Consequently, this row generation method ensures the order by only using a moderate number of (SOC). This allows to handle bigger instances to be solved in a reasonable computing time as we will see in Sect. 4.



**Algorithm 1** (SOC) separation

**Input:** Let  $(\hat{x}, \hat{y})$  be a solution of (DOMP<sub>WOC</sub>)/(DOMP<sub>relax</sub>) with a subset of (SOC).

**Output:** Violated cuts / model constraints (SOC).

1: Let  $(i, j)$  such that  $r_{ij} = 1$ . Then, compute:

$$lhs := \sum_{i \in I} \sum_{\substack{j \in I: \\ r_{ij} \geq 1}} \hat{x}_{ij}^1 + \hat{x}_{ij}^2.$$

2: if  $lhs > 1$  then

3:   Add constraint  $\sum_{i \in I} \sum_{\substack{j \in I: \\ r_{ij} \geq 1}} x_{ij}^1 + x_{ij}^2 \leq 1$ .

4: end if

5: for  $\ell = 2, \dots, n^2$  do

6:   Let  $(i, j), (i', j')$  such that  $r_{ij} = \ell, r_{i'j'} = \ell - 1$ . Then, compute:

$$lhs := lhs + \hat{x}_{ij}^2 - \hat{x}_{i'j'}^1.$$

7:   if  $lhs > 1$  then

8:     Add constraint  $\sum_{i \in I} \sum_{\substack{j \in I: \\ r_{ij} \geq \ell}} x_{ij}^1 + \sum_{i \in I} \sum_{\substack{j \in I: \\ r_{ij} \leq \ell}} x_{ij}^2 \leq 1$ .

9:   end if

10: end for

11: for  $k = 2, \dots, n - 1$  do

12:   Let  $(i, j), (i', j')$  such that  $r_{ij} = 1, r_{i'j'} = n^2$ . Then, compute:

$$lhs := lhs + \hat{x}_{ij}^{k+1} - \hat{x}_{i'j'}^{k-1}.$$

13:   if  $lhs > 1$  then

14:     Add constraint  $\sum_{i \in I} \sum_{\substack{j \in I: \\ r_{ij} \geq 1}} x_{ij}^k + x_{ij}^{k+1} \leq 1$ .

15:   end if

16:   for  $\ell = 2, \dots, n^2$  do

17:     Let  $(i, j), (i', j')$  such that  $r_{ij} = \ell, r_{i'j'} = \ell - 1$ . Then, compute:

$$lhs := lhs + \hat{x}_{ij}^{k+1} - \hat{x}_{i'j'}^k.$$

18:     if  $lhs > 1$  then

19:       Add constraint  $\sum_{i \in I} \sum_{\substack{j \in I: \\ r_{ij} \geq \ell}} x_{ij}^k + \sum_{i \in I} \sum_{\substack{j \in I: \\ r_{ij} \leq \ell}} x_{ij}^{k+1} \leq 1$ .

20:     end if

21:   end for

22: end for

**Return:** All violated cuts / model constraints found from (SOC) family.

### 3.2.1 Bounds in constraint relaxations

In constraint relaxations, any integer solution has to be checked to be valid according to the problem definition. However, there are different alternatives for continuing the branch-and-bound tree exploration when a fractional solution arises. One of them is checking all model constraints to improve the lower bound. Another one is to branch in a particular fractional variable.

One issue to be taken into account is the way in which a subset of (SOC), that a solution does not verify, is selected to be included in the formulation in order

to improve the lower bound without increasing too much the formulation size. To develop this idea, we introduce the following proposition.

**Proposition 1** *Given an integer solution  $(\hat{x}, \hat{y})$  for  $\text{DOMP}_{\text{relax}}$ , if  $\hat{x}$  verifies that*

$$\sum_{i' \in I} \sum_{j' \in I} x_{i'j'}^k + \sum_{i' \in I} \sum_{j' \in I} x_{i'j'}^{k-1} \leq b, \quad i, j \in I, k \in K, k \neq 1, \quad (7)$$

$$r_{i'j'} \leq r_{ij} \quad r_{i'j'} \geq r_{ij}$$

for  $b \in [1, 2)$ , then  $\hat{x}$  satisfies all (SOC).

**Proof** If  $b = 1$ , then (7) are equal to (SOC). Thus, the result follows trivially. Assume that  $b > 1$ . In this case, since  $\hat{x}$  is integer and  $b < 2$ , then the left hand side of (7) must be at most one. Consequently,  $\hat{x}$  satisfies (SOC).  $\square$

As a result of Proposition 1, when an integer solution is obtained in the branch-and-bound tree of  $\text{DOMP}_{\text{relax}}$ , the *lhs* described in Algorithm 1 could be compared with  $b \in [1, 2)$  instead of comparing it with 1. Therefore, if  $lhs > b$  in Algorithm 1, then the corresponding (SOC) are included.

However, varying this  $b$  value could affect the number of added cuts when a fractional solution is found and thus, the number of explored nodes in the branch-and-bound tree. When  $b$  is close to 2, then the number of identified constraints (7) which are not verified by the solution is smaller and they are the most violated cuts. Consequently, for big values of  $b$ , the number of added cuts will be reduced. Nonetheless, since the number of added cuts is smaller, the number of explored nodes in the branch-and-bound is expected to be bigger.

**Remark 2** Following Remark 1, we separate fractional solutions only at root node. Hence, for deeper nodes, Algorithm 1 is called only when integer solutions are found obtaining upper bounds. Beyond these concerns, lower and upper bounds get closer within the branch-and-bound tree as usually.

In order to experimentally check how the value of  $b$  could impact in times, cuts, and nodes of the row generation proposed in this section, we present Table 1. We show the results for the instances of sizes  $n = 20, 30$ , and 40 that will be detailed in Sect. 4. Particularly, in Table 1, first column shows the number of clients; the second column reports the number of open facilities; the third set of columns shows the computing time for each  $b$  value; the fourth set of columns represents the number of added cuts in the row generation procedure; and finally, the last group of columns reports the number of explored nodes in the branch-and-bound tree. In all cases, each row reports the average value of ten instances. We have tested the results for  $b = 1, b = 1.1$ , and  $b = 1.3$ .

