

The Discrete Ordered Median Problem revisited: new formulations, properties and algorithms

Diego Ponce Lopez

▶ **To cite this version:**

Diego Ponce Lopez. The Discrete Ordered Median Problem revisited: new formulations, properties and algorithms. Operations Research [math.OC]. Université Libre de Bruxelles (U.L.B.), Belgium, 2016. English. NNT : . tel-01421868

HAL Id: tel-01421868
https://inria.hal.science/tel-01421868

Submitted on 23 Dec 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.





Universidad de Sevilla
Instituto de Matemáticas de la Universidad de Sevilla (IMUS)
Programa de Doctorado "Matemáticas"

Université Libre de Bruxelles
Faculté des Sciences
Doctorat en Sciences

The discrete ordered median problem revisited: new formulations, properties and algorithms

---

PhD Dissertation

Author
Diego Ponce

Supervisors
Prof. Dr. *Martine Labbé*
Prof. Dr. *Justo Puerto*

July 13, 2016

Abstract

This dissertation studies in depth the structure of the Discrete Ordered Median Problem (DOMP), to define new formulations and resolution algorithms. Furthermore we analyze an interesting extension for DOMP, namely MDOMP (Monotone Discrete Ordered Median Problem).

This thesis is structured in three main parts.

First, a widely theoretical and computational study is reported. It presents several new formulations for the Discrete Ordered Median Problem (DOMP) based on its similarity with some scheduling problems. Some of the new formulations present a considerably smaller number of constraints to define the problem with respect to some previously known formulations. Furthermore, the lower bounds provided by their linear relaxations improve the ones obtained with previous formulations in the literature even when strengthening is not applied. We also present a polyhedral study of the assignment polytope of our tightest formulation showing its proximity to the convex hull of the integer solutions of the problem. Several resolution approaches, among which we mention a branch and cut algorithm, are compared. Extensive computational results on two families of instances, namely randomly generated and from Beasley's OR-library, show the power of our methods for solving DOMP. One of the achievements of the new formulation consists in its tighter LP-bound.

Secondly, DOMP is addressed with a new set partitioning formulation using an exponential number of variables. This chapter develops a new formulation in which each variable corresponds to a set of demand points allocated to the same facility with the information of the sorting position of their corresponding distances. We use a column generation approach to solve the continuous relaxation of this model. Then, we apply a branch-cut-and-price algorithm to solve to optimality small to moderate size of DOMP in competitive computational time.

To finish, the third contribution of this dissertation is to analyze and compare formulations for the monotone discrete ordered median problem. These formulations combine different ways to represent ordered weighted averages of elements by using linear programs together with the p-median polytope. This approach gives rise to two efficient formulations for DOMP under a hypothesis of monotonicity in the lambda vectors. These formulations are theoretically compared and also compared with some other formulations valid for the case of general lambda vector. In addition, it is also developed another new formulation, for the general case, that exploits the efficiency of the rationale of monotonicity. This representation allows to solve very efficiently some DOMP instances where the monotonicity is only slightly lost. Detailed computational tests on all these formulations is reported in the dissertation. They show that specialized formulations allow to solve to optimality instances with sizes that are far beyond the limits of those that can solve in the general case.

Resumen

Este trabajo estudia en profundidad la estructura del problema disctreto de la mediana ordenada (DOMP, por su acrónimo en inglés) con el objetivo de definir nuevas formulaciones y algoritmos de resolución. Además, analizamos una interesante extensión del DOMP conocida como el problema monótono discreto de la mediana ordenada (MDOMP, de su acrónimo en inglés).

Esta tesis se compone de tres grandes bloques.

En primer lugar, se desarrolla un detallado estudio teórico y computacional. Se presentan varias formulaciones nuevas para el problema discreto de la mediana ordenada (DOMP) basadas en su similaridad con algunos problemas de secuenciación. Algunas de estas formulaciones requieren de un cosiderable menor número de restricciones para definir el problema respecto a algunas de las formulaciones previamente conocidas. Además, las cotas inferiores proporcionadas por las relajaciones lineales mejoran a las obtenidas con formulaciones previas de la literatura incluso sin reforzar la nueva formulación. También presentamos un estudio poliédrico del politopo de asignación de nuestra formulación más compacta mostrando su proximidad con la envolvente convexa de las soluciones enteras del problema. Se comparan algunos procedimientos de resolución, entre los que destacamos un algoritmo de ramificación y corte. Amplios resultados computacionales sobre dos familias de instancias -aleatoriamente generadas y utilizando la Beasley's OR-library- muestran la potencia de nuestros métodos para resolver el DOMP.

En el segundo bloque, el problema discreto de la mediana ordenada es abordado con una formulación de particiones de conjuntos empleando un número exponencial de variables. Este capítulo desarrolla una nueva formulación en la que cada variable corresponde a un conjunto de puntos de demanda asignados al mismo servidor con la información de la posición obtenida de ordenar las distancias correspondientes. Utilizamos generación de columnas para resolver la relajación continua del modelo. Después, empleamos un

algoritmo de ramificación, acotación y "*pricing*" para resolver a optimalidad tamaños moderados del DOMP en un tiempo computacional competitivo.

Por último, el tercer bloque de este trabajo se dedica a analizar y comparar formulaciones para el problema monótono discreto de la mediana ordenada. Estas formulaciones combinan diferentes maneras de representar medidas de pesos ordenados de elementos utilizando programación lineal junto con el politopo de la p-mediana. Este enfoque da lugar a dos formulaciones eficientes para el DOMP bajo la hipótesis de monotonía en su vector λ. Se comparan teóricamente las formulaciones entre sí y frente a algunas de las formulaciones válidas para el caso general. Adicionalmente, se desarrolla otra formulación válida para el caso general que explota la eficiencia de las ideas de la monotonicidad. Esta representación permite resolver eficientemente algunos ejemplos donde la monotonía se pierde ligeramente. Finalmente, llevamos a cabo un detallado estudio computacional, en el que se aprecia que las formulaciones *ad hoc* permiten resolver a optimalidad ejemplos cuyo tamaño supera los límites marcados en al caso general.

Résumé

Cette dissertation étudie en profondeur la structure du "*Discrete Ordered Median Problem*" (DOMP), afin de proposer de nouvelles formulations et de nouveaux algorithmes de résolution. De plus, une extension intéressante du DOMP nommée MDOMP ("*Monotone Discrete Ordered Median Problem*") a été étudiée.

