Les Cahiers du GERAD ISSN: 0711–2440

Ranking decomposition for the discrete ordered median problem

M. Cherkesly, C. Contardo, M. Gruson

G–2023–06

March 2023

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Ranking decomposition for the discrete ordered median problem

Marilène Cherkesly [a, b, c]

Claudio Contardo [d, c]

Matthieu Gruson [a, b]

[a] *Département d'analytique, opérations et technologies de l'information, Université du Québec à Montréal, Montréal (Qc), Canada, H3C 3P8*

[b] *CIRRELT, Montréal (Qc), Canada, H3T 1J4*

[c] *GERAD, Montréal (Qc), Canada, H3T 1J4*

[d] *Department of Mechanical, Industrial and Aerospace Engineering, Concordia University, Montréal (Qc), Canada, H3G 1M8*

marilene.cherkesly@gerad.ca
claudio.contardo@gerad.ca
gruson.matthieu@uqam.ca

March 2023
Les Cahiers du GERAD
G–2023–06

Abstract : Given a set $\mathcal{N}$ of size n, a non-negative, integer-valued distance matrix D of dimensions $n \times n$, an integer $p \in \mathbb{N}$ and an integer-valued weight vector $\lambda \in \mathbb{Z}^n$, the discrete ordered median problem (**DOMP**) consists of selecting a subset $\mathcal{C}$ of exactly p points from $\mathcal{N}$ (also referred to as the *centers*) so as to: 1) assign each point in $\mathcal{N}$ to its closest center in $\mathcal{C}$; 2) rank the resulting distances (between every point and its center) from smallest to largest in a sorted vector that we denote d^*; 3) minimize the scalar product $\langle \lambda, d^* \rangle$. The **DOMP** generalizes several classical location problems such as the p-center, the p-median and the obnoxious median problem. We introduce an exact branch-and-bound algorithm to solve the **DOMP**. This branch-and-bound decouples the ranking attribute of the problem to form a series of simpler subproblems which are solved using innovative binary search methods. We consider several acceleration techniques such as warm starts, primal heuristics, variable fixing and symmetry breaking. We perform a thorough computational analysis and show that the proposed method is competitive against several MIP models from the scientific literature. We also comment on the limitations of our method and propose avenues of future research.

Keywords : Discrete ordered median problem, ranking decomposition, p-center problem, p-median problem, branch-and-bound

Acknowledgements: This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Discovery Grants 2017–06106, 2020–06311 and 2021–03327. This support is gratefully acknowledged.

1 Introduction

In the discrete ordered median problem (**DOMP**), we are given a set of nodes $\mathcal{N}$ of size n, an integer-valued, non-negative distance (or dissimilarity) matrix D of dimensions $n \times n$, a natural number p and a integer-valued vector λ of size n, usually referred to as the *weights*. The problem consists of selecting a subset of nodes $\mathcal{C} \subset \mathcal{N}$, referred to as the set of centers, of size exactly p such that: 1) every node in $\mathcal{N}$ is assigned to its closest center in $\mathcal{C}$; 2) the resulting assignment distances are sorted from smallest to largest in a vector d^*; 3) and the scalar product $\langle \lambda, d^* \rangle$ is minimized.

The **DOMP** as introduced by Nickel (2001) plays an unifying role in determining models and algorithms with the potential to tackle multiple classes of problems at once. It is a generalization of the vertex p-center problem (**vPCP**) (Kariv and Hakimi, 1979) by using the unitary weight vector $\lambda_{\mathbf{vPCP}} = (0, \ldots, 0, 1)$ such that the largest assignment distance is minimized. The p-median problem (**pMP**) (Hakimi, 1964, 1965), on the other hand, can be derived from the **DOMP** by considering a weight vector $\lambda_{\mathbf{pMP}} = (1, \ldots, 1)$. The k-centrum problem (**kCP**) (Slater, 1978) can be modeled using a weight vector $\lambda_{\mathbf{kCP}} = (0, \ldots, 0, 1, \ldots, 1)$ where the ones appear at the last k positions only such that the k-largest assignment distances are minimized. The obnoxius p-median problem (**OpMP**) (Hansen and Cohon, 1981) seeks to maximize the sum of the distances between every node in $\mathcal{N}$ and its closest center, and resorts to selecting $\lambda_{\mathbf{OpMP}} = (-1, \ldots, -1)$.

The **DOMP** (and in particular the **vPCP**, the **pMP**, the **kCP** and the **OpMP**) is relevant in practice as its multiple applications in multiple domains demonstrate. The problem of designing a network of solid-waste landfills can for instance be modeled as a **pMP** (Antunes, 1999) and has been used to support the deployment of such a system in Portugal. The problem of designing a lens to correct double vision problems with time-of-day considerations can be modeled using location theory, in particular **vPCP** or **pMP** models (Francis, 2009). The problem of placing sensors for the detection of pollution agents in drinkable water has been modeled as a **pMP** and used to design a contamination warning system by the Environmental Protection Agency (EPA) in the United States (Murray et al., 2009). Romeijn et al. (2006) use a **kCP** criterion as an approach to determine the dosage in a radiation treatment for cancer patients as a proxy to produce a treatment plan with low risk. In Cappanera et al. (2003); Erkut and Neuman (1989) we can find applications of the **OpMP** to the location of undesirable facilities such as garbage dumpsters or chemical plants.

A few variants of the **DOMP** have also been studied in the literature. Puerto and Rodríguez-Chía (2015) generalized further the **DOMP** by introducing a modeling framework that encompasses continuous and discrete variants of these problems, and revisited it later in Puerto and Rodríguez-Chía (2019). Other variants of the **DOMP** include the capacitated **DOMP** (Puerto, 2008; Kalcsics et al., 2010; Espejo et al., 2021) where each facility has a maximal coverage capacity and the **DOMP** with induced orders where two different types of facilities are considered (Domínguez and Marín, 2020).

Many algorithms have been developed to solve the **DOMP**. In terms of exact methods, several mixed-integer programming models —each of which exploiting a specific characteristic of the problem— have been proposed in the literature. Boland et al. (2006) introduce two mixed-integer linear formulations for the **DOMP** with a quadratic number of constraints and between a quadratic and cubic number of variables. In Labbé et al. (2017), the authors introduce two new formulations for the problem with better numerical properties. In particular, the authors introduce a large model providing very tight integrality gaps; and another, more compact model, providing reasonable bounds but in much shorter computing times and less memory consuming. Marín et al. (2020) present telescopic models valid under the assumption that the weight vectors λ are non-decreasing. The proposed models are very compact and provide tight integrality gaps, allowing for substantial speedups with respect to previous models. In addition, *ad-hoc* exact algorithms, such as branch-and-bound (Boland et al., 2006) and branch-and-price (Deleplanque et al., 2020), have been implemented and used to solve small-sized and medium-sized problems of the **DOMP** (less than 50 and 400 nodes, respectively). To the best of our knowledge, the state-of-the-art method for the exact solution of the **DOMP** is

the branch-and-price of Deleplanque et al. (2020), capable of solving instances with up to 50 nodes within a few hours of computing time. The same authors also show that on larger instances (involving up to 400 nodes), their branch-and-price method consumes less memory than a traditional branch-and-cut algorithm derived from the model proposed by Labbé et al. (2017) which is based on *weak order* constraints. While the focus of our paper concerns exact methods, several non-exact methods which rely on approximation algorithms (Byrka et al., 2018; Aouad and Segev, 2019), population-based algorithms (Domínguez-Marín et al., 2005; Stanimirović et al., 2007), variable neighborhood search algorithms (Domínguez-Marín et al., 2005; Puerto et al., 2014; Olender and Ogryczak, 2019), and GRASP (Deleplanque et al., 2020) have been developed.