In Table 1, we can observe that  $b = 1$  reports better results than  $b = 1.1$  and  $b = 1.3$  when  $n = 40$ . Note that for the largest instances, although the number of added cuts in the branch-and-bound process is bigger, the number of nodes and the

**Table 1** Time, cuts, and nodes for different r. h. s. in the row generation procedure using Algorithm 1 to separate (7)

n	p	Time			Cuts			Nodes		
		b = 1.0	b = 1.1	b = 1.3	b = 1.0	b = 1.1	b = 1.3	b = 1.0	b = 1.1	b = 1.3
		20	5	11.64	11.73	<b>9.93</b>	1440	1293	<b>1048</b>	1
	6	<b>6.47</b>	6.83	6.82	1215	1093	<b>917</b>	<b>1</b>	1	3
	10	1.22	<b>1.09</b>	1.12	554	512	<b>438</b>	<b>1</b>	<b>1</b>	<b>1</b>
30	7	158.08	<b>137.63</b>	506.88	4576	3930	<b>3258</b>	<b>11</b>	16	1133
	10	75.47	<b>69.65</b>	131.64	2714	2412	<b>1992</b>	77	122	645
	15	22.15	<b>21.84</b>	27.70	1429	1302	<b>1129</b>	44	<b>43</b>	170
40	10	<b>576.82</b>	859.65	4338.31	7147	6243	<b>5308</b>	<b>91</b>	421	5517
	13	<b>1367.96</b>	1564.24	4222.45	5374	4918	<b>4064</b>	<b>964</b>	1268	6789
	20	<b>1060.18</b>	1091.21	1860.01	2691	2491	<b>2163</b>	2408	<b>2398</b>	5935
<b>Total Average:</b>		<b>328.00</b>	376.40	1110.52	2714	2419	<b>2032</b>	<b>360</b>	427	2020

The minimum values among the compared options are in bold

computing times are smaller since the gaps at the root node are smaller. Consequently, for the computational results reported in Sect. 4, the choice of  $b = 1$  is used in the row generation procedure.

## 4 Computational experiments

This section is devoted to the analysis of the solution methods introduced in this paper. The goals of this computational study are the following: (1) checking the differences among the results of formulations  $\text{DOMP}_{\text{SOC}}$  and  $\text{DOMP}_{\text{WOC}}$  and determining their limitations; (2) comparing two approaches to implement the branch-and-cut and the row generation algorithms for DOMP. The difference between these two approaches relies on the fact that the first one uses a fixed constraint/cut pool handled by the solver and the second one applies the separator described in Algorithm 1 implementing a callback; (3) comparing the different results between the two solution methods defined in Sect. 3.

The instances used in this computational study were introduced for the first time in Deleplanque et al. (2020). These instances were created to test different weighted order vectors  $\lambda$  beyond the classical ones, namely  $p$ -median,  $p$ -center,  $k$ -centrum problems, etc. The weighted vector  $\lambda$  was randomly generated such that  $\lambda^k \in \left[ \frac{n}{4}, n \right]$  for  $k \in K$ . Furthermore, in that data set, there are small- to large-sized instances to perform an exhaustive computational study. The reader can find the mentioned instances in (Deleplanque et al. 2022).

The models were coded in C and solved with SCIP v.6.0.2 (Achterberg 2009; Gleixner et al. 2018) using as optimization solver CPLEX 20.1.0 on a Mac OS Catalina with a Core Intel Xeon W clocked at 3.2 GHz and 96 GB of RAM memory.

In the computational experience, for all the considered formulations and solutions methods, we have included a preprocessing phase. In this stage, we are able to reduce the number of necessary variables to define the problem in terms of optimality. We refer the reader to Labbé et al. (2017) for more details. Besides, we have given an incumbent solution provided by a GRASP heuristic (Deleplanque et al. 2020). This solution let us provide a good upper bound from the beginning of the corresponding solution method.

Table 2 contains the results within two hours of 90 instances up to 40 clients, namely ten instances of each configuration of  $n$  and  $p$  for two different formulations:  $\text{DOMP}_{\text{WOC}}$  and  $\text{DOMP}_{\text{SOC}}$ . This table and the following ones show the average results for these ten instances: the average CPU time (Time), the number of instances not solved in the time limit (#Unsolved), the gap at the root node ( $\text{GAP}_{\text{root}}(\%)$ ), the gap at termination ( $\text{GAP}(\%)$ ), the number of variables after preprocessing (Vars), the number of constraints (OrigCons), the number of cuts/constraints added in the procedure (Cuts), the number of nodes (Nodes), and the required memory (Memory (MB)). Observe that, for instances with  $n = 30$  and  $n = 40$ ,  $\text{DOMP}_{\text{SOC}}$  formulation provides better results than  $\text{DOMP}_{\text{WOC}}$ .  $\text{DOMP}_{\text{SOC}}$  presents a better linear relaxation value (see gap at the root node), and it needs less nodes at the branch-and-bound tree. Consequently,  $\text{DOMP}_{\text{SOC}}$  requires less solution

**Table 2** Results of formulations (DOMP<sub>woc</sub>) and (DOMP<sub>soc</sub>) for  $n \in \{20, 30, 40\}$

$n$	$p$	Time (#UnsolVED)		GAProot (%)		GAP (%)		Vars		OrigCons		Nodes		Memory (MB)		
		DOMP <sub>woc</sub>	DOMP <sub>soc</sub>	DOMP <sub>woc</sub>	DOMP <sub>soc</sub>	DOMP <sub>woc</sub>	DOMP <sub>soc</sub>	DOMP <sub>woc</sub>	DOMP <sub>soc</sub>	DOMP <sub>woc</sub>	DOMP <sub>soc</sub>	DOMP <sub>woc</sub>	DOMP <sub>soc</sub>	DOMP <sub>woc</sub>	DOMP <sub>soc</sub>	
20	5	<b>6.30</b>	(0)	35.73	(0)	4.42	<b>0.10</b>	0.00	0.00	6054	<b>460</b>	8041	348	<b>1</b>	<b>39</b>	543
	6	<b>6.10</b>	(0)	23.19	(0)	4.37	<b>0.01</b>	0.00	0.00	5706	<b>460</b>	8041	297	<b>1</b>	<b>36</b>	518
	10	<b>1.59</b>	(0)	9.03	(0)	2.74	<b>0.01</b>	0.00	0.00	4211	<b>460</b>	8041	11	<b>1</b>	<b>23</b>	406
30	7	3271.17	(2)	<b>553.75</b>	(0)	7.33	<b>0.76</b>	0.82	<b>0.00</b>	20643	<b>990</b>	27061	109507	<b>6</b>	<b>1910</b>	3280
	10	2592.65	(1)	<b>339.62</b>	(0)	7.56	<b>0.66</b>	0.19	<b>0.00</b>	18245	<b>990</b>	27061	128481	<b>16</b>	<b>1055</b>	2870
	15	442.39	(0)	<b>198.13</b>	(0)	7.43	<b>0.30</b>	0.00	0.00	13952	<b>990</b>	27061	22433	<b>17</b>	<b>261</b>	2308
40	10	7200.95	(10)	<b>3437.82</b>	(0)	9.04	<b>1.46</b>	6.67	<b>0.00</b>	48065	<b>1720</b>	64081	56677	<b>51</b>	<b>5250</b>	12724
	13	7200.87	(10)	<b>4221.43</b>	(3)	10.39	<b>2.05</b>	7.37	<b>0.36</b>	43664	<b>1720</b>	64081	62211	<b>148</b>	<b>4335</b>	11221
	20	7200.65	(10)	<b>3748.44</b>	(1)	11.62	<b>2.46</b>	5.45	<b>0.42</b>	32820	<b>1720</b>	64081	116803	<b>331</b>	<b>2605</b>	8665
<b>Total Aver-</b>		3102.52	(33)	<b>1396.35</b>	(4)	7.21	<b>0.87</b>	2.28	<b>0.09</b>	21484	<b>1057</b>	33061	55196	<b>64</b>	<b>1724</b>	4726