Cette thèse a été structurée en trois grandes parties.

La première partie présente une étude riche aux niveaux théorique et expérimentale. Elle développe plusieurs formulations pour le DOMP qui sont basées sur des problèmes d'ordonnancement largement étudiés dans la littérature. Plusieurs d'entres elles nécessitent un nombre réduit de contraintes pour définir le problème en ce qui concerne certaines formulations connues antrieurement. Les bornes inférieures, qui sont obtenues par la résolution de la relaxation linéaire, donnent de meilleurs résultats que les formulations précédentes et ceci même avec tout processus de renforcement désactivé. S'ensuit une étude du polyhèdre de notre formulation la plus forte qui montre sa proximité entre l'enveloppe convexe des solutions entières de notre problème. Un algorithme de branch and cut et d'autres méthodes de résolution sont ensuite comparés. Les expérimentations qui montrent la puissance de nos méthodes s'appuient sur deux grandes familles d'instances. Les premières sont générées aléatoirement et les secondes proviennent de Beasley's OR-library. Ces expérimentations mettent en valeur la qualité de la borne obtenue par notre formulation.

La seconde partie propose une formulation "*set partitioning*" avec un nombre exponentiel de variables. Dans ce chapitre, la formulation comporte des variables associées à un ensemble de demandes affectées à la même facilité selon l'ordre établi sur leurs distances correspondantes. Nous avons alors développé un algorithme de génération de colonnes pour la résolution de la relaxation continue de notre modèle mathématique. Cet algorithme est

ensuite déployé au sein d'un Branch-and-Cut-and-Price afin de résoudre des instances de petites et moyennes tailles avec des temps compétitifs.

La troisième partie présente l'analyse et la comparaison des différentes formulations du problème DOMP Monotone. Ces formulations combinent plusieurs manières de formuler l'ordre des éléments selon les moyennes pondé rées en utilisant plusieurs programmes linéaires du polytope du p-median. Cette approche donne lieu à deux formulations performantes du DOMP sous l'hypothèse de monotonie des vecteurs lambda. Ces formulations sont comparées de manière théorique puis comparées à d'autres formulations valides pour le cas général du vecteur lambda. Une autre formulation est également proposée, elle exploite l'efficacité du caractère rationnel de la monotonie. Cette dernière permet de résoudre efficacement quelques instances où la monotonie a légèrement disparue. Ces formulations ont fait l'objet de plu sieurs expérimentations dècrites dans ce manuscrit de thèse. Elles montrent que les formulations spécifiques permettent de résoudre des instances plus importantes que pour le cas général.

Acknowledgements

This work has been partially supported by the project FQM-5849 (Junta de Andalucía \FEDER) and the Interuniversity Attraction Poles Programme P7/36 COMEX initiated by the Belgian Science Policy Office. Further, I would like to express my heartfelt appreciation to the DEMACOM (DEsafíos de la MAtemática COMbinatoria) and GOM (Graphes et Optimisation Mathématique) teams for their invaluable encouragement through thick and thin.

I am grateful to all the staff at the "*Departamento de Estadística e Investigación Operativa de la Universidad de Sevilla*", staff at the "*Département d'Informatique de l'Université Libre de Bruxelles*" and staff at IMUS (Instituto de Matemáticas de la Universidad de Sevilla). Specially, I would like to direct a big thank you to Pedro, Vanesa, Ana Belén, Justo, M^a José, Begoña, Alba, Miguel, Amaya, David, Mariló Cubiles, Mariló Jiménez, Rafi, Marina, Juan Luis, Alicia, Jerónimo, etc. You all offered valuable input and helped me prepare my teaching lessons, some even going beyond the call of duty.

I would be remiss if I did not mention the people who invest their time and energy in organizing workshops and conferences. It is thanks to these events that young researchers such as myself are given the chance of learning the ropes from first rate researchers. Thank you Luis Gouveia, Cristóbal Miralles, Marco Lübecke, Gerald Gamrath, Justo Puerto and Antonio Manuel Ródriguez Chía among many others. These events also have a highly social aspect to them. Attending conferences and workshops has allowed me to interact with other young researchers in my field, establishing strong links of friendship which I hold very dear.

Over the course of this PhD, I have frequently had to computationally implement theoretical ideas in a solver to either test or compare different methods. I have focused on Branch-and-Price techniques and I would not have been able to accomplish what I did, had I not had the much needed

assistance of the ZIB institute and its team of SCIP developers. Thank you for developing good solvers that the research community as a whole can benefit from. Specifically, Id like to thank Gerald, Gregor and Felipe, as well as other researchers who are not part of the SCIP team but whose expertise and aid has been priceless; Thank you Alessia, Cristina and Sam.

I have had the fortune of having great teachers and mentors at different stages of my life, all of which have inspired me to go down this road. I am very thankful to all of you: (in chronological order) Jerónima, Marita, Patro, Chari, Montse, Francisco Carmona, Emilio, Luis Javier Lozano, M^a Carmen Ruíz, Mathieu Kessler, Chema Espinosa and Alfredo Marín, to mention but a few.

On an even more personal note, I would like to thank friends and family, whose wholehearted support Ive always enjoyed and will undoubtedly continue to enjoy. I especially want to mention my grandparents, my parents and brothers.

A big shout-out also goes to my "adoptive families" in Cartagena, Murcia, Espinardo, Moratalla, Seville, Brussels and Vilvoorde. You stuck with me in good times and bad; Thank you.

Last, but definitely not least, I want to acknowledge both of my supervisors for the time, dedication and care they have both invested in me. This work would not have been possible without your steadfast support.

To all of you: Thank you so very much, muchísimas gracias et un grand merci à tous!