From an algorithmic perspective, the **DOMP** is not sufficiently well understood. In particular, the existing literature on the **DOMP** offers models and algorithms whose performance strongly depends on the magnitudes of the matrix D or the weight vectors λ Espejo et al. (see e.g., 2021). These models also suffer severe scalability issues when the magnitudes associated with λ or D are not tightly bounded. Mitigating this undesirable effect lies at the core of the objectives pursued in this manuscript.

In this article, we introduce the **DOMP**(k) which consists of selecting p centers from $\mathcal{N}$ so as to minimize $\lambda_k \cdot d_k^*$, where d_k^* is the k-th shortest distance after assigning every node in $\mathcal{N}$ to its closest center. We propose two binary search methods to solve the problem for the case $\lambda_k > 0$ and for the case $\lambda_k < 0$. We show that for $\lambda_k > 0$, this problem resorts to solving a series of minimum partial covering problems, while for $\lambda_k < 0$, the problem resorts to solving a series of maximum partial packing problems. The minimum partial covering problem aims at covering a minimum number of points, i.e., covering points within a radius r of an opened facility, the set of opened facilities being a result of the optimization problem. The maximal partial packing problem aims at covering a maximum number of points, i.e., covering points within a radius r of an opened facility, the set of opened facilities being a result of the optimization problem. In both cases, the problems solved assume a fixed radius r, and we denote them as **MPC**(k,r) and **MPP**(k,r), respectively. Finally, we embed these binary search methods are embedded into a branch-and-bound algorithm to solve the **DOMP** which differs from typical branch-and-bound as it does not solve a linear relaxation at each node of the branching tree.

Our contributions can be summarized as follows:

1. We derive two methods to tackle the **DOMP**(k), one for the case $\lambda_k > 0$, and another for the case $\lambda_k < 0$. The algorithms solve a logarithmic number of **MPC**(k,r) or **MPC**(k,r) in a binary search fashion.
2. We derive a valid dual bound for the **DOMP** from solving a series of problems **DOMP**(k), $k = 1 \dots n$.
3. We embed this dual bounding procedure within a branch-and-bound algorithm, for which we propose several acceleration techniques such as warm starts, variable fixing, primal heuristics, and symmetry breaking.
4. We propose new benchmark instances and conduct a thorough computational campaign to assess the quality of the proposed method.

The remainder of this article is organized as follows. In Section 2, we propose the two binary search algorithms to solve **DOMP**(k) for the case $\lambda_k < 0$ and $\lambda_k > 0$. We also describe the **MPC**(k,r) and the **MPP**(k,r) and explain how they are embedded into our binary search algorithms. In Section 3, we explain our branch-and-bound algorithm. We introduce new benchmark instances and perform extensive numerical experiments for the **DOMP**(k) and the **DOMP** in Section 4. We also discuss the limits of our branch-and-bound algorithm. Finally, conclusions are drawn in Section 5.

2 Binary search methods for the DOMP(k)

The **DOMP**(k) consists in solving a **DOMP** where the weight vector λ is replaced by $\lambda_k e_k$, where e_k is the vector of size n composed of zeroes in all positions except at position k where it takes the value 1.

In the following, we propose two binary search methods to solve the **DOMP**(k). These binary search methods are not impacted by the positions k for which λ_k are equal to zero, but are impacted by the positive or negative value for the non-zero λ_k. In particular, each binary search is specific either for the case $\lambda_k < 0$ or for the case $\lambda_k > 0$. The proposed algorithms assume that the matrix D contains only integer values as specified in Section 1. In the case where the matrix D would not be integral, the values in the matrix could then be multiplied by a large factor and rounded to the nearest integer. Our algorithms would then provide an ϵ-optimal solution for a certain $\epsilon > 0$.

2.1 The case $\lambda_k > 0$

In this section, we propose a binary search method to solve the **DOMP**(k) for the case $\lambda_k > 0$. We first show that the problem is equivalent to finding the smallest radius $r > 0$ such that **MPC**(k,r) equals `TRUE`, i.e., admits a feasible solution, and propose a mathematical model to solve the **MPC**(k,r). Then, we explain how this mathematical model is inserted into a binary search method to solve the **DOMP**(k).

For the case $\lambda_k > 0$, the **DOMP**(k) consists of determining whether there exists a set of p centers $\mathcal{C} \subset \mathcal{N}$ such that, upon assigning nodes in $\mathcal{N}$ to their closest centers, the k-th smallest distance d_k is minimized. This is equivalent to finding the smallest radius $r > 0$ such that its corresponding minimum partial covering problem, i.e., **MPC**(k,r), equals `TRUE`. Note that the **MPC**(k,r) is equivalent to that of deciding whether it is possible to cover *at least* k nodes in $\mathcal{N}$ using at most p balls of radius r centered at nodes in $\mathcal{N}$. If such a solution exists, then the problem is feasible and an algorithm used to solve it would return `TRUE`. Figure 1 illustrates a feasible solution of the **MPC**(k,r) for $k = 9$ and a certain radius r, on an instance with $n = 10$ and $p = 2$. In the figure, the blue, red, and green nodes represent the centers in $\mathcal{C}$, the non-covered nodes in $\mathcal{N}$, and the covered nodes in $\mathcal{N}$, respectively. The radius of the black circle around each center represents the radius r. The matrix D is computed using the Euclidean distances between each pair of nodes.

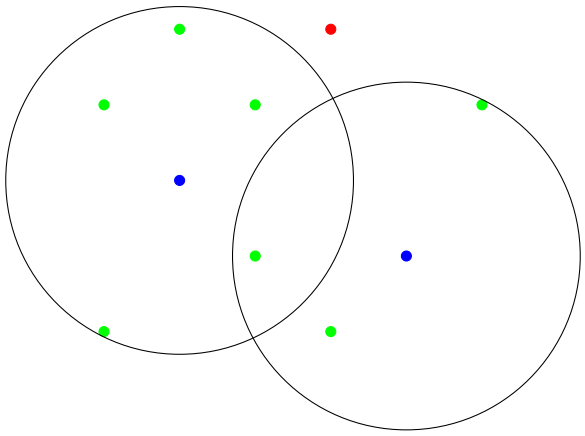

Figure 1: Feasible solution for MPC(9,r), $\lambda_k > 0$ on an instance with $n = 10$ and $p = 2$

To formulate the **MPC**(k,r), we propose a mixed-integer program using the model introduced in Cordeau et al. (2019) for the maximum set-covering problem. This model considers a linear number of variables and constraints. Let D_{ij} be the distance between nodes i and j. For a fixed $r \geq 0$, we define, for every $i \in \mathcal{N}$, $N(i,r) = \{j \in \mathcal{N} : D_{ij} \leq r\}$, which is the set of nodes in $\mathcal{N}$ that lie within a distance of r from i. For every $i \in \mathcal{N}$, let x_i be a binary variable that takes the value 1 if and only if a center is located at node i. Also, let y_i be a continuous variable taking value between 0 and 1 that takes the value 1 if node i is covered by a center. Problem **MPC**(k,r) is equivalent to solving the

following MIP:

$$\min \quad \sum_{i \in \mathcal{N}} x_i \tag{1}$$

subject to

$$\sum_{i \in \mathcal{N}: j \in N(i,r)} x_i - y_j \geq 0, \qquad j \in \mathcal{N}, \tag{2}$$

$$\sum_{j \in \mathcal{N}} y_j \geq k, \tag{3}$$

$$x \in \{0,1\}^n, \tag{4}$$

$$y \in [0,1]^n. \tag{5}$$

Proposition 1. *Model (1)-(5) admits a feasible solution of value $\leq p$ if and only if* ***MPC(k,r)*** *equals* `TRUE`.