The minimum values among the compared options are in bold

time. In addition, we can conclude that the large number of constraints implies an increment of memory for  $\text{DOMP}_{\text{SOC}}$  which makes this formulation too heavy for sizes of  $n$  greater than 40. Then, together with the fact that  $\text{DOMP}_{\text{WOC}}$  cannot solve any of the instances for  $n = 40$ , we study other alternatives which add (SOC) iteratively until certifying optimality. Thus, in the following tables, we report the results of the methods proposed in Sect. 3.

Nowadays, commercial solvers include options to code valid inequalities in a branch-and-cut procedure. In this context, these valid inequalities are usually known as *user cuts*, while in constraint relaxations, these model constraints are known as *lazy constraints*. The coding of the cuts/constraints can be done just giving them as input of the linear program. These automatic strategies need to encode all the cuts/constraints in advance within a fixed pool, with an ensuing waste of computer memory. Also, the management of these potential cuts/constraints follows a pre-implemented strategy without taking advantage of the particularities of the formulation beyond the solver pattern recognition based on developers' experience. On the other hand, the use of an oracle (in our case, Algorithm 1) to add the cuts/constraints on the fly implementing callbacks could save memory on the whole process (see, e.g., Ackooij and de Oliveira 2014; Blado and Toriello 2021; de Oliveira and Sagastizábal 2014; Mazzi et al. 2021; Wolf et al. 2014, and the references therein). Besides, it allows to control when to check and to add those cuts/constraints that is an advantage by itself. For instance, in the row generation solution method, we check model constraints (SOC) at the root node (regardless the solution is fractional or integer) and in any node with integer solution. We refer the reader to (CPLEX 2022; SCIP 2022a, b) for a detailed discussion.

Table 3 presents the results, in two hours of time limit, of the branch-and-cut introduced in Sect. 3.1, i.e., ( $\text{DOMP}_{\text{WOC}}$ ) with (SOC) as valid inequalities. Here, we follow two different strategies: we code (SOC) defining a fixed user cut pool in SCIP and the solver decides when to check and to add them (Pool); or we check the constraints using Algorithm 1 which adds them by an user callback when needed at the root node (Callback). The results exhibit that the method using Algorithm 1 shows better solving times than the automatic approach. Besides, the method using the callback can solve nine more instances than the automatic one. Note that these better results are explained by the efficiency of our separation algorithm. Regarding the required memory, observe that memory used by the automatic method increases quickly with  $n$ . Therefore, the application of this method does not seem to be useful for bigger instances since they could not be loaded: it requires around 50 GB of RAM memory already for  $n = 60$ .

Table 4 reports the results, in two hours of computing time, of the row generation method, i.e.,  $\text{DOMP}_{\text{relax}}$  with (SOC) as model constraints not included from the beginning. Two approaches to carry out the row generation are considered: the automatic use of (SOC) defining a fixed lazy constraint pool (Pool) and the application of Algorithm 1 to add (SOC) when necessary (Callback). In this table, we have included the same columns as in Table 3. Note that the callback approach provides the best computing time results and only 16 instances remain unsolved after the time limit. The automatic method requires less cuts and nodes than Callback. However, the required memory increases faster when using the automatic approach because it

**Table 3** B&C: DOMP<sub>woc</sub> with (SOC) as valid inequalities

<i>n</i>	<i>p</i>	Time (# Unsolved)		GAProot (%)		GAP (%)		OrigCons		Cuts		Nodes		Memory (MB)			
		Pool	Callback	Pool	Callback	Pool	Callback	Pool	Callback	Pool	Callback	Pool	Callback	Pool	Callback		
20	5	<b>15.06</b>	(0)	15.78	(0)	<b>0.00</b>	0.33	0.00	0.00	8060	<b>460</b>	<b>544</b>	1471	<b>1</b>	<b>1</b>	348	<b>106</b>
	6	<b>9.68</b>	(0)	9.70	(0)	<b>0.01</b>	0.20	0.00	0.00	8060	<b>460</b>	<b>485</b>	1237	<b>1</b>	<b>1</b>	342	<b>101</b>
	10	3.79	(0)	<b>1.51</b>	(0)	<b>0.00</b>	0.01	0.00	0.00	8060	<b>460</b>	<b>297</b>	607	<b>1</b>	<b>1</b>	263	<b>67</b>
30	7	197.98	(0)	<b>168.44</b>	(0)	<b>1.02</b>	1.08	0.00	0.00	27090	<b>990</b>	<b>1627</b>	4753	<b>8</b>	<b>20</b>	1949	<b>430</b>
	10	103.31	(0)	<b>66.60</b>	(0)	<b>1.06</b>	1.17	0.00	0.00	27090	<b>990</b>	<b>1254</b>	2953	<b>28</b>	<b>41</b>	1697	<b>306</b>
	15	54.09	(0)	<b>34.57</b>	(0)	<b>0.33</b>	0.85	0.00	0.00	27090	<b>990</b>	<b>891</b>	2432	<b>26</b>	<b>67</b>	1324	<b>266</b>
40	10	1279.63	(0)	<b>626.38</b>	(0)	4.73	<b>1.55</b>	0.00	0.00	64120	<b>1720</b>	<b>4300</b>	7899	153	<b>95</b>	7142	<b>1041</b>
	13	2316.83	(1)	<b>1343.62</b>	(0)	5.43	<b>2.22</b>	0.21	<b>0.00</b>	64120	<b>1720</b>	<b>5140</b>	6445	<b>688</b>	<b>890</b>	6410	<b>861</b>
	20	1418.48	(1)	<b>717.14</b>	(0)	3.82	<b>2.78</b>	0.24	<b>0.00</b>	64120	<b>1720</b>	<b>2839</b>	4636	<b>813</b>	<b>1126</b>	4897	<b>641</b>
50	12	3223.91	(0)	<b>1858.95</b>	(0)	2.12	<b>0.89</b>	0.00	0.00	125150	<b>2650</b>	<b>4393</b>	12466	<b>90</b>	<b>150</b>	21412	<b>2228</b>
	16	4057.82	(2)	<b>2364.00</b>	(1)	3.41	<b>1.11</b>	0.17	<b>0.05</b>	125150	<b>2650</b>	<b>4756</b>	10172	<b>341</b>	<b>658</b>	19636	<b>1922</b>
	25	<b>3166.70</b>	(1)	2311.38	(2)	4.01	<b>1.27</b>	0.40	<b>0.35</b>	125150	<b>2650</b>	<b>3273</b>	9787	<b>611</b>	<b>912</b>	14404	<b>1512</b>
60	15	6953.04	(8)	<b>5884.18</b>	(5)	5.64	<b>1.14</b>	3.43	<b>0.71</b>	216180	<b>3780</b>	<b>4239</b>	23249	62	<b>58</b>	51404	<b>5013</b>
	20	6762.80	(7)	<b>5267.87</b>	(6)	5.89	<b>1.06</b>	2.77	<b>0.53</b>	216180	<b>3780</b>	<b>4989</b>	22206	<b>111</b>	<b>144</b>	46079	<b>4528</b>
	30	6950.37	(9)	<b>5749.35</b>	(6)	7.61	<b>1.74</b>	1.88	<b>0.72</b>	216180	<b>3780</b>	<b>5655</b>	15535	<b>261</b>	<b>1169</b>	35170	<b>3011</b>
<b>Total Average:</b>		2434.23	(29)	<b>1761.30</b>	(20)	3.01	<b>1.16</b>	0.61	<b>0.16</b>	88120	<b>1920</b>	<b>2979</b>	8390	<b>213</b>	<b>356</b>	14165	<b>1469</b>