Contents

1 Introduction and State of the art **11**
- 1.1 Location theory . . . 13
  - 1.1.1 The Ordered Median Problem . . . 18
- 1.2 Combinatorial optimization . . . 19
- 1.3 Column generation . . . 23
  - 1.3.1 Linear programs . . . 23
  - 1.3.2 Mixed Integer Linear Programs . . . 25
  - 1.3.3 Column Generation on Location Problems . . . 26

2 New formulations and comparative study for DOMP **29**
- 2.1 Introduction . . . 29
- 2.2 The problem and some formulations . . . 31
  - 2.2.1 Three-index formulation . . . 32
  - 2.2.2 Two-index formulation . . . 33
  - 2.2.3 A new formulation for DOMP . . . 34
  - 2.2.4 An aggregated formulation . . . 40
- 2.3 Theoretical results . . . 41
  - 2.3.1 Comparison of formulations . . . 41
  - 2.3.2 On the polytope defined by the assignment constraints 53
- 2.4 Computational study . . . 56
  - 2.4.1 Description of the test instances . . . 56
  - 2.4.2 Preprocessing . . . 57
  - 2.4.3 Computational results . . . 59

3 A Branch-and-Cut-and-Price procedure for the discrete ordered median problem **65**
- 3.1 Introduction . . . 65
- 3.2 The problem and its formulation . . . 66

3.3 Column generation to solve LRMP . . . 71
3.3.1 Solving the pricing subproblem . . . 74
3.3.2 Dealing with infeasibility . . . 78
3.3.3 Stabilization . . . 79
3.4 Branch-and-cut-and-price . . . 83
3.4.1 Preprocessing . . . 83
3.4.2 Initialization . . . 84
3.4.3 Upper bound for the Master Problem: A GRASP heuristic . . . 86
3.4.4 Branching . . . 88
3.4.5 Valid inequalities . . . 92
3.5 Computational results . . . 93
3.5.1 Cuts . . . 94
3.5.2 Comparing with $DOMP_4(B\&C-3)$ . . . 95

4 The monotone ordered median problem 97
4.1 Introduction . . . 97
4.2 The model by Ogryczak-Tamir (OT) . . . 98
4.2.1 Further extensions: New formulations for DOMP . . . 100
4.3 The Blanco-El Haj-Puerto model (BHP) . . . 104
4.4 Theoretical results . . . 107
4.5 Computational experiments . . . 110

5 Conclusions and Future Research 115

A Supplementary material for the Chapter 2 "New formulations and comparative study for DOMP" 119
A.1 GAP . . . 120
A.2 Random instances: CPU times . . . 121
A.3 Beasley instances: CPU times . . . 125

B Supplementary material for the Chapter 4 "The monotone ordered median problem" 129
B.1 Detailed results for comparison between formulations $DOMP_{OTG}$ and $DOMP_2$ . . . 130
B.2 GAP . . . 132
B.3 Random instances: CPU times (pleriminary study) . . . 133
B.4 Random instances: CPU times and number of nodes . . . 135

Chapter 1

Introduction and State of the art

Location problems have attracted the interest of scientists since the old times of classic Greek culture. However, the development of the location science started thanks to Alfred Weber who described a pure theory on industry locations in 1909.

Location theory, within Operational Research, includes problems of locating facilities at some sites belonging to a set J of possible sites with the goal of minimizing transportation costs to the clients, I. The discrete ordered median problem -also known by its acronym DOMP- had been studied in the 90's on a continuous environment and over networks. In the last 20 years, it has also been studied on a discrete environment applied to location problems. DOMP consists in solving the following optimization problem

$$\min_{J\subseteq I:|J|=p} \sum_{k=1}^{n} \lambda^k c^k_{\leq}(J).$$

where λ is a vector of nonnegative numbers and $c_{\leq}(J)$ is the ordered vector of costs of the solution obtained from the set of servers J. The choice of the vector λ defines different location problems and it is the ordered vector of cost associated to a feasible solution J. This solution J defines the open facilities.

Scanning the previous contributions to this problem in the literature one can see the evolution followed in the analysis of DOMP; obtaining linear formulations, specific branch and bound methods for the DOMP, studying some

variants of the problems as the free self-service or the capacitated DOMP, applications on p-hubs problems... Furthermore, one can see how the interest for this problem increases nowadays according to the number of citations in research papers.

Taking into account previous work on the DOMP, this thesis tries to focus on theoretical aspects of previous and new formulation, among them its polyhedral structure. Furthermore, we study unexploited approaches and new extensions of this problem. The structure of this thesis is presented in the following paragraphs.

Next subsections of this chapter explains briefly the state of the art in three important fields of operational research for the sake of readability and better understanding of this research: location problems, combinatorial problems and the column generation procedure for linear programming. The aim is to introduce the reader to the combinatorial location problems paradigm and to provide the basic notions of some of the more recent techniques to solve this sort of problems.

In Chapter 2 a widely theoretical and computational study is reported. It presents several new formulations for the Discrete Ordered Median Problem (DOMP) based on its similarity with some scheduling problems. Some of the new formulations present a considerably smaller number of constraints to define the problem with respect to some previously known formulations. Furthermore, the lower bounds provided by their linear relaxations improve the ones obtained with previous formulations in the literature even when strengthening is not applied. We also present a polyhedral study of the assignment polytope of our tightest formulation showing its proximity to the convex hull of the integer solutions of the problem. Several resolution approaches, among which we mention a branch and cut algorithm, are compared. Extensive computational results on two families of instances, namely randomly generated and from Beasley's OR-library, show the power of our methods for solving DOMP. One of the achievements of the new formulation consists in its tighter LP-bound.

In Chapter 3 a branch-cut-and-price procedure is developed for the DOMP. Once the bound provided by the relaxed problem is improved, the next natural step consists in defining a column generation formulation in order to reduce the number of generated variables in huge programs. We define ad hoc procedures to solve the different stages of a branch-cut-and-price framework: a polynomial complex pricing subproblem; an interesting branching methodology which is not a trivial task when binary variables are involved in

a column generation approach; and a separation algorithm which allows us to discard fractional solutions and that is adapted to our pricing subproblem. Furthermore, we use general improvements to help the resolution stage. For instance, we use a stabilization method and we build an add hoc warm-start procedure to add variables to the program which is highly advisable in column generation. A GRASP heuristic for DOMP is used on this framework. This GRASP heuristic, described in Chapter 3, is applied for the first time to this problem. It is used to generate an initial solution for our branch-cut-and-price. The reader may observe that this initial solution could have been used also with other formulations presented in this thesis.

Finally, in Chapter 4 an extension of the OMP is studied. We explain the monotone ordered median problem and we apply two existing formulations to solve the MDOMP (monotonic discrete ordered median problem). With this two add hoc formulations we develop an exhaustive computational study. Furthermore, we detail a theoretical comparison between this two formulations and the general ones defined in Chapter 2.