Proof. Let (x^*, y^*) be a feasible solution of value $q \leq p$ obtained when solving model (1)– (5). Constraints (3) imply that at least k variables y_j^* take the value one. In turn, constraints (2) imply that for each such j there exists at least one variable x_i^* that takes the value one and covers node j. Therefore, **MPC**(k,r) equals `TRUE`. Now, let us assume the existence of a set $\mathcal{C}$ of cardinality $q \leq p$ that solves **MPC**(k,r). It is easy to see that by assigning x_i^* equal to one for these centers and by assigning every node $j \in \mathcal{N}$ to its closest center, we will have at least k nodes for which constraint (2) holds. □

By embedding model (2)–(5) in a binary search algorithm, we can then solve the **DOMP**(k). Let us define r^L and r^U as a lower and an upper bound on the value of that radius r. Our binary search algorithm starts by initializing $r^L = D_{MIN}$ and $r^U = D_{MAX}$, where $D_{MIN} = \min\{D_{ij} : i, j \in \mathcal{N}\}$ and $D_{MAX} = \max\{D_{ij} : i, j \in \mathcal{N}\}$ are the minimal and maximal values in matrix D. **MPC**(k,r) is then solved using $r = \lfloor (r^L + r^U)/2 \rfloor$. If there is a feasible solution, then this implies that the optimal solution value of **DOMP**(k) is at most $\lambda_k r$, and we set $r^U = r$. Otherwise, this implies that the optimal solution value is more than $\lambda_k r$ and we increase the value of r^L to the next minimal value in matrix D. This procedure is repeated until $r^L = r^U$, and when this is reached we have an optimal solution to **DOMP**(k). Algorithm 1 presents a pseudo-code of our binary search method.

Algorithm 1 Optimal binary search method for DOMP(k), $\lambda_k > 0$

```
Initialize r^L ← D_MIN, r^U ← D_MAX
Initialize x ← ∅, y ← ∅
while r^L < r^U do
   Let r ← ⌊(r^L + r^U)/2⌋
   if MPC(k,r) returns a feasible solution (x*, y*) then
      Let x ← x*, y ← y*
      Let r^U ← r
   else
      Let r^L ← min{D_ij : i, j ∈ N, D_ij ≥ r + 1}
   end if
end while
return (x, y) and λ_k r^U
```

2.2 The case $\lambda_k < 0$

In this section, we propose a binary search method to solve **DOMP**(k) for the case $\lambda_k < 0$. We first show that this problem is equivalent to finding the largest radius $r > 0$ such that **MPP**($k-1$,$r-1$) equals `TRUE`. Then, we propose a mathematical model to solve **MPP**(k,r) for given values of $k = 1 \ldots n$ and $r \geq 0$. Finally, we embed this mathematical model into a binary search method which provides an optimal solution to **DOMP**(k).

For the case $\lambda_k < 0$, the **DOMP**(k) consists of determining if there exists a subset of at least p centers $\mathcal{C}$ such that upon assigning every node in $\mathcal{N}$ to its closest center, the $(k-1)$-th smallest distance d_{k-1} satisfies $d_{k-1} \geq r-1$. This is equivalent to finding the largest radius $r > 0$ such that its corresponding maximum partial packing problem, i.e., **MPP**$(k-1,r-1)$, equals `TRUE`. Note that the **MPP**(k,r) is equivalent to deciding whether it is possible to cover *at most* k nodes from $\mathcal{N}$ with balls of radius r centered on at least p centers $\mathcal{C}$. If such a solution exists, then the problem is feasible and its solution algorithm would return `TRUE`.

We formulate **MPP**(k,r) using a similar notation to the one introduced in Section 2.1, but define the y variables as binary variables which are equal to 1 if node i is covered by a center, and 0 otherwise. We can then model **MPP**(k,r) using the following binary program:

$$\max \quad \sum_{i \in \mathcal{N}} x_i \tag{6}$$

subject to

$$\sum_{i \in \mathcal{N}: j \in N(i,r)} x_i - |N(j,r)| y_j \leq 0, \qquad j \in \mathcal{N}, \tag{7}$$

$$\sum_{j \in \mathcal{N}} y_j \leq k, \tag{8}$$

$$x \in \{0,1\}^n, \tag{9}$$

$$y \in \{0,1\}^n. \tag{10}$$

Proposition 2. *Model (6)–(10) admits a feasible solution of value $\geq p$ if and only if* ***MPP****(k,r) equals* `TRUE`*.*

Proof. Let (x^*, y^*) be a feasible solution of value $q \geq p$ obtained when solving model (6)–(10). Constraints (8) imply that at least $n-k$ variables y_j^* take the value zero. In turn, constraints (7) imply that for each node j such that $y_j^* = 0$, there are no selected nodes in $\mathcal{N}$ that can cover it. Now, let us assume the existence of a set $\mathcal{C}$ of cardinality $q \geq p$ that solves **MPP**(k,r). It is easy to see that by assigning x_i^* equal to one for these centers and by assigning every node $j \in \mathcal{N}$ to their closest center, we will find at most k nodes in $\mathcal{N}$ covered. □

One can realize that a solution for which $q \geq p$ centers are located can be reduced to a solution containing exactly p centers which would remain valid for **MPP**(k,r) because it would necessarily cover less nodes in $\mathcal{N}$. Therefore, constraints (7) can be strengthened as follows:

$$\sum_{i \in \mathcal{N}: j \in N(i,r)} x_i - n(j,r) y_j \leq 0, \qquad j \in \mathcal{N}, \tag{11}$$

where $n(j,r) = \min\{p, |N(j,r)|\}$. In the remainder of this paper, we will use model (6), (8)–(11), to refer to **MPP**(k,r).

By embedding model (6), (8)–(11) into a binary search algorithm, we can then solve **DOMP**(k). Similarly to Algorithm 1, this algorithm starts by initializing $r^L = D_{MIN}$ and $r^U = D_{MAX}$. **MPP**$(k-1,r-1)$ is then solved using $r = \lceil (r^L + r^U)/2 \rceil$. If there is a feasible solution, then this implies that the optimal solution value of **DOMP**(k) is at least $\lambda_k r$, and we set $r^L = r$. Otherwise, this implies that the optimal solution value is less than $\lambda_k r$ and we decrease the value of r^U to the next maximal value in matrix D. This procedure is repeated until $r^L = r^U$, and when this is reached we have an optimal solution to **DOMP**(k). Algorithm 2 presents a pseudo-code of our binary search method.

Algorithm 2 Optimal binary search method for DOMP(k), $\lambda_k < 0$

Initialize $r^L \leftarrow D_{MIN}, r^U \leftarrow D_{MAX}$
Initialize $x \leftarrow \emptyset, y \leftarrow \emptyset$
while $r^L < r^U$ **do**
  Let $r \leftarrow \lceil (r^L + r^U)/2 \rceil$
  if MPP($k-1$,$r-1$) returns a feasible solution (x^*, y^*) **then**
    Let $x \leftarrow x^*$, $y \leftarrow y^*$
    Let $r^L \leftarrow r$
  else
    Let $r^U \leftarrow \max\{D_{ij} : i, j \in \mathcal{N}, D_{ij} \leq r-1\}$
  end if
end while
return (x, y) and $\lambda_k r^L$

2.3 Sensitivity to the magnitudes of the input parameters

We would like to highlight that the method proposed to solve **DOMP(k)** is designed to be little sensitive to the magnitudes of the input parameters D and λ. Indeed, the solution of each problem **DOMP(k)** is agnostic to the magnitude of λ_k, whose value is only used at the end of the corresponding binary search method to compute and return the optimal value. Moreover, the binary search methods perform $O(\log(D_{MAX}))$ computations of the problems **MPC(k,r)** or **MPP(k,r)** —where $D_{MAX} = \max\{D_{uv}, u, v = 1 \dots n\}$—, making the overall algorithm very little sensitive to the value of D_{MAX} as well.