The minimum values among the compared options are in bold

**Table 4** Row generation, i.e., SOC (model constraints) added iteratively

$n$	$p$	Time (# Unsolved)		GAProot (%)		GAP (%)		OrigCons		Cuts		Nodes		Memory (MB)			
		Pool	Callback	Pool	Callback	Pool	Callback	Pool	Callback	Pool	Callback	Pool	Callback	Pool	Callback		
20	5	15.63	(0)	<b>11.64</b>	(0)	<b>0.19</b>	0.29	0.00	0.00	8041	<b>441</b>	<b>514</b>	1440	<b>1</b>	<b>1</b>	346	<b>129</b>
	6	10.28	(0)	<b>6.47</b>	(0)	0.03	<b>0.00</b>	0.00	0.00	8041	<b>441</b>	<b>483</b>	1215	<b>1</b>	<b>1</b>	340	<b>124</b>
	10	3.80	(0)	<b>1.22</b>	(0)	<b>0.00</b>	0.01	0.00	0.00	8041	<b>441</b>	<b>309</b>	554	<b>1</b>	<b>1</b>	260	<b>91</b>
30	7	<b>150.01</b>	(0)	158.08	(0)	<b>0.69</b>	1.11	0.00	0.00	27061	<b>961</b>	<b>1422</b>	4576	<b>4</b>	<b>11</b>	1928	<b>497</b>
	10	88.03	(0)	<b>75.47</b>	(0)	<b>0.96</b>	1.15	0.00	0.00	27061	<b>961</b>	<b>1082</b>	2714	<b>15</b>	<b>77</b>	1668	<b>395</b>
	15	49.04	(0)	<b>22.15</b>	(0)	<b>0.42</b>	0.85	0.00	0.00	27061	<b>961</b>	<b>783</b>	1429	<b>12</b>	<b>44</b>	1321	<b>367</b>
40	10	837.75	(0)	<b>576.82</b>	(0)	<b>1.51</b>	1.57	0.00	0.00	64081	<b>1681</b>	<b>2907</b>	7147	<b>44</b>	<b>91</b>	7059	<b>1256</b>
	13	<b>1061.32</b>	(0)	1367.96	(1)	<b>2.04</b>	2.23	<b>0.00</b>	0.07	64081	<b>1681</b>	<b>3212</b>	5374	<b>207</b>	<b>964</b>	6331	<b>1099</b>
	20	<b>1093.46</b>	(0)	1060.18	(1)	<b>2.55</b>	2.81	<b>0.00</b>	0.13	64081	<b>1681</b>	<b>2404</b>	2691	<b>680</b>	<b>2408</b>	4839	<b>877</b>
50	12	3061.61	(1)	<b>1648.54</b>	(0)	2.08	<b>0.88</b>	0.12	<b>0.00</b>	125101	<b>2601</b>	<b>3858</b>	10720.2	<b>91.7</b>	<b>116</b>	21271	<b>2609</b>
	16	3104.23	(0)	<b>1691.65</b>	(0)	<b>1.09</b>	1.11	0.00	0.00	125101	<b>2601</b>	<b>3429</b>	7236.6	<b>219.9</b>	<b>662</b>	19496	<b>2248</b>
	25	2958.29	(2)	<b>1838.77</b>	(1)	<b>1.23</b>	1.26	0.34	<b>0.28</b>	125101	<b>2601</b>	<b>2993</b>	3919.9	<b>526</b>	<b>2217</b>	14316	<b>1752</b>
60	15	7313.70	(9)	<b>5357.93</b>	(5)	4.45	<b>1.15</b>	3.19	<b>0.67</b>	216121	<b>3721</b>	<b>4515</b>	17078	<b>9.2</b>	<b>63</b>	51469	<b>5346</b>
	20	6439.44	(6)	<b>4529.22</b>	(4)	1.45	<b>1.05</b>	0.94	<b>0.36</b>	216121	<b>3721</b>	<b>4605</b>	11582	<b>33.6</b>	<b>444</b>	46014	<b>4530</b>
	30	6733.81	(8)	<b>5093.43</b>	(4)	2.42	<b>1.74</b>	1.02	<b>0.46</b>	216121	<b>3721</b>	<b>4402</b>	6169.1	<b>256.6</b>	<b>2127</b>	34981	<b>3463</b>
<b>Total Average:</b>		2194.69	(26)	<b>1562.64</b>	(16)	1.41	<b>1.15</b>	0.37	<b>0.13</b>	88081	<b>1881</b>	<b>2461</b>	5590	<b>140</b>	<b>615</b>	14109	<b>1652</b>

The minimum values among the compared options are in bold



needs to encode all the original (SOC) constraints which are  $\mathcal{O}(n^3)$ . Consequently, the performance of the row generation following the automatic separation shows, in general, worse results than using Algorithm 1.