1.1 Location theory

A location problem consists of determining the position of one or more facilities in order to optimize a measure of effectiveness with respect to a set of known demand locations. Location theory, as any other discipline in Operations Research, develops mathematical models to represent in the best possible way the real situation being studied, with adequate solutions to the problem under study. This area of research has already a long history and it is now in full expansion since a lot of methods and procedures can successfully be adapted in order to solve complex problems belonging to other knowledge areas.

Location problems can be classified into three categories: discrete location, location on networks and continuous location. Discrete location imposes that the set of candidate locations for placing the new facility(ies) is finite. Network location problems assume demand points in a graph and facilities have to be located at the nodes or at interior points of edges of the graph. Finally, continuous location considers problems where demand points and the facility locations belong to a continuous space, typically the Euclidean space. Excellent references that cover these fields in Location Theory are Larson and Odoni [1981], Mirchandani and Francis [1990], Daskin [1995,

2013] and other references covering all fields are Drezner [1995], Drezner and Hamacher [2002], Puerto [1996], Laporte et al. [2002].

In order to get a better understanding of the location problems structure, we briefly describe next the common elements to all of them.

The solution space:

The solution space is the framework where the problem is defined. It contains as elements existing facilities and the new facility(ies). The choice of an appropriate solution space is crucial, because it determines important aspects such as the accuracy and efficiency of the model. Some usual solution spaces are:

- *Discrete spaces:* When there exists a finite number of potential locations for the new facilities.

- *Networks:* The solution candidates lie within a graph, usually representing a communication network. Nodes represent important elements, such as cities or crossroads. Arcs represent connections between nodes, like roads, streets, cables, etc. A kind of network that has received considerable attention is the "tree network". This is due mainly to the uniqueness of a path between pairs of points.

- *Euclidean space* $\mathbb{R}^n$*:* It is used when the problem presents regional aspects that cannot be discretized. In addition, it can be used to approximate networks when the number of nodes and arcs is large.
  The cases $n = 2$ and $n = 3$ have a clear physical meaning. Cases where $n \geq 4$ have been used to model and solve estimation problems in statistics.

- *Sphere:* It is useful for those real situations that cope with large scale distances.

- *Embedded network in a continuous space:* This is the solution space where a network, that represents high speed connections, overlaps an Euclidean space or a sphere.

Existing locations:

In terms of Location Theory, *existing facilities* are the users that require to be served. Therefore, they are called demand points. Usually, they are

modelled by means of a set D and an *intensity* function to weight the elements of D.

There exist two different ways of representing demand in the solution space: by a finite set of points and by regions. In the first case, a set of points $D = \{a_1, \ldots, a_n\}$ is considered as well as a set of *weights* $\{w_1, \ldots, w_n\}$ that represent the importance (or intensity) of the demand generated at each point. In the regional model, demand is represented by means of a region $\mathcal{R}$ (not necessarily connected) included in the solution space and it is a probability measure which gives importance to each measurable subset of $\mathcal{R}$.

The new facility(ies):

The location of the new facility is the decision variable of the general location problem. This variable is characterized by

a) Number and quality of the service provided. If more than one facility is to be located, it will be necessary to specify the characteristics of each one of them. When they are identical, as for instance mail boxes, we face with a multifacility problem; otherwise as in the case of health services, we may face hierarchical location problems.

b) Nature of the service. Not all the services are attractive for the community where they will be located. For instance, nuclear plants, solid waste disposals or garbage plants are usually refused by population. Therefore, in modeling a problem it is very important to determine the attractiveness of the service.

The objective function:

Location problems mentioned in this section have the following objective function in common:

$$\text{opt}_{X=(x_1,\ldots,x_p)\subset S} F\Big(d(X,a)_{a\in D}\Big),$$

where

F is a globalizing function,

"opt" means optimize, either minimize or maximize,

S is the solution space,

$X = \{x_1, \ldots, x_p\} \subset S$ is the new facility(ies), either single $p = 1$ or multiple $p > 1$.

D is the set of existing facilities (demand points),

a is a general existing facility,

$d(\cdot, \cdot)$ is a measure of distances. In general, $d(X, a)$ stands for the distance between demand a and the set of facilities $(x_1, \ldots, x_p)$, i.e. $d(X, a) = \min_{b \in X} d(b, a)$.

Determining which objective function has to be used is sometimes a hard task. It should be noted that the final solution strongly depends on that choice. Therefore, it is important to devote some effort to this part of the modelling process.

1. *The p-median problem or "minisum".* The p-median problem [Hakimi, 1964, 1965] searches for the location of p facilities with the objective of minimizing the weighted sum of distances between the demand points and the facilities to which they are located. A general p-median formulation is the following:

$$\min_{X \subset S} \sum_{a \in D} d(X, a).$$

The choice of minimizing the average distance can be justified when an economic criterion is imposed.

2. *The p-center problem problem or minmax.* The p-center problem assumes that all the demand is covered with p facilities and minimizes the coverage distance for doing so, that is, the maximum weighted distance between a demand and its nearest facility is minimized as follows:

$$\min_{X \subset S} \max_{a \in D} d(X, a).$$

The minmax model can be interpreted as a quality criterion of the developed service in terms of equity.

3. *Centdian problem.* Given a positive scalar $\lambda \in (0,1)$, the objective function corresponds now to a convex combination of the criteria minisum and minmax. That is, the problem is:

$$\min_{X \subset S} (\lambda \sum_{a \in D} d(X,a) + (1-\lambda) \max_{a \in D} d(X,a)).$$

The cent-dian model corresponds to a compromise between the center and median criteria, that are conflicting criteria in most of the cases.

4. *Ordered median problem.*

Given a finite number of existing facilities $D = \{a_1, \ldots, a_M\}$ and weights $\lambda_1, \ldots, \lambda_M$, the objective is to find the location of X minimizing an ordered sum of distances, i.e.,

$$\min_{X \subset S} \sum_{i=1}^{p} \lambda_i d_{(i)}(X).$$

Here, $d_{(i)}(X) = d(X, a_{\sigma_i})$ is the i-th element in the list of sorted distances

$$d(X, a_{\sigma_1}) \leq \ldots \leq d(X, a_{\sigma_M}),$$

where σ is a permutation of $\{1, ..., M\}$. Note that this objective function is point-wise defined, because its expression changes when the order between distances is modified. This function is somehow similar to the p-median, but is more general because it includes as particular instances the minsum, minmax and centdian (among others not referenced in this list).