3 A branch-and-bound method for the DOMP

In this section we present a branch-and-bound algorithm for the **DOMP**. This branch-and-bound differs from typical branch-and-bound algorithms where a linear relaxation of the problem is solved at each node of the tree. To compute the dual bound, we solve a series of **DOMP(k)**, and compute the dual bound as the sum of the optimal value of each **DOMP(k)**. The primal bound, on the other hand, is computed by using the solution for **DOMP(k)** which returns the best objective function value for **DOMP**. Because we do not solve a linear relaxation of the problem at each node, the fractional solution is computed as an average of the x-variables obtained when solving the **DOMP(k)**. If the resulting solution is not fractional, then we prune the branch. Otherwise, we branch on the x-variables. In this branch-and-bound, we decouple the decisions of locating the p centers with minimizing the ranked distances which removes a complexity of the **DOMP**. In the following, we first define the dual and primal bounds. Then, we describe our branching mechanism which includes the solution at a branching node, the branching strategy, and variable fixing. Finally, we propose improvement strategies to improve the quality of the primal bound and to reduce the computational time required at each branching node.

3.1 Dual and primal bounds

At each branching node, we solve a series of **DOMP(k)** for different values of k. For each of these problems, we keep its optimal solution for the x-variables, i.e., the location of the centers, denoted as $\mathbf{x}_k^*$ as well as the optimal solution value z_k^*. Using the obtained optimal solutions as well as the optimal solution values, we can derive dual and primal bounds. We hereby detail how these are computed.

Dual bound of the DOMP

Let $I = \{k = 1 \dots n : |\lambda_k| > 0\}$ be the set of indices associated to a non-zero weight. Let z_k be the optimal value associated with each **DOMP(k)**, $k \in I$, and let us define

$$z_D = \sum_{k \in I} z_k. \tag{12}$$

Proposition 3. *z_D as computed using Equation (12) is a valid dual bound for the **DOMP**.*

Proof. Let us consider a subset $\mathcal{C}^*$ of optimal locations for the **DOMP**, and let $d_1^* \leq \cdots \leq d_n^*$ be the associated sorted vector of distances of nodes to centers. The optimal value of **DOMP** is then equal to $z^* = \sum\{\lambda_k d_k^* : k \in I\}$. Let us consider now **DOMP**(k) for $k \in I$. The same solution $\mathcal{C}^*$ is also feasible for this problem, whose value we denote by z_k^*. It follows that $z_k^* \leq \lambda_k d_k^*$ for all $k \in I$. This in turn implies that $z_D = \sum\{z_k^* : k \in I\} \leq \sum\{\lambda_k d_k^* : k \in I\} = z^*$. □

Primal bound of the DOMP

Let us remark that any subset of p centers is feasible for the **DOMP**. Therefore, solving each **DOMP**(k) yields a feasible solution for the **DOMP** as the obtained solutions are integer. The primal bound can then be obtained by computing the **DOMP**-cost associated with each of these solutions and keeping the best cost for the **DOMP** as primal bound.

3.2 Branching

Upon computing the different solutions $\mathbf{x}_k^*$ for every $k \in I$, we verify if the values of the dual and primal bounds coincide. If they do, we may prune the current branching node and declare the node as integral. If they do not, it necessarily means that some of the $\mathbf{x}_k^*$ differ, and must proceed to branch. We do branch on one x-variable, i.e., x_j, and create two children associated to the disjunction $[x_j \leq 0], [x_j \geq 1]$. Our branching scheme uses *reliability branching* which mixes strong branching with pseudo-cost branching (see Achterberg et al., 2005, for the detailed methodology). We also resort to variable fixing to improve the performance of the branch-and-bound algorithm. The branching tree is explored in a best-first fashion. In the following, we define how we compute the solution, provide an overview of our *reliability branching*, and explain our proposed variable fixing mechanism.

Solution of a branching node

At each branching node, we define $\mathbf{x}^*$ as the solution of the branching node. This solution is computed as an average of the **DOMP**(k) solutions, that is,

$$\mathbf{x}^* = \frac{1}{|I|} \sum_{k \in I} \mathbf{x}_k^*. \tag{13}$$

Reliability branching

Throughout our branching tree, we keep a vector of pseudo-costs $(\rho_j^L, \rho_j^U)_{j=1}^n$. The pseudo-costs of each x-variable are initialized to zero (or equivalently uninitialized). A parameter $\Delta > 0$ is used to determine whether pseudo-costs are declared as *reliable* or not. More specifically, the pseudo-costs for a variable x_j will be declared as being *reliable* if $\min\{\rho_j^L, \rho_j^U\} \geq \Delta$. At each branching node associated with a fractional solution, we select the five x-variables which are the closest to 0.5. When selecting a fractional value x_j as candidate for branching, two situations may occur: 1) all the pseudo-costs are reliable; or 2) at least one of the pseudo-costs is declared as unreliable and triggers a recomputation. Let us define z_D as the dual bound of a branching node and $\tilde{x}_j$ as the value of variable x_j as computed with Equation (13) at a given branching node. The branches $[x_j \leq 0]$ and $[x_j \geq 1]$ are created and solved. For each branch, we obtain a new dual bound, i.e., referred to as z_D^0 and z_D^1 for the branches $[x_j \leq 0]$ and $[x_j \geq 1]$, respectively. The pseudo-costs are then computed and updated as

$$\rho_j^L = (z_D^0 - z_D)/\tilde{x}_j \qquad [x_j \leq 0] \tag{14}$$

$$\rho_j^U = (z_D^1 - z_D)/(1 - \tilde{x}_j) \qquad [x_j \geq 1]. \tag{15}$$

If the pseudo-costs are deemed as reliable, they are not recomputed but instead used as a score system to select variables in future iterations. Once the pseudo-costs of all the five candidate x-variables are available (either because they are declared as reliable or recomputed), we then select the variable to branch on in the following manner. If one of the candidate x-variables allows one to prune one of the branches (this can only be done when the children nodes are solved entirely), we then select that variable and prune the offending branch. Otherwise, we branch on the variable x_j such that $\min\{\rho_j^L, \rho_j^U\}$ is the largest. In case of ties, we select the variable x_j such that $\rho_j^L + \rho_j^U$ is the largest. Further ties are broken arbitrarily. Note that every time that we decide to branch on a variable for which the pseudo-costs were deemed as reliable, the children nodes will have to be solved, and the resulting values on both branches used to update the pseudo-costs, as an average of the current values and the ones obtained using Equations (14)–(15).

Variable fixing

When creating the two child nodes, i.e., $x_j \leq 0$ and $x_j \geq 1$, we resort to variable fixing in the corresponding problems **MPC(k,r)** and **MPP(k,r)** in order to achieve a faster convergence. Let us denote L, U the set of low ($x_j \leq 0$) and up ($x_j \geq 1$) branching decisions for a given branching node. Branching constraints of the form $x_j \leq 0$ are used to omit the associated variables from the problems **MPC(k,r)**, **MPP(k,r)**. Branching constraints of the form $x_j \geq 1$ are used to also omit the associated variables and all the nodes within a radius of r from them, and to replace p by $p - |U|$. Let κ be the number of nodes that are covered by the branching decisions fixed to one. The MIPs are then modified by replacing k by $k - \kappa$ and can be trivially solved when $\kappa \geq k$ or when $|U| \geq p$.

For the **MPC(k,r)** dominance is also used to reduce the number of potential locations. Given that we want to cover as many nodes as possible with as little locations as possible, one may simply omit from the optimization model (fix to zero), any center i if there is another center j that covers a superset of the nodes covered by center i. We refer the reader to Chen and Chen (2009) for similar ideas applied to the p-center problem.

3.3 Improvement strategies

In the following, we describe three improvement strategies. The first strategy allows to improve the quality of the primal bound at the root node. The second strategy allows to improve the performance of the algorithm when computing the solutions at each branching node. The third strategy proposes a warm-start for each branching node.