From Tables 3 and 4, we can conclude that the performance of the automatic branch-and-cut and the automatic row generation are quite limited in comparison with the use of the separation presented in Algorithm 1. The reason of the better performance of the callback approach is that Algorithm 1 exploits the knowledge of the problem. Therefore, for the study of larger instances, we focus on the branch-and-cut and row generation approaches using Algorithm 1.

Table 5 shows a comparison between the results provided by the callback versions of the branch-and-cut method (B&C) and the row generation technique (Row-Gen). For these experiments, we establish a time limit of five hours. Overall, the row generation technique outperforms the branch-and-cut in terms of computational times. Moreover, the row generation is able to solve more instances than the branch-and-cut in five hours.

Note that, for some huge instances, the problem does not get even the root node bounds. For those instances,  $\text{GAP}_{\text{root}}(\%)$  and  $\text{GAP}(\%)$  report the same value and thus,  $\text{GAP}_{\text{root}}(\%)$  cannot be analyzed as the gap at the root node, but as the gap at the root node at the time limit.

Regarding  $n = 60$  instances, the branch-and-cut is able to solve 20 out of 30 instances in five hours, whereas this solution method certifies optimality for only 13 instances in two hours (see Table 3). These three extra hours also let the row generation algorithm to solve 24 instances, seven more than the same algorithm in two hours (see Table 4).

For most of the instances, the integrality gap at termination provided by the row generation procedure is smaller than the one obtained by the branch-and-cut, even with less cuts added. This gives us the idea that the added cuts are more accurate when (WOC) family is not included in the formulation. However, for  $n = 100$ , the gap at termination is smaller for the branch-and-cut procedure since the added cuts cannot improve the lower bound given by the linear relaxation of the program in both solution methods within the time limit.

To analyze the differences between the branch-and-cut and the row generation, starting from  $\text{DOMP}_{\text{WOC}}$  and  $\text{DOMP}_{\text{relax}}$ , respectively, we give the solver up to 24 hours of time limit. Thereby, the influence of the linear relaxation bound is not so decisive. Furthermore, the branch-and-price-and-cut (B&P &C) described in Delplanque et al. (2020) has been tested for those instances in the same computer and with the same time limit. The reader could see, in Table 6, how the gaps at termination are smaller on average for the row generation procedure. In fact, they are reduced by half and for the row generation are less than 2%. This approach needs less cuts to have a reasonable bound at the root node what let it branch faster to generate more nodes and improve the bounds. Thus, whereas the solution method detailed in Sect. 3.1 solves four instances out of 60 for these large-sized instances, the solution method proposed in Sect. 3.2 is able to solve eight instances to optimality and on top of that, the row generation procedure also reduces the gap at termination for those instances which are not solved. The B&P &C procedure uses less

**Table 5** B&C vs row generation in 5 hours of time limit

<i>n</i>	<i>p</i>	Time (#Unsolved)		GAProot (%)		GAP (%)		Vars		OrigCons		Cuts		Nodes		Memory (MB)		
		B&C	RowGen	B&C	Row-Gen	B&C	Row-Gen	B&C	Row-Gen	B&C	Row-Gen	B&C	Row-Gen	B&C	Row-Gen	B&C	RowGen	
20	5	15.78	(0)	<b>11.64</b>	(0)	0.33	<b>0.29</b>	0.00	0.00	6054	460	<b>441</b>	1471	<b>1440</b>	1	1	<b>106</b>	129
	6	9.70	(0)	<b>6.47</b>	(0)	0.20	<b>0.00</b>	0.00	0.00	5706	460	<b>441</b>	1237	<b>1215</b>	1	1	<b>101</b>	124
	10	1.51	(0)	<b>1.22</b>	(0)	0.01	<b>0.01</b>	0.00	0.00	4211	460	<b>441</b>	607	<b>554</b>	1	1	<b>67</b>	91
30	7	168.44	(0)	<b>158.08</b>	(0)	<b>1.08</b>	1.11	0.00	0.00	20643	990	<b>961</b>	4753	<b>4576</b>	20	<b>11</b>	<b>430</b>	497
	10	<b>66.60</b>	(0)	75.47	(0)	1.17	<b>1.15</b>	0.00	0.00	18245	990	<b>961</b>	2953	<b>2714</b>	<b>41</b>	77	<b>306</b>	395
	15	34.57	(0)	<b>22.15</b>	(0)	<b>0.85</b>	0.85	0.00	0.00	13952	990	<b>961</b>	2432	<b>1429</b>	67	<b>44</b>	<b>266</b>	367
40	10	626.38	(0)	<b>576.82</b>	(0)	<b>1.55</b>	1.57	0.00	0.00	48065	1720	<b>1681</b>	7899	<b>7147</b>	95	<b>91</b>	<b>1041</b>	1256
	13	<b>1343.62</b>	(0)	1390.00	(0)	<b>2.22</b>	2.23	0.00	0.00	43664	1720	<b>1681</b>	6445	<b>5374</b>	<b>890</b>	981	<b>861</b>	1099
	20	<b>717.14</b>	(0)	1082.85	(0)	<b>2.78</b>	2.81	0.00	0.00	32820	1720	<b>1681</b>	4636	<b>2691</b>	<b>1126</b>	2546	<b>641</b>	877
50	12	1858.95	(0)	<b>1648.54</b>	(0)	0.89	<b>0.88</b>	0.00	0.00	94784	2650	<b>2601</b>	12466	<b>10720</b>	150	<b>116</b>	<b>2228</b>	2609
	16	2497.71	(0)	<b>1691.65</b>	(0)	1.11	<b>1.11</b>	0.00	0.00	85630	2650	<b>2601</b>	10172	<b>7237</b>	687	<b>662</b>	<b>1923</b>	2248
	25	3425.89	(1)	<b>2918.77</b>	(1)	1.27	<b>1.26</b>	0.29	<b>0.23</b>	63776	2650	<b>2601</b>	9787	<b>3920</b>	<b>1103</b>	3277	<b>1516</b>	1777
60	15	11007.79	(4)	<b>9004.34</b>	(2)	<b>1.13</b>	1.14	0.33	<b>0.30</b>	161807	3780	<b>3721</b>	23249	<b>17095</b>	291	<b>268</b>	<b>5046</b>	5438
	20	9238.02	(3)	<b>6695.45</b>	(1)	1.06	<b>1.05</b>	0.28	<b>0.17</b>	144983	3780	<b>3721</b>	22206	<b>11582</b>	<b>353</b>	599	4542	<b>4540</b>
	30	10103.21	(3)	<b>8524.74</b>	(3)	1.74	<b>1.74</b>	0.39	<b>0.24</b>	109804	3780	<b>3721</b>	15535	<b>6169</b>	<b>2071</b>	3381	<b>3030</b>	3493
70	17	16541.03	(8)	<b>15920.52</b>	(8)	1.01	<b>0.95</b>	0.67	<b>0.56</b>	259406	5110	<b>5041</b>	31406	<b>23228</b>	95	161	<b>9281</b>	9666
	23	16506.34	(8)	<b>15748.13</b>	(8)	<b>1.10</b>	1.10	0.68	<b>0.54</b>	231680	5110	<b>5041</b>	33833	<b>16004</b>	<b>215</b>	654	8792	<b>7851</b>
	35	18011.10	(10)	<b>17281.08</b>	(8)	2.05	<b>2.04</b>	1.34	<b>1.13</b>	173955	5110	<b>5041</b>	22814	<b>8648</b>	<b>721</b>	1924	<b>5686</b>	6259
80	20	18041.98	(10)	<b>18001.87</b>	(10)	4.22	<b>1.47</b>	4.22	<b>1.37</b>	383199	6640	<b>6561</b>	42109	<b>30421</b>	1	11	14227	<b>13641</b>
	26	18030.43	(10)	<b>17433.69</b>	(9)	1.39	<b>0.78</b>	1.32	<b>0.57</b>	346926	6640	<b>6561</b>	38792	<b>20784</b>	<b>100</b>	325	13194	<b>11930</b>
	40	16047.37	(7)	<b>14613.12</b>	(6)	0.57	<b>0.57</b>	0.32	<b>0.29</b>	259186	6640	<b>6561</b>	17419	<b>10420</b>	<b>794</b>	1273	<b>6530</b>	8940