5. *Set covering problem.* In this problem, the number of facilities to be located is not fixed *a priori*, that is, the cardinality of X (denoted by $|X|$) has to be minimized and determined together with its elements. Each existing facility s should be within a specified distance from at least a new facility r_a. The objective is to find the lowest number of facilities and their location verifying the above constraint. Thus, the problem can be written as:

$$\min_{X \subset S : d(X,a) \leq r_a, a \in D} |X|.$$

6. *Maximal covering problem.* The objective of this problem is to have as many existing facilities within a specified distance, called the covering distance, from a nearest facility. Unlike the set covering model, the set of new facilities is fixed to p. Let us consider $\delta(d(X,a)) = 1$ iif $d(X,a) \leq r_s$; 0 otherwise. The problem is:

$$\max_{X \subset S} \sum_{a \in D} \delta(d(X,a)).$$

7. *Multiobjective problem.*

   The previous objectives establish *a priori* the criteria used to locate the new facility(ies). However, there exist real situations where it would be reasonable to use simultaneously several criteria. This implies to find solutions that are optimal to several criteria at the same time. This type of problems are called multiobjective location problems. Given that the different criteria are usually in conflict, the "ideal" solution rarely exists and, therefore, one has to decide which concept of "optimality" to choose. For example, the non-dominated solutions are those that cannot be improved for all objectives by any other solution.

   For a detailed discussion on the nature of multiobjective location problems we refer to Nickel et al. [2005] and Nickel et al. [2015].

1.1.1 The Ordered Median Problem

In this section, we introduce formally the ordered median function (OMf). This function is a weighted average of ordered elements. For any $x \in \mathbb{R}^d$ denote $x_{\leq} = (x_{(1)}, ..., x_{(n)})$ where $x_{(1)} \leq \cdots \leq x_{(n)}$. We consider the function:

$$\begin{aligned} sort_n : \quad & \mathbb{R}^d \longrightarrow \mathbb{R}^d \\ & x \longmapsto x_{\leq}. \end{aligned}$$

Let $\langle ., . \rangle$ denote the usual scalar product in $\mathbb{R}^d$.

Definition 1. *The function $f_\lambda : \mathbb{R}^d \longrightarrow \mathbb{R}$ is an ordered median function, for short $f_\lambda \in OMf(n)$, if $f_\lambda(x) = \langle \lambda, sort_n(x) \rangle$ for some $\lambda = (\lambda_1, ..., \lambda_n) \in \mathbb{R}^n$.*

Depending on the choice of the vector of weight λ we get different class of problems:

Example 1.

- *For* $\lambda = (1, ..., 1)$ *and* $f_\lambda(x) = \sum_{i=1}^{n} x_i$*, we refer to the classical Weber objective funtion.*
- *For* $\lambda = (0, ...0, 1)$ *and* $f_\lambda(x) = \max_{i=1}^{n} x_i$*, we refer to the center function (maximum component).*
- *For* $\lambda = (-1, 0, ...0, 1)$ *and* $f_\lambda(x) = \max_{i=1}^{n} x_i - \min_{i=1}^{n} x_i$*, we refer to the range objective function.*
- *For* $\lambda = (0, ..., 0, 1, \overbrace{...}^{k}, 1)$ *and* $f_\lambda(x) = \max_{i=1}^{k} x_{(i)}$*, we refer to the k-centrum objective function.*

It is clear that ordered median functions are nonlinear functions. Whereas the nonlinearity is induced by the sorting. To understand better the behavior of these functions, we present some of its interesting properties.

Proposition 1. *Nickel and Puerto [2005]. Let* $f_\lambda, f_\mu \in OMf(n)$*.*

1. $f_\lambda(x)$ *is a continuous function.*
2. $f_\lambda(x)$ *is a symmetric function, i.e. for any* $x \in \mathbb{R}^d$ $f_\lambda(x) = f_\lambda(sort_n(x))$*.*
3. $f_\lambda(x)$ *is a convex function iff* $\lambda_1 \leq \cdots \leq \lambda_n$*.*
4. *If* c_1 *and* c_2 *are constants, then the function* $c_1 f_\lambda(x) + c_2 f_\mu(x) = OMf(n)$*.*
5. *If* $f_\lambda(x) \in OMf(n)$ *and* $f_\mu(u) \in OMf(1)$*, then the composite function is an ordered median function of* x *on* $\mathbb{R}^d$*.*

1.2 Combinatorial optimization

Combinatorial Optimization is a field of Mathematics that involves problems for which the solution consists on choosing the best solution from a finite set of candidates. In the last decades it has been applied successfully in many different fields: industry, biology, social science, logistics, location theory and many others.

In many cases a solution of a Combinatorial Optimization Problem (COP) can be described by its characteristic vector. This representation allows to use 0-1 variables to transform the original formulation into a mathematical programming problem of the form: $\min\{f(x) : x \in X\}$ where $f(x)$ is the objective function, x is the characteristic vector of a feasible solution and X is called the feasible region. In the particular case that $f(x)$ is linear and X can be modeled by linear inequalities, the problem results in

$$\begin{array}{lll} \min & cx & \\ s.t. & Ax & \geq b \\ & x & \in \{0,1\}^n. \end{array}$$

This sort of problems has a finite number of solutions. However, finite does not mean easy.

For some problems, there is an exponential number of feasible solutions to enumerate. For instance, in the Traveling Salesman Problem, the salesman has $n-1$ choices for the first trip, $n-2$ for the second, and so on. Thus there are $(n-1)!$ feasible tours. Table 1.1 shows how rapidly the number of feasible solutions can grow in function of the size of the problem n.

n	$\log n$	$\sqrt{n}$	n^2	2^n	$n!$
10	3.32	3.16	10^2	1.02×10^3	3.60×10^6
100	6.64	10.00	10^4	1.27×10^{30}	9.33×10^{157}
1000	9.97	31.62	10^6	1.07×10^{301}	4.02×10^{2567}

Table 1.1: Number of feasible solutions for some typical functions.