An iterated local search routine for the primal bound

At the root node, the primal bound as defined in Section 3.1 is improved with an iterated local search (`ILS`) routine. Let us first define the `incumbent` as the solution with the best cost for **DOMP** found during our `ILS`. Our `ILS` starts from the solution for **DOMP(k)** with the best cost for **DOMP**. We then perform `swap` moves by swapping between open and closed locations, and evaluate the cost of the resulting solution by computing the ranked distances. The incumbent is updated in a first-improvement fashion, that is, every time an improvement is identified. When no more improvements are identified, i.e., convergence to a local optimum, one `perturbation` move is done. That is, the `incumbent` is perturbed by performing a random `swap` move. The process is then repeated (i.e., improving `swap` and `pertubation` moves) and stops as soon as it reaches five perturbations without being able to improve the `incumbent` solution. The `incumbent` solution is then returned and its **DOMP**-cost is returned as the improved primal bound.

Ordering to solve the DOMP(k)

When solving the **DOMP(k)**, low indices k (close to one) associated with positive values of λ_k usually admit multiple alternate optima, while larger indices (close to n) usually admit less optimal solutions.

Analogously, large indices k (close to n) are bad in terms of symmetries for negative values of λ_k while the opposite occurs for low indices. Therefore, it is important to solve the different **DOMP(k)** is a specific scheme in order to improve the quality of our proposed approach. That is, all indices i are sorted according to a score, s_i, which is computed as follows:

$$s_i = \begin{cases} 0 & \text{if } i \leq p \\ i & \text{if } i > p \text{ and } \lambda_i > 0 \\ n - i & \text{if } i > p \text{ and } \lambda_i < 0. \end{cases} \tag{16}$$

The **DOMP(k)** are then solved in non-increasing order of score, and the solution vectors are stored and used to drive the algorithm towards finding solutions that are close to those encountered for the largest scores. That is, by denoting, $K = \{k_1, \ldots, k_n\}$ as the ordering according to the scores (larger scores first), and $\xi^t = \sum\{\mathbf{x}^*_{k_t} : 1 = 1 \ldots t-1\}$ as the aggregated sum of the solution vectors constructed up to iteration $t-1$ of this sequence, the solution of the t-th problem **DOMP(k_t)** modifies the MIPs (either a series of **MPC(k_t,r)** or of **MPP(k_t,r)**) by putting the objective as a constraint

$$\sum_{i=1}^{n} x_i \leq p, \qquad \text{for the } \mathbf{MPC}(k_t,r) \tag{17}$$

$$\sum_{i=1}^{n} x_i \geq p, \qquad \text{for the } \mathbf{MPP}(k_t,r), \tag{18}$$

and by considering instead the following objective function:

$$\max \qquad \langle \xi^t, x \rangle, \tag{19}$$

where ξ^0 is initialized as $(0, \ldots, 0)$.

The rationale behind this mechanism is the following. By solving the problems associated with higher scores first, if the optimal solutions are not unique, they admit few alternate optima. Problems that are likely to admit more alternate optima are solved last, in which case the objective (19) will drive the model toward encountering optimal solutions which will favor the locations with more occurrences in the previous iterations. This mechanism incentivates the appearance of integer solutions as early as possible in the tree by realizing that if such a solution exists, then it must be available as an optimal solution for all values of k, and in particular for those admitting few optima.

Warm-start for the branching nodes

At each branching node, we use a warm-start to reduce the computational time. As explained, when solving a branching node, we obtain integer solutions for each **DOMP(k)**, i.e., the x-variables either take values 0 or 1. When creating the two child nodes, i.e., branching on $x_j \leq 0$ and $x_j \geq 1$, it is not necessary to solve all **DOMP(k)**. In particular, for the branch $x_j \leq 0$, we do not have to solve the **DOMP(k)** for which $x_j = 0$ in the parent node. Analogously, for the branch $x_j \geq 1$, we do not have to solve the **DOMP(k)** for which $x_j = 1$ in the parent node.

4 Computational experiments

In this section we conduct extensive computational experiments to analyze the performance of our method. First, we describe our experimental setting, including the set of instances and a three state-of-the-art mathematical models that were tested to compare the performance of our proposed algorithms. Second, we show that using our binary search methods to solve **DOMP(k)** is faster than existing algorithms. Third, for the **DOMP**, we analyze the sensitivity of our branch-and-bound as well as existing state-of-the-art mathematical models to the size of the problem (associated with the size of $\mathcal{N}$), the magnitudes of the input parameters (represented by the values $D_{MAX} = \max\{D_{ij} : 1 \leq$

$i < j \leq n\}, \lambda_{MAX} = \max\{\lambda_k : 1 \leq k \leq n\}$), and the topology of the weight vector λ. Fourth, we compare our branch-and-bound algorithm with the state-of-the-art branch-price-and-cut algorithm of Deleplanque et al. (2020). Given the large number of tests performed, in the main manuscript we present aggregate results only.[1]

4.1 Experimental setting

In order to compare the performance of our proposed algorithms (i.e., the binary search methods for **DOMP(k)** and the branch-and-bound for **DOMP**), we have re-implemented three pure MIP models from the model. Note that all of our conclusions are consistent with our experiences, but might not coincide with the conclusions reached by previous authors. On the other hand, our re-implementation is at least as good as the initial implementation (i.e., often faster). The three models are the weak-ordering constraints model (**WOC**) proposed by Labbé et al. (2017), the radius formulation (**DOMP**$_{OTr1}$) proposed by Marín et al. (2020), and a three-index model (**DOMP**$_{OT\theta}$) from the same article. Based on the results reported in Puerto and Rodríguez-Chía (2019), we did not test other models that obtained performance similar to **DOMP**$_{OTr1}$ and **DOMP**$_{OT\theta}$. In addition, for some analysis, we also compare our algorithm with the branch-price-and-cut method of Deleplanque et al. (2020) denoted by **BPAC**. Note that this branch-price-and-cut was not re-implemented. Finally, our binary search methods for the **DOMP(k)** as well as our branch-and-bound method for the **DOMP** are denoted by **CCG23**.

All models have been used with IBM CPLEX 22.1. **WOC**, **DOMP**$_{OTr1}$, and **DOMP**$_{OT\theta}$ have been implemented using C++ CPLEX Concert Technology. **CCG23** has been implemented in Julia v1.7 with JuMP v1.5. The three MIP models and our algorithms have been executed on a computer equipped with an Intel Xeon E5-2637 v2 @3.5GHz processor. Our method is available as a Julia package in the GitHub repository `https://github.com/claud10cv/DiscreteOrderedMedian.jl.git`.[2]

We consider two sets of instances for our tests, one set based on benchmark instances and the second set was created for this paper to conduct thorough sensitivity analyses for **DOMP**. The first set of instances, denoted as D20, has been introduced in Deleplanque et al. (2020) and consists of problems with n ranging between 20 and 400, with values of p ranging between $\frac{1}{4}n$ and $\frac{1}{2}n$. In these problems, the weight vectors λ and the distance matrices D are of low magnitudes (in the tens and hundreds, respectively). The second set of instances, denoted by C23, has been generated for this paper. These instances have a similar distribution for n and p as the D20 dataset, but the weight vectors and distance matrices are of larger magnitudes (in the hundreds and hundreds of thousands, respectively). For each value of n, p and choice of λ (see below), we generate 10 instances. Both datasets can be retrieved online in the GitHub repository `https://github.com/claud10cv/DOMPInstances`.

We also consider two families of weight vectors: unit weight vectors consisting of vectors with a single non-zero entry; and general weight vectors to mimic several classes of classic median-like problems such as the p-center and the p-median. The unit weight vectors are used to assess the performance of our proposed binary search methods to solve the **DOMP(k)**, which are also the subproblems of our branch-and-bound. On the other hand, the general weight vectors are used to assess the performance of our branch-and-bound method to solve the **DOMP**. We hereby describe the two families of weight vectors.