Table 5 (continued)

n	p	Time (#Unsolved)		GAProot (%)		GAP (%)		Vars		OrigCons		Cuts		Nodes		Memory (MB)		
		B&C	RowGen	B&C	Row-Gen	B&C	Row-Gen	B&C	Row-Gen	B&C	Row-Gen	B&C	Row-Gen	B&C	Row-Gen	B&C	Row-Gen	
90	22	18103.76	(10)	<b>18002.56</b>	(10)	8.21	<b>7.34</b>	8.21	<b>7.34</b>	549561	8370	<b>8281</b>	28820	<b>28267</b>	1	1	14849	<b>13946</b>
	30	18078.18	(10)	<b>17886.15</b>	(9)	4.35	<b>0.56</b>	4.34	<b>0.46</b>	488316	8370	<b>8281</b>	35203	<b>23544</b>	3	37	<b>15053</b>	16879
	45	14629.97	(7)	<b>14340.26</b>	(6)	<b>0.56</b>	0.56	0.36	<b>0.31</b>	368560	8370	<b>8281</b>	18634	<b>13363</b>	410	700	<b>8715</b>	13660
100	25	18150.10	(10)	<b>18006.01</b>	(10)	<b>8.01</b>	11.71	<b>8.01</b>	11.71	749074	10300	<b>10201</b>	<b>19420</b>	21398	1	1	14407	<b>13231</b>
	33	18138.18	(10)	<b>18004.78</b>	(10)	7.05	<b>3.88</b>	7.05	<b>3.88</b>	672511	10300	<b>10201</b>	24045	<b>23931</b>	1	1	<b>14681</b>	16009
	50	18065.05	(10)	<b>17980.43</b>	(9)	0.58	<b>0.58</b>	0.53	<b>0.44</b>	504981	10300	<b>10201</b>	25922	<b>16253</b>	66	351	<b>13819</b>	18941
<b>Total Average:</b>		9239.22	(121)	<b>8778.77</b>	(110)	2.09	<b>1.80</b>	1.42	<b>1.09</b>	216352	4447	<b>4388</b>	17195	<b>11856</b>	345	648	<b>5975</b>	6515

age:

The minimum values among the compared options are in bold

**Table 6** B&P &C, B &C, and row generation in 24 hours of time limit

<i>n</i>	100						120						Total Average		
	25	33	50	60	40	60	30	40	60	40	60				
Time (#Unsolved)	B&P &C	86400.19	(10)	86400.29	(10)	86400.08	(10)	86400.18	(10)	86400.27	(10)	86400.12	(10)	86400.19	(60)
	B&C	86557.06	(10)	85594.56	(9)	<b>78019.88</b>	(8)	86841.70	(10)	86833.42	(10)	81439.12	(9)	84214.29	(56)
	RowGen	<b>74555.59</b>	(7)	<b>76691.23</b>	(8)	79403.08	(8)	<b>86403.24</b>	(10)	<b>86402.13</b>	(10)	<b>79855.20</b>	(9)	<b>80551.74</b>	(52)
GAProot (%)	B&P &C	4.50		3.65		2.32		<b>5.86</b>		4.59		2.50		3.90	
	B&C	4.84		1.08		0.58		8.74		7.51		<b>0.49</b>		3.87	
	RowGen	<b>0.57</b>		<b>0.66</b>		<b>0.50</b>		7.60		<b>2.06</b>		0.49		<b>1.98</b>	
GAP (%)	B&P &C	4.50		3.65		2.32		<b>5.86</b>		4.59		2.50		3.90	
	B&C	4.83		1.01		0.40		8.74		7.51		0.40		3.81	
	RowGen	<b>0.36</b>		<b>0.52</b>		<b>0.28</b>		7.60		<b>2.04</b>		<b>0.37</b>		<b>1.86</b>	
Vars	B&P &C	<b>47921</b>		<b>45038</b>		<b>40436</b>		<b>52678</b>		<b>49576</b>		<b>41743</b>		<b>46232</b>	
	B&C	595107		725934		605525		1293335		1155202		871156		874377	
	RowGen	595107		725934		605525		1293335		1155202		871156		874377	
OrigCons	B&P &C	<b>301</b>		<b>301</b>		<b>301</b>		<b>361</b>		<b>361</b>		<b>361</b>		<b>331</b>	
	B&C	10300		10300		10300		14760		14760		14760		12530	
	RowGen	10201		10201		10201		14641		14641		14641		12421	
Cuts	B&P &C	<b>10573</b>		<b>9214</b>		<b>5760</b>		<b>13717</b>		<b>12259</b>		<b>7611</b>		<b>9856</b>	
	B&C	89913		77195		25922		44518		55814		34981		54724	
	RowGen	25990		49665		26137		44244		42998		24038		35512	
Nodes	B&P &C	<b>1</b>		<b>1</b>		<b>1</b>		<b>1</b>		<b>1</b>		<b>1</b>		<b>1</b>	
	B&C	2		58		1205		<b>1</b>		<b>1</b>		140		235	
	RowGen	1198		88		1245		<b>1</b>		26		324		480	
Memory (MB)	B&P &C	<b>1739</b>		<b>1215</b>		<b>663</b>		<b>2105</b>		<b>1413</b>		<b>697</b>		<b>1305</b>	
	B&C	46512		37105		13992		39891		40861		25062		33904	
	RowGen	23225		32057		23869		35066		40005		35755		31663	