For the sake of completeness, in the following we recall the basic definitions of complexity for decision problems. For more details about complexity, see Garey and Johnson [1979] or Chapter 6 in Wolsey [1998].

In order to check whether or not COP is polynomially solvable, it is appropriate to define a decision problem having a YES-NO answer. Thus the opmimization problem $min\{cx : x \in X\}$ for which an instance consists of: $\{c$ and a "standard" representation of $X\}$ is replaced by the decision problem: Is there an $x \in X$ with value $cx \leq k$?

Definition 2. *$\mathcal{NP}$ is the class of decision problems with the property that: for any instance for which the answer is YES, there is a "short" (polynomial) proof of the YES.*

Definition 3. *$\mathcal{P}$ is the class of decision problems in $\mathcal{NP}$ for which there exists a polynomial algorithm.*

Definition 4. *A decision problem belongs to the class $\mathcal{NP}$-complete if it belongs to $\mathcal{NP}$ and every problem in $\mathcal{NP}$ is reducible to it in polynomial time.*

It is currently an open question whether $\mathcal{P} = \mathcal{NP}$, although there are many combinatorial optimization problems which belong to $\mathcal{NP}$-complete. In particular, the DOMP is an $\mathcal{NP}$-complete problem (Nickel and Puerto [2005]). Other classical examples of problems in $\mathcal{NP}$-complete are the 0-1 Knapsack Problem, the Set Covering Problem, The Traveling Salesman Problem or the Maximum Independent Set Problem. In general, it is well-known that the linear integer programming problem belongs to $\mathcal{NP}$.

On the contrary, linear programming is proved to be solved in polynomial time by using either ellipsoid methods or interior point algorithms (Khachiyan [1979], Khachiyan [1980], Karmarkar [1984], Schrijver [1986], Grötschel et al. [1988])[1]. This fact means that if we are able to avoid the use of integer variables in our linear representation of the COP, the problem is proved to be solved in polynomial time for any instance. Thus, searching for the description of the polyhedron of this problem is a crucial issue.

Definition 5. *Given a set $X \subseteq \mathbb{R}^n$, the convex hull of X, denoted by conv(X) is defined as: conv$(X) = \{x : x = \sum_{i=1}^{t} \lambda_i x^i, \sum_{i=1}^{t} \lambda_i = 1, \lambda_i \geq 0$ for $i = 1 \ldots, t$ over all finite subsets $\{x^1, \ldots, x^t\}$ of $X\}$.*

The convex hull is the minimum convex set containing any feasible solution. Therefore, in those problems where all variables are defined as integer all the vertices of the convex hull will be integer.

For this reason, if we were able to define the convex hull of a problem by means of linear inequalities it could be solved by means of linear programming. Hence, if we have a formulation which defines the convex hull with a polynomial number of constraints or with an exponential number but with

[1]Although in practice LP are solved by simplex method (Dantzig et al. [1955]) whose complexity is not polynomial.

an efficient separation algorithm, our problem is polynomial in terms of complexity. This occurs, for instance, when the matrix of coefficients A is totally unimodular (TU) and the resource vector b is integer, e.g. the assignment problem.

For the $\mathcal{NP}$-complete decision problems it is not possible to define a LP formulation (unless $\mathcal{P} = \mathcal{NP}$), but it is possible to improve the formulation working on this idea. In this regard we would like to describe exact faces of the polyhedron of feasible solution. Among them we are interested on those faces of full dimension.

Full-dimensional polyhedra have the property that there is no equation $ax = b$ satisfied at equality by all points $x \in P$.

Theorem 1. *If P is a full-dimensional polyhedron, it has a unique minimal description $P = \{x \in \mathbb{R}^n : a^i x \leq b_i \text{ for } i = 1, \ldots, m\}$, where each inequality is unique to within a possitive multiple.*

Definition 6. *(i) F defines a face of the polyhedron P if $F = \{x \in P : \pi x = \pi_0\}$*

(ii) F is a facet of P if F is a face of P and $dim(F) = dim(P) - 1$

(iii) If F is a face of P with $F = \{x \in P : \pi x = \pi_0\}$, the valid inequality $\pi x \leq \pi_0$ is said to represent or define the face.

Proposition 2. *If P is full-dimensional, a valid inequality $\pi x \leq \pi_0$ is necessary in a description of P if and only if it defines a facet of P.*

We refer to the Chapter 9 of Wolsey [1998] for sufficient conditions ensuring that an inequality defines a facet.

The idea is that if one defines a family of facets we have tighter formulations because facets are strong valid inequalities of our formulation. Nevertheless, facets not always help to solve the problems since they may describe polyhedron regions where we are not interested to optimize.

In those cases where an efficient representation of the convex hull is not known, it is still possible to improve its description by adding valid inequalities (non-necessarily facets). This strategy will improve the bound provided by the linear relaxation of the problem which will result in better formulations. Another strategy that can also improve the bounds is to use Lagrangean relaxation or Benders decomposition.

From the above discussion, unless $\mathcal{P} = \mathcal{NP}$ (Garey and Johnson [1979]), we cannot expect to close the integrality GAP in all instances of a COP in $\mathcal{NP}$. In spite of that, it is currently a crucial topic to develop formulations and tools that tighten the polyhedral description and improve the primal bounds of COP.

1.3 Column generation

Column generation is a well-known technique to solve linear programs which have a big number of variables in comparison with the number of constraints. The idea behind this approach is based on the reduced cost of the variables: when the reduced cost are nonnegative (nonpositive) for all the variables in a minimization (maximization) problem the optimum is reached. By means of the classic simplex algorithm one can certify optimality if there is no nonbasic variable to add to the basis. In Section 1.3.1 it will be detailed how it is possible to certify optimality even when the variables are not defined explicitly in the problem.

The first time that the term *column generation* appeared in a paper was in Appelgren [1969]. However, the idea comes from Ford and Fulkerson [1958] and it was developed before in Dantzig and Wolfe [1960], Gilmore and Gomory [1961, 1963]. See Nemhauser [2012] for more historical details.

Nowadays, the branch-and-price method (column generation with integer variables) has been applied successfully to many combinatorial problems: crew scheduling, cutting stock, binpacking, VRP, etc (see Barnhart et al. [1998]).

In Section 1.3.2 the insights of a column generation approach when there are integer variables in the formulation of the problem will be described.