Unit weight vectors We consider eight different types of unit weight vectors. For each value of $k \in \{p+1, \frac{n}{2}, \lceil\frac{3n}{4}\rceil, n\}$ and for each value of $s = \{-1, 1\}$, we consider $\lambda(k, s) = se_k$, i.e., the vector having a coefficient of s at position k.

[1] Detailed results will be made available as an online supplemental material upon a successful clearance of the review process.

[2] The repository is private for now, therefore inaccessible to the public. It will be made public as soon as the article clears the review process with success.

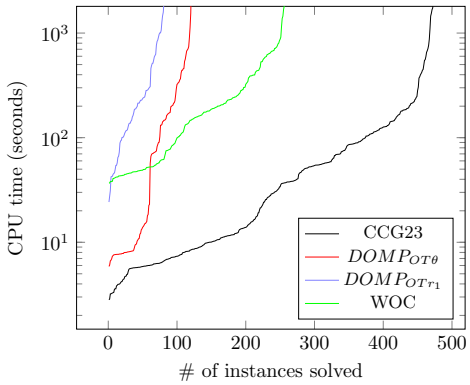

Figure 2: Profile curves on unit weight vectors

General weight vectors We consider 9 types of general weight vectors with different characteristics related to the sparsity (low, medium, high) and the contiguity of the non-zeroes (high, low). Table 1 presents an overview of the types of general weight vectors. For each type of weight vector, we present its notation as it was previously used in the literature, its characteristics (λ) and its name.

Table 1: Characteristics of the tested general weight vectors λ

Notation	λ	Name
T1	(1, 1, . . . , 1, 1)	p-median
T2	(0, 0, . . . , 0, 1)	p-center
T3	(0, 0, . . . , 0, 1, 1, . . . , 1)	k-centrum, $k = n/2$
T4	(0, 0, . . . , 0, 1, . . . , 1, 0, 0, . . . , 0)	k_1-k_2 Trimmed mean, $k_2 - k_1 = n/2$
T6	(0, 1, 0, 1, . . .)	Alternate 0, 1
T7	λ random	Random
T10	(-1, 0, . . . , 0, 1)	Range
T12	(-1,. . . , -1)	Dispersion
T14	(1, -1, 1, -1,. . .)	Alternate 1, -1

4.2 Performance on unit weight vectors

In this section we assess the performance of our method in solving the **DOMP** for unit weight vectors, this is for λs having exactly one single non-zero coefficient. The objective of this experiment is to justify the choice of the proposed binary search algorithms used to solve the subproblems **DOMP**(k), as opposed to using one of the other models considered in our computational study. We restrict our analysis to instances from the dataset D20 with $n = \{100, 200\}$. These instances were selected as the computing times were not too low, nor too high, thus allowing for a good comparison. In total, the testbed consists of 480 instances with different values of n, p, k, and s. For this first experiment, our maximal computational time is set to 30 minutes and our memory limit is set to 5GB of RAM.

Figure 2 reports the profile curves for the different methods in logarithmic scale. The x-axis indicates the number of instances solved (out of the 480 instances), while the y-axis indicates, for a given x, the CPU time required to solve x instances to proven optimality. Our binary search methods outperform all other methods for this family of weight vector. This also justifies using the proposed binary search methods to solve the associated unit weight subproblems in the proposed branch-and-bound method for the case of general weight vectors. We can also notice that model **DOMP**$_{OTr_1}$ is outperformed by all three other models. In a preliminary analysis we also observed that this behavior happened with general weight vectors. Therefore, in the remainder of the computational experiments, we have discarded this model.

4.3 Performance on general weight vectors

In this section we perform an analysis of the proposed branch-and-bound method by comparing its performance against that of the other two MIP models, namely **WOC** and **DOMP**$_{OT\theta}$. For all these experiments, the maximal computational time is set to two hours and our memory limit is set to 5GB of RAM.

We now analyze the overall performance of our method, for all possible values of the different parameters. That is, for every type of general weight vector λ (as described in Table 1), for every instance size (ranging between $n = 20$ and $n = 400$), and for both sets of instances (D20, CCG23). Figure 3 reports the profile graphs of the performance of our proposed branch-and-bound **CCG23** as well as the two MIP models (**DOMP**$_{OT\theta}$, **WOC**). Our method is competitive when compared with **DOMP**$_{OT\theta}$ and allows to solve approximately 200 additional instances within the two-hour time limit. On the other hand, for more difficult instances, i.e., which require more computational time, **WOC** provides a substantially better performance.

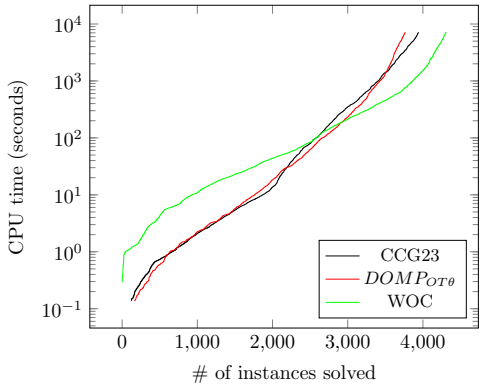

Figure 3: Profile curves on general weight vectors

To better assess the strengths and the limitations of our branch-and-bound algorithm, we conduct sensitivity analyses to grasp the factors influencing the performance of **CCG23**. In the following, we perform sensitivity analyses with respect to the parameters' magnitudes, the problem size n, and the type of weight vector λ. For each analysis, we present two figures: a first figure with the number of instances solved to proven optimality within the two-hour computational time, and a second figure with the average CPU time.

4.3.1 Effect of the parameters' magnitudes

In this section, we analyze the effect of the magnitudes of the input parameters in the performance of the different methods by comparing the performance of the different methods with respect to the two sets of instances (D20 and C23). Figure 4 reports the number of problems solved to proven optimality, and Figure 5 reports the average CPU time. In terms of number of optimal solutions, **WOC** offers the best overall performance, and in particular for instances with parameters of small magnitudes (i.e, D20). On the other hand, our method offers the most robust performance, without showing a significant deterioration on the problems with input parameters of larger magnitudes, as opposed to what we observe for **WOC** and **DOMP**$_{OT\theta}$. In terms of CPU time taken, our method is the fastest for the D20 instances, and the second fastest for the newly generated instances.

4.3.2 Effect of the problem size

In this section, we report for different values of n the number of instances solved to optimality and the average CPU time taken. These results are reported in Figures 6 and 7, respectively. These

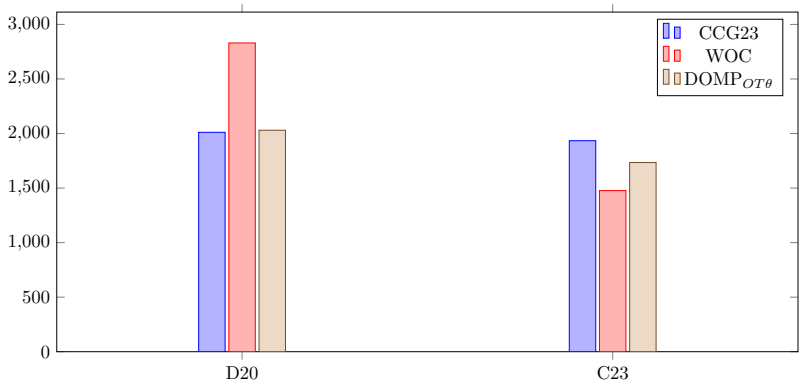

Figure 4: Number of instances solved for different magnitudes of the input parameters