The minimum values among the compared options are in bold

variables but the gap at termination is worse because it is not able to solve even the root node. However, this approach would still be useful for bigger instances since it requires much less memory.

## 5 Conclusions

In this work, we have introduced a row generation solution method which has improved the best known performances for DOMP regarding the medium-sized instances of the data set described in Deleplanque et al. (2020). In addition, we have improved the best known solution for an instance with  $n = 90$  and  $p = 45$  (domp90p45v5.domp) that can be found in the mentioned dataset.

For large-sized instances, the lower bound provided by formulations which include (WOC) makes them also a good alternative. For these instances, the lower bound given by the linear relaxation can be barely improved within the time limit. Moreover, comparing with the integrality gap given by the branch-and-price algorithm (Deleplanque et al. 2020), one should note that the column generation of its master problem gives theoretically better lower bounds.

Taking into account that the limits of our row generation algorithm come from the huge number of variables, a combination of row and column generation seems to be a promising approach to be considered as future research line.

**Acknowledgements** The authors of this research acknowledge financial support by the Spanish Ministerio de Ciencia y Tecnología, Agencia Estatal de Investigación and Fondos Europeos de Desarrollo Regional (FEDER) via projects PID2020-114594GB-C21/C22. The authors also acknowledge partial support from NetmeetData: Ayudas Fundación BBVA a equipos de investigación científica 2019, Junta de Andalucía P18-FR-1422, and project CIGE/2021/161 from Generalitat Valenciana. The first author thanks partial support from project FEDER-UCA18-106895. The second and third authors also acknowledge partial support from projects PID2020-114594GB-C21, B-FQM-322-UGR20, FEDER-US-1256951, and FQM-331. The first author has been supported by Subprograma Juan de la Cierva Formación 2019 (FJC2019-039023-I) and PAIDI 2020, postdoctoral fellowship financed by European Social Fund and Junta de Andalucía. The second author also acknowledges PAIDI 2020, postdoctoral fellowship financed by European Social Fund and Junta de Andalucía. We would like to thank Reviewers for taking the necessary time and effort to make constructive suggestions that help to improve the quality of the manuscript.

**Funding** Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

## Declarations

**Conflict of interest** The authors declare no conflict of interest.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## References

- Achterberg T (2009) SCIP: Solving constraint integer programs. *Math Progr Comput* 1(1):1–41. <https://doi.org/10.1007/s12532-008-0001-1>
- Ackooij W, de Oliveira W (2014) Level bundle methods for constrained convex optimization with various oracles. *Comput Opt Appl* 57(3):555–597. <https://doi.org/10.1007/s10589-013-9610-3>
- Aouad A, Segev D (2019) The ordered k-median problem: surrogate models and approximation algorithms. *Math Progr* 177:55–83. <https://doi.org/10.1007/s10107-018-1259-3>
- Archetti C, Corberán A, Plana I, Sanchis JM, Speranza MG (2016) A branch-and-cut algorithm for the orienteering arc routing problem. *Comput Oper Res* 66:95–104. <https://doi.org/10.1016/j.cor.2015.08.003>
- Benati S, Puerto J, Rodríguez-Chía AM (2017) Clustering data that are graph connected. *Eur J Oper Res* 261:43–53. <https://doi.org/10.1016/j.ejor.2017.02.009>
- Blado D, Toriello A (2021) A column and constraint generation algorithm for the dynamic knapsack problem with stochastic item sizes. *Math Progr Comput* 13:85–223. <https://doi.org/10.1007/s12532-020-00189-0>
- 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. <https://doi.org/10.1016/j.cor.2005.03.025>
- Calvino JJ, López-Haro M, Muñoz-Ocaña JM, Puerto J, Rodríguez-Chía AM (2022) Segmentation of scanning-transmission electron microscopy images using the ordered median problem. *Eur J Oper Res* 302(2):671–687. <https://doi.org/10.1016/j.ejor.2022.01.022>
- CPLEX (last accessed November 13, 2022) Differences between user cuts and lazy constraints. <https://www.ibm.com/docs/en/icos/20.1.0?topic=pools-differences-between-user-cuts-lazy-constraints>
- De Oliveira W, Sagastizábal C (2014) Level bundle methods for oracles with on-demand accuracy. *Opt Methods Software* 29(6):1180–1209. <https://doi.org/10.1080/10556788.2013.871282>
- Deleplanque S, LabbéM Ponce D, Puerto J (2020) A branch-price-and-cut procedure for the discrete ordered median problem. *Inform J Comput* 32:582–599. <https://doi.org/10.1287/ijoc.2019.0915>
- Deleplanque S, LabbéM, Ponce D, Puerto J (last accessed November 13, 2022) DOMP repository. [https://gom.ulb.ac.be/gom/wp-content/uploads/2018/12/DOMP\\_Repository.zip](https://gom.ulb.ac.be/gom/wp-content/uploads/2018/12/DOMP_Repository.zip)
- Domínguez E, Marín A (2020) Discrete ordered median problem with induced order. *TOP* 28:793–813. <https://doi.org/10.1007/s11750-020-00570-1>
- Edmonds J, Johnson EL (1973) Matching, Euler tours and the Chinese postman. *Math Progr* 5(1):88–124. <https://doi.org/10.1007/BF01580113>
- Espejo I, Marín A, Puerto J, Rodríguez-Chía AM (2009) A comparison of formulations and solution methods for the minimum-envy location problem. *Comput Oper Res* 36(6):1966–1981. <https://doi.org/10.1016/j.cor.2008.06.013>
- Fernández E, Pozo MA, Puerto J, Scozzari A (2017) Ordered weighted average optimization in multiobjective spanning tree problem. *Eur J Oper Res* 260(3):886–903. <https://doi.org/10.1016/j.ejor.2016.10.016>
- Focacci F, Lodi A, Milano M (2002) Embedding relaxations in global constraints for solving TSP and TSPTW. *Ann Math Artif Intell* 34:291–311. <https://doi.org/10.1023/A:1014492408220>
- Focacci F, Lodi A, Milano M (2002) Optimization-oriented global constraints. *Constraints* 7:351–365. <https://doi.org/10.1023/A:1020589922418>
- Focacci F, Lodi A, Milano M, Vigo D (1999) Solving TSP through the integration of OR and CP techniques. *Electron Notes Discrete Math* 1:3–25. [https://doi.org/10.1016/S1571-0653\(04\)00002-2](https://doi.org/10.1016/S1571-0653(04)00002-2)
- Fourour S, Lebbah Y (2020) Equitable optimization for multicast communication. *Int J Decis Support Syst Technol* 12(3):1–25. <https://doi.org/10.4018/IJDSST.2020070101>
- Gleixner A, Bastubbe M, Eifler L, Gally T, Gamrath G, Gottwald RL, Hendel G, Hojny C, Koch T, Lübbecke ME, Maher SJ, Miltenberger M, Müller B, Pfetsch ME, Puchert C, Rehfeldt D, Schlösser F, Schubert C, Serrano F, Shinano Y, Viernickel JM, Walter M, Wegscheider F, Witt JT, Witzig J (2018) The SCIP Optimization Suite 6.0. ZIB-Report 18-26, Zuse Institute Berlin
- Grötschel M, Holland O (1985) Solving matching problems with linear programming. *Math Progr* 33:243–259. <https://doi.org/10.1007/BF01584376>
- Hakimi SL (1964) Optimum location of switching centers and the absolute centers and medians of a graph. *Oper Res* 12:450–459. <https://doi.org/10.1287/opre.12.3.450>