1.3.1 Linear programs

Let us assume that, either from a Dantzig-Wolfe reformulation or from a direct formulation with a big number of variables, a Master Problem (MP) is defined. We assume that in this problem $n \gg m$, being n the number of variables and m the number of constraints. We explain in the following the

analysis for a minimization problem.

$$\begin{aligned} (MP)\min \quad & cx \\ s.t. \quad & Ax \geq b \\ & x \geq 0 \end{aligned}$$

From the above (MP) the Restricted Master Problem (RMP) is built employing a subset of variables. This is the key of the column generation method, since most of the variables are not needed in the basis to certify the optimality of the solution. From the RMP, the dual multipliers, π, are calculated by solving the dual problem.

$$\begin{aligned} (D)\max \quad & b'\pi \\ s.t. \quad & A'\pi \leq c' \\ & \pi \geq 0 \end{aligned}$$

For each variable x_i the reduced cost is $\bar{c}_i = c_i - A_i'\pi_i$. Obviously, if the variable is in the Restricted Master Problem, $\bar{c}_i \geq 0$. Thereby, the next step consists in checking whether a variable with negative reduced cost (taking into account this solution) exists. In order to certify optimality, one strategy consists in solving $\min\{c_i - \pi_i A_i : i \in I\}$. This problem is known as the pricing subproblem and gives the most promising variable to add to the RMP or to certify optimality in the case that its objective value is nonnegative. Once the solution of the pricing problem is nonnegative, it means that no addition of a nonbasic variable may improve the value of the objective function of the MP.

We have to mention an important issue in column generation. At the beginning, if one does not create some variables, the MP could be infeasible. To avoid this inconvenient, there are some possibilities. For instance, to include an artificial variable, with a big M coefficient in the objective function, and satisfying all the constraints. Another option is to create heuristically a feasible solution. With this latter option we ensure feasibility during the rest of the process. However, when the problem has integer variables, the feasibility could be lost because of the branching conditions (see Section 1.3.2). Next we will outline a generic method to recover feasibility (or to check the infeasibility if it is the case).

The Farkas pricing[2] is based on the Farkas Lemma (Farkas [1894]) which

[2]Although this method was used previously, this terminology was introduced in Achterberg [2009].

establishes that either $Ax = b, x \geq 0$ is feasible or $\max\{\pi b : \pi A \leq 0\}$ is unbounded. Hence, a way to turn the MP on feasible consists in finding a constraint of the dual such that $\pi A_i > 0$. Note that finding this new constraint (column in terms of the primal) is equivalent to the pricing problem with $c = 0$. The advantage of this method lies in its simplicity since once the pricing subproblem is defined, the complexity of the Farkas subpricing is similar.

1.3.2 Mixed Integer Linear Programs

One can think in the branch-and-price as an usual branch-and-bound technique for which a column generation procedure is used to solve the linear relaxation at each node. Therefore, the concepts of Master, Restricted Master and Pricing problems, explained in Section 1.3.1, do not change when integrated within a branch-and-price scheme. In particular when the variables are defined as global [3], there is no difference. However, to use it within a branch-and-bound or branch-and-cut techniques, it is necessary to take into account some particular details.

The standard branch-and-bound methodology, i.e. branching on the x variables generates some difficulties (Vance et al. [1994]). On the one hand, the obtained tree will not be balanced enough , i.e the number of feasible solutions in the nodes of the tree is very different. On the other hand, it is not an easy task to provide a tractable subproblem because it could be difficult to encode the branching information. For this reason, there are other branching techniques such as the Ryan and Foster branching (Ryan and Foster [1981]) or branching on original variables (Johnson [1989]) which are more appropriate for a branch-and-price framework. We refer the reader to Barnhart et al. [1998] for a detailed explanation of the branching involved on a column generation approach.

Some authors define the branch-cut-and-price as a variant of a branch-and-bound, with bounds provided by solving linear programs using column-and-cut generation at nodes of the branch-and-bound tree (see for instance Barnhart et al. [2000]). For this reason, cut techniques in column generation approaches present the difficulty of integrating a new family of multipliers in the pricing problem.

[3] Once a variable is added to the Restricted Master Problem it will be used at the remaining open nodes.

Implementation of Branch-and Price-and-Cut procedures

In column generation, the more difficult implementation stage is the pricing subproblem. However, in a branch-and-price procedure this issue is even more complicated since the column generation has to be embedded into a branch-and-bound tree. From last decades some programs are able to implement this feature. For example, `MINTO`, `CPLEX` (Barnhart et al. [1998]). Nowadays, the current state of the art is given by `SCIP` (Achterberg [2009]). This fact explains the increasing interest of the operational research community in this software. `SCIP` is a framework for branch-cut-and-price (among other utilities) with was developed at ZIB.

From a practical point of view `SCIP` is a very flexible and advisable framework due to its versatility. This program is based on plugins and callbacks, which let the researcher implement the methodology developed theoretically. In the words of the developers *"It allows for total control of the solution process and the access of detailed information down to the guts of the solver"*. In particular, on a branch-cut-and-price approach, `SCIP` is designed to implement all the aspects commented above: pricing subproblem, Farkas pricing subproblem, branching, separator to deal with the cuts, etc.

1.3.3 Column Generation on Location Problems

In this subsection we review some of the papers in the literature which have used column generation ideas to solve location problems. In du Merle and Vial [2002] the authors present a variant of the analytic center cutting plane method (ACCPM) for column generation and they apply this method to the $p-$median problem. Branch-and-price has been applied successfully on classical location problems, as the capacitated $p-$median (Lorena and Senne [2004]) or the p-median problem (Senne et al. [2005]). This last two papers use column generation stabilized using Lagrangean/surrogate relaxation (Senne and Lorena [2001]). In Ceselli and Righini [2005] the authors present a branch-and-price algotithm that exploits column generation, heuristics and branch-and-bound to solve the capacitaded p-median problem. See Reese [2006] for more information about references and topics up to 2006.

Later in Avella et al. [2006] column-and-row generation using cutting planes was applied to the p-median problem. A column branch-and-price procedure was applied also to capacitated hub location problems in Contreras et al. [2011] where the lower bound was improved by means of a Lagrangean

relaxation. Another branch-and-price with Lagrangean relaxation was applied in Klose and Görtz [2007] for the capacitated facility location problem. Later in Ceselli et al. [2008] the authors show a general framework for network location problems, based on column generation and branch-and-price.