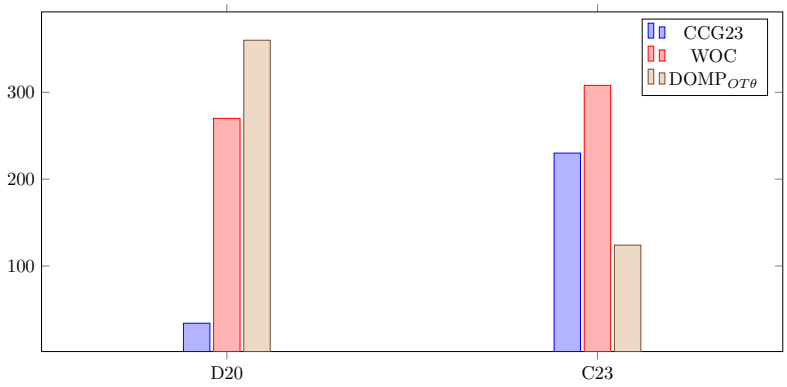

Figure 5: Average CPU times for different magnitudes of the input parameters

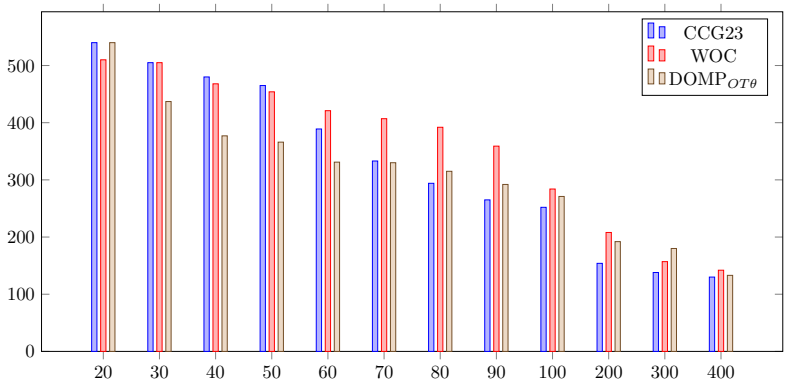

Figure 6: Number of instances solved for varying values of n

results highlight that our branch-and-bound is competitive for low values of n (i.e., $n \leq 50$), but its comparative performance decreases for larger values of n. In addition, while the average CPU time increases as the value of n increases, our method remains the fastest one for most values of n.

4.3.3 Effect of the weight vector λ

We now analyze the performance of each method disaggregated by class of weight vector λ. Figures 8 and 9 report the number of instances solved to proven optimality and the average CPU time disaggregated by class of weight vector. Our branch-and-bound algorithm solves more instances with λs of types T2, T7 and T10, and is second best for λs of types T4, T6 and T12. Also, we notice that

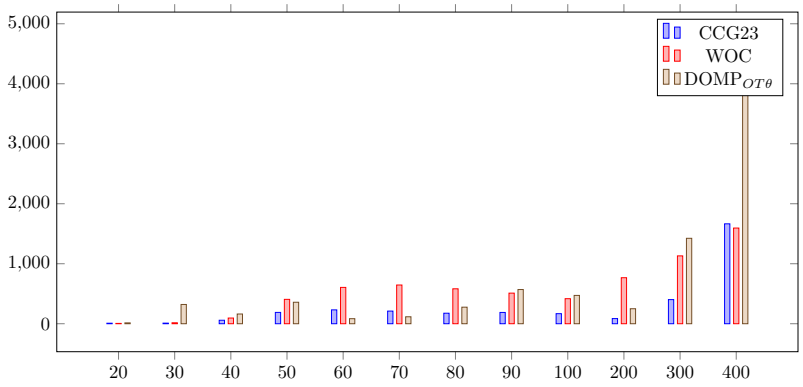

Figure 7: Average CPU times for varying values of n

our method shows a very poor performance for λs of the type T14. Our method is also the fastest with λs of types T2, T4, T6, T7, and T10, while it is the slowest with types T1, and T3. This can be explained by looking at the sparsity of the λs. More precisely, our method is designed to exploit the sparsity of the weight vector λ. Therefore, given that λs of types T2 and T10 are the sparsest (one and two non-zeroes respectively), this confirms that our branch-and-bound is best with sparse λs.

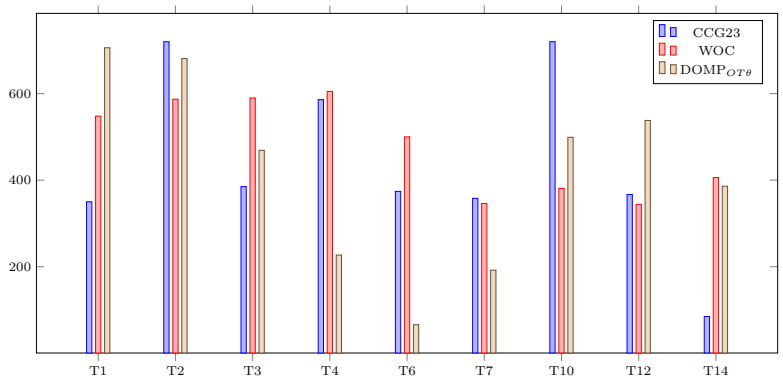

Figure 8: Number of instances solved per type of weight vector λ

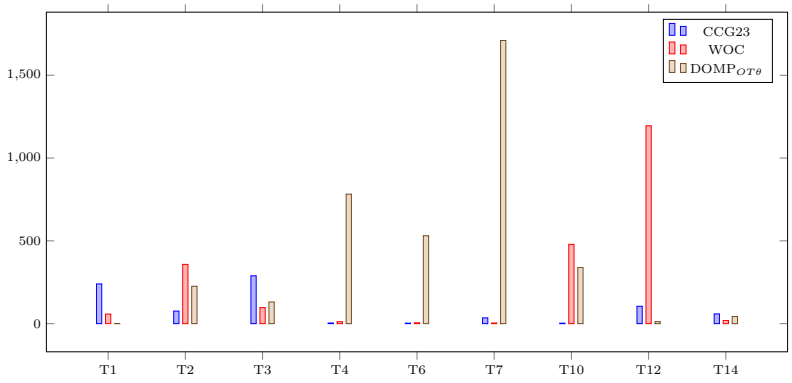

Figure 9: Average CPU times per type of weight vector λ

4.4 Comparison against the branch-price-and-cut method of Deleplanque et al. (2020)

In this section, we compare our branch-and-bound algorithm with the state-of-the-art branch-price-and-cut proposed by Deleplanque et al. (2020) which is denoted by **BPAC**. To compare the two methods, we have normalized their computing times. The results reported in Deleplanque et al. (2020) for **BPAC** are based on runs performed on a machine running an Intel i7 CPU clocked at 2.8 GHz with 4GB of RAM. According to the renown benchmarking website http://www.passmark.com, their machine achieves a score of 2,952, while our machine achieves a score of 6,390. Therefore, we compare their results obtained within a time limit of two hours (7,200 seconds) with our results obtained with a time limit of 3,326 seconds ($7,200 \times 2,952/6,390$). In addition, we restrict our algorithm to the dataset D20 with T7, as these are the only solved instances reported in Deleplanque et al. (2020). Figure 10 reports the number of instances solved by each method within the normalized time limit. Note that no instances with $n \geq 200$ have been solved by both methods and the x-axis is limited to $n = \{20, 30, 40, 50, 60, 70, 80, 100\}$. We can observe that our branch-and-bound algorithm outperforms the **BPAC**. One of the most remarkable selling points of **BPAC** is, as observed by the authors, its consumption of RAM resources which is substantially lower than for the MIP models. We would like to highlight that our method possesses the same characteristic as it never required more than 2GB of RAM for all our computational experiments.