- Kalcsics J, Nickel S, Puerto J, Rodríguez-Chía AM (2010) The ordered capacitated facility location problem. *TOP* 18:203–222. <https://doi.org/10.1007/s11750-009-0089-0>
- 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. <https://doi.org/10.1016/j.cor.2016.06.004>
- Letchford AN, Reinelt G, Theis DO (2004) A faster exact separation algorithm for blossom inequalities. In: Bienstock D, Nemhauser G (eds) *Integer Programming and Combinatorial Optimization*. Springer, Berlin Heidelberg, pp 196–205
- Magnanti TL, Wolsey LA (1995) Chapter 9 optimal trees. In: Ball MO, Magnanti TL, Monma CL, Nemhauser GL (eds) *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*. Elsevier, pp 503–615
- Marín A, Martínez-Merino LI, Puerto J, Rodríguez-Chía AM (2022) The soft-margin support vector machine with ordered weighted average. *Knowl Based Syst* 237:107705. <https://doi.org/10.1016/j.knsys.2021.107705>
- 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. <https://doi.org/10.1016/j.dam.2008.03.013>
- 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. <https://doi.org/10.1007/s00186-009-0288-3>
- Marín A, Pelegrín M (2019)  $p$ -Median Problems In: Laporte, G., Nickel, S., Saldanha da Gama, F. (eds) *Location Science*. Springer, Cham, pp 25–50 [https://doi.org/10.1007/978-3-030-32177-2\\_2](https://doi.org/10.1007/978-3-030-32177-2_2)
- Martínez-Merino LI, Albareda-Sambola M, Rodríguez-Chía AM (2017) The probabilistic  $p$ -center problem: Planning service for potential customers. *Eur J Oper Res* 262(2):509–520. <https://doi.org/10.1016/j.ejor.2017.03.043>
- Mazzi N, Grothey A, McKinnon K, Sugishita N (2021) Benders decomposition with adaptive oracles for large scale optimization. *Math Progr Comput* 13:683–703. <https://doi.org/10.1007/s12532-020-00197-0>
- Nickel S (2001) Discrete ordered Weber problems. In: Fleischmann B, Lasch R, Derigs U, Domschke W, Rieder U (eds) *Operations Research Proceedings*. Springer, Berlin Heidelberg, pp 71–76
- Nickel S, Puerto J (2005) *Location Theory: a unified approach*. Springer-Verlag, Berlin Heidelberg
- Ogryczak W, Perny P, Weng P (2011) On minimizing ordered weighted regrets in multiobjective Markov decision processes, volume 6992 LNAI of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*
- 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. <https://doi.org/10.1016/j.cor.2018.05.014>
- Puerto J, Ramos AB, Rodríguez-Chía AM (2013) A specialized branch & bound & cut for single-allocation ordered median hub location problems. *Discrete Appl Math* 16:2624–2646. <https://doi.org/10.1016/j.dam.2013.05.035>
- ReVelle CS, Swain R (1970) Central facilities location. *Geograph Anal* 2:30–42. <https://doi.org/10.1111/j.1538-4632.1970.tb00142.x>
- SCIP (last accessed November 13, 2022a) When should I implement a constraint handler, when should I implement a separator? <https://www.scipopt.org/doc/html/FAQ.php#conshdlrvsseparator>
- SCIP (last accessed November 13, 2022b) SCIPcreateCons(). [https://www.scipopt.org/doc/html/group\\_\\_PublicConstraintMethods.php#ga8d771b778ea2599827e84d01c0aceaf2](https://www.scipopt.org/doc/html/group__PublicConstraintMethods.php#ga8d771b778ea2599827e84d01c0aceaf2)
- Tamir A (2001) The  $k$ -centrum multi-facility location problem. *Discrete Appl Math* 109(3):93–307. [https://doi.org/10.1016/S0166-218X\(00\)00253-5](https://doi.org/10.1016/S0166-218X(00)00253-5)
- Wolf C, Fábian CI, Koberstein A, Suhl L (2014) Applying oracles of on-demand accuracy in two-stage stochastic programming-A computational study. *Eur J Oper Res* 239(2):437–448. <https://doi.org/10.1016/j.ejor.2014.05.010>

## Authors and Affiliations

Luisa I. Martínez-Merino<sup>1,2</sup> · Diego Ponce<sup>2,3</sup>  · Justo Puerto<sup>2,3</sup>

Luisa I. Martínez-Merino  
luisa.martinez@uca.es

Justo Puerto  
puerto@us.es

<sup>1</sup> Departamento de Estadística e Investigación Operativa, Universidad de Cádiz, Cádiz, Spain

<sup>2</sup> Instituto de Matemáticas de la Universidad de Sevilla (IMUS), Sevilla, Spain

<sup>3</sup> Departamento de Estadística e Investigación Operativa, Universidad de Sevilla, Sevilla, Spain