In this work, we propose for the first time a column generation formulation for Discrete Ordered Median Problem, DOMP.

Chapter 2

New formulations and comparative study for DOMP

2.1 Introduction

Recognizing the need for more flexible logistic models, Nickel [2001] proposed the Discrete Ordered Median Location Problem (DOMP) which could be used to model different locations problems, as the p-median or the p-center. It is a flexible formulation based on applying an ordered weighted averaging operator to the costs as they appear in the solution and taking them into account with a suitable n-vector λ.

Given a vector of weights, the ordered weighted average of n real numbers is obtained by first ranking those numbers by nondecreasing order and then computing the scalar product of the ranked allocation cost vector and the weight vector, see e.g. Nickel and Puerto [2005].

Consider a set of clients and a set of candidate locations where some facility can be established. Further we are given the costs for allocating clients to facilities. DOMP consists in choosing p facility locations and assigning each client to a facility with smallest allocation cost in order to minimize a special objective function, the so called ordered median function. Given a vector of weights, this function consists in an ordered weighted average of the allocation costs, namely it sorts these costs in non-decreasing sequence and then it performs the scalar product of this so obtained sorted cost vector and the given vector of weights. This objective function has been widely applied in the field of location analysis and distribution models (Marín et al. [2009],

Kalcsics et al. [2010] and Puerto et al.). In addition, it has the potential to yield new models for order statistics embedded within mathematical programming formulations; thus enlarging the applications of optimization tools to the resolution of statistical problems in data analysis.

DOMP is known to be $\mathcal{NP}$-complete, see Nickel and Puerto [2005].

The first formulation of DOMP, proposed by Nickel [2001] consists in an integer nonlinear problem. Then, Boland et al. [2006] propose several linearizations of Nickel's model.

Instances with up to 30 clients could be solved to optimality by Boland et al. [2006]. Further, if clients and facility locations coincide and if the allocation cost of a client to itself is equal to zero (the so-called free self service), then instances with up to 100 clients could be solved by Marín et al. [2009, 2010]. We observe that in all previously considered formulations the gaps with respect to the linear programming relaxations of those models are rather large, as mentioned in all those papers.

In this chapter, we propose new formulations for DOMP and we develop a theoretical comparison of the lower bounds obtained from their LP-relaxations and show that our new formulations are rather tight. Our theoretical results also attempt to shed some light on the polyhedral structure of the new formulation based on scheduling constraints. We conclude with extensive computational experiments to compare the respective efficiency of these formulations.

For the ease of presentation, we assume in some cases in this chapter that the client and facility location sets coincide. However, it is important to remark that all the models and results presented extend to the general case in which client and facility location may differ since we do not impose that the cost of allocating a client to itself is equal to zero.

The remaining of this chapter is organized as follows: in Section 2.2 we define the problem and some previous formulations. Next, we present new formulations for the DOMP. In addition, we analyze the relationship between the polytopes of the previously known formulations of this problem and we identify facets for the related assignment polytope in Section 2.3. Finally, some computational experiments are reported in Section 2.4.

2.2 The problem and some formulations

Let I be a set of n points which at the same time represent clients and potential facility locations.

The cost for serving client i's demand from facility j is denoted by C_{ij} and a facility can serve as many clients as needed, i.e. facilities are uncapacitated.

The Discrete Ordered Median Problem (DOMP) consists in

(i) determining a subset J of p facility locations, $J \subset I$, to open and

(ii) assigning clients to closest open facilities in order to minimize the ordered median objective function defined as follows.

Given the set J of p open facilities, let $c_i(J)$ represents the cost for allocating client i to some facility in J such that $c_i(J) = \min_{j \in J} C_{ij}$.

Now let us rank the costs $c_i(J)$, $i \in I$ by non-decreasing order of their values. These ordered costs are denoted by $c^k_{\leq}(J)$ and verify

$$c^1_{\leq}(J) \leq \cdots \leq c^n_{\leq}(J).$$

Then, given a vector $\lambda = (\lambda^k)_{k=1}^n$ satisfying $\lambda^k \geq 0, k = 1, \ldots, n$, the DOMP objective function, also called ordered median function, is defined as

$$\sum_{k=1}^{n} \lambda^k c^k_{\leq}(J). \tag{2.1}$$

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

We define the p-facility Discrete Ordered Median Problem as determining the subset J, of p facilities to open in order to minimize the ordered median function:

$$\min_{J \subset I : |J| = p} \sum_{k=1}^{n} \lambda^k c^k_{\leq}(J). \tag{DOMP}$$

2.2.1 Three-index formulation

The formulation that we present below, denoted by ($DOMP_1$), was introduced by Boland et al. [2006]. It uses three-index variables x_{ij}^k such that $x_{ij}^k = 1$, if client i is served by facility j and cost $c_i(J) = C_{ij}$ is the k-th smallest in the ordered sequence $c_{\leq}(J)$, $x_{ij}^k = 0$ otherwise. Further, it also uses location variables y_j such that $y_j = 1$ if $j \in J$ and $y_j = 0$ otherwise.

If $x_{ij}^k = 1$, we say that allocation of client i to facility j is in position k, or that couple ij is in position k.

$$(DOMP_1) \quad \min \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} \lambda^k C_{ij} x_{ij}^k \tag{2.2}$$

$$s.t. \quad \sum_{j=1}^{n}\sum_{k=1}^{n} x_{ij}^k = 1 \qquad i = 1, \ldots, n \tag{2.3}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij}^k = 1 \qquad k = 1, \ldots, n \tag{2.4}$$

$$\sum_{k=1}^{n} x_{ij}^k \leq y_j \qquad i, j = 1, \ldots, n \tag{2.5}$$

$$\sum_{j=1}^{n} y_j = p \tag{2.6}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij} x_{ij}^{k-1} \leq \sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij} x_{ij}^k \quad k = 2, \ldots, n \tag{2.7}$$

$$x_{ij}^k \in \{0, 1\} \qquad i, j, k = 1, \ldots, n \tag{2.8}$$

$$y_j \in \{0, 1\} \qquad j = 1, \ldots, n. \tag{2.9}$$

By means of (2.3) we ensure that each location is served by exactly one