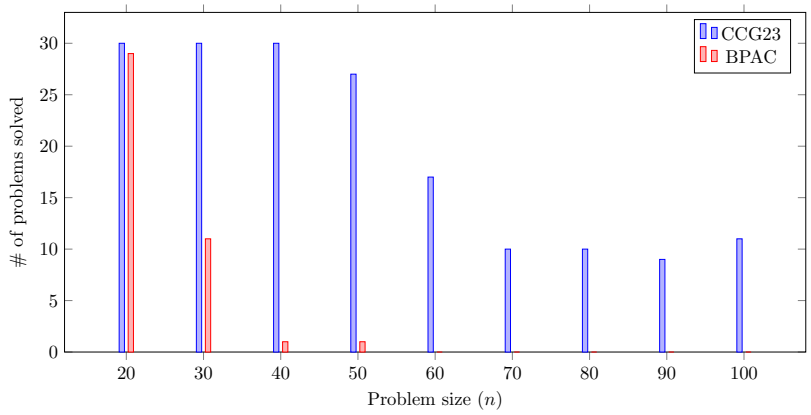

Figure 10: Comparison against the branch-price-and-cut method of Deleplanque et al. (2020)

5 Conclusions

In this paper we proposed two binary search methods to solve the **DOMP**(k) as well as a branch-and-bound algorithm to solve the **DOMP** which resorts on these binary search methods. The branch-and-bound algorithm consists of decoupling the ranking attribute of the problem to compute a dual bound by solving a series of easier problems on unit weight vectors. These problems are solved by means of tailored binary search methods that work by solving a series of covering or packing subproblems. Several acceleration techniques to speed-up the solution process have also been proposed and implemented.

Our computational experiments where tested using a newly generated dataset with input parameters of large magnitudes. We have also conducted an extensive computational campaign on different weight vectors. To the best of our knowledge, this is also the first study to compare different methods on such a broad variety of instances and λ vectors. Our results show that solving unit weight vectors with our proposed binary search methods outperforms existing MIP models. Therefore, these are a good base to solve the subproblems in our branch-and-bound algorithm. For the general weight vectors, our branch-and-bound algorithm seems robust in terms of number of optimal solutions found

and CPU time taken to solve the instances. In general, it is also better than models $\mathbf{DOMP}_{OTr1}$ and $\mathbf{DOMP}_{OT\theta}$ from the literature, as well as the branch-and-price algorithm developed by Deleplanque et al. (2020). Our results indicate that our method performs best on instances with sparse weight vectors, but less on instances with dense weight vectors. In addition, our method is not sensitive with respect to the magnitudes of the input parameters, unlike the other MIP models which show a poorer scalability in this regard. Finally, our proposed method also consumes little memory, although its scalability seems to be negatively affected by an increase in the number of nodes and in the density of the weight vector.

We can identify several potential avenues for future research. First, we believe that addressing the scalability of the algorithm to the size of the problem and to the density of the weight vector is crucial to make it more robust. Second, while our algorithm is designed for the **DOMP**, we believe that the concept of *ranking decomposition* could be applied to other classes of ordered problems. In particular, similar ideas could be used to solve the ordered median hub-location problem (Puerto et al., 2011, 2016), which introduces an ordering criterion to the problem of locating hubs and designing commodity paths in the hub network.

References

T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. Operations Research Letters, 33(1):42–54, 2005.

A. P. Antunes. Location analysis helps manage solid waste in central portugal. Interfaces, 29(4):32–43, 1999. doi: 10.1287/inte.29.4.32. URL https://doi.org/10.1287/inte.29.4.32.

A. Aouad and D. Segev. The ordered k-median problem: surrogate models and approximation algorithms. Mathematical Programming, 177(1):55–83, 2019.

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

J. Byrka, K. Sornat, and J. Spoerhase. Constant-factor approximation for ordered k-median. In Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, pages 620–631, 2018.

P. Cappanera, G. Gallo, and F. Maffioli. Discrete facility location and routing of obnoxious activities. Discrete Applied Mathematics, 133(1–3):3–28, 2003.

D. Chen and R. Chen. New relaxation-based algorithms for the optimal solution of the continuous and discrete p-center problems. Computers & Operations Research, 36(5):1646–1655, 2009.

J.-F. Cordeau, F. Furini, and I. Ljubić. Benders decomposition for very large scale partial set covering and maximal covering location problems. European Journal of Operational Research, 275(3):882–896, 2019.

S. Deleplanque, M. Labbé, D. Ponce, and J. Puerto. A branch-price-and-cut procedure for the discrete ordered median problem. INFORMS Journal on Computing, 32(3):582–599, 2020.

E. Domínguez and A. Marín. Discrete ordered median problem with induced order. Top, 28:793–813, 2020.

P. Domínguez-Marín, S. Nickel, P. Hansen, and N. Mladenović. Heuristic procedures for solving the discrete ordered median problem. Annals of Operations Research, 136(1):145–173, 2005.

E. Erkut and S. Neuman. Analytical models for locating undesirable facilities. European Journal of Operational Research, 40(3):275–291, 1989.

I. Espejo, J. Puerto, and A. M. Rodríguez-Chía. A comparative study of different formulations for the capacitated discrete ordered median problem. Computers & Operations Research, 125:105067, 2021.

R. L. Francis. Location theory helps solve a double-vision problem. Interfaces, 39(6):527–532, 2009. doi: 10.1287/inte.1090.0466. URL https://doi.org/10.1287/inte.1090.0466.

S. L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. Operations research, 12(3):450–459, 1964.

S. L. Hakimi. Optimum distribution of switching centers in a communication network and some related graph theoretic problems. Operations research, 13(3):462–475, 1965.

P. Hansen and J. Cohon. On the location of an obnoxious facility. Sistemi urbani Napoli, (3):299–317, 1981.

J. Kalcsics, S. Nickel, J. Puerto, and A. M. Rodríguez-Chía. The ordered capacitated facility location problem. Top, 18(1):203–222, 2010.

O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. i: The p-centers. SIAM Journal on Applied Mathematics, 37(3):513–538, 1979.

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

A. Marín, D. Ponce, and J. Puerto. A fresh view on the discrete ordered median problem based on partial monotonicity. European Journal of Operational Research, 286(3):839–848, 2020.

R. Murray, W. E. Hart, C. A. Phillips, J. Berry, E. G. Boman, R. D. Carr, L. A. Riesen, J.-P. Watson, T. Haxton, J. G. Herrmann, R. Janke, G. Gray, T. Taxon, J. G. Uber, and K. M. Morley. Us environmental protection agency uses operations research to reduce contamination risks in drinking water. Interfaces, 39 (1):57–68, 2009. doi: 10.1287/inte.1080.0415. URL https://doi.org/10.1287/inte.1080.0415.

S. Nickel. Discrete ordered weber problems. In Operations research proceedings, pages 71–76. Springer, 2001.

P. Olender and W. Ogryczak. A revised variable neighborhood search for the discrete ordered median problem. European Journal of Operational Research, 274(2):445–465, 2019.

J. Puerto. A new formulation of the capacitated discrete ordered median problems with {0, 1}-assignment. In Operations Research Proceedings 2007, pages 165–170. Springer, 2008.

J. Puerto and A. M. Rodríguez-Chía. Ordered median location problems. In Location science, pages 249–288. Springer, 2015.

J. Puerto and A. M. Rodríguez-Chía. Ordered median location problems. In Location Science, pages 261–302. Springer, Cham, 2019.

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

J. Puerto, D. Pérez-Brito, and C. G. García-González. A modified variable neighborhood search for the discrete ordered median problem. European Journal of Operational Research, 234(1):61–76, 2014.

J. Puerto, A. Ramos, A. M. Rodríguez-Chía, and M. C. Sánchez-Gil. Ordered median hub location problems with capacity constraints. Transportation Research Part C: Emerging Technologies, 70:142–156, 2016.

H. E. Romeijn, R. K. Ahuja, J. F. Dempsey, and A. Kumar. A new linear programming approach to radiation therapy treatment planning problems. Operations Research, 54(2):201–216, 2006.

P. J. Slater. Centers to centroids in graphs. Journal of graph theory, 2(3):209–222, 1978.

Z. Stanimirović, J. Kratica, and D. Dugošija. Genetic algorithms for solving the discrete ordered median problem. European Journal of Operational Research, 182(3):983–1001, 2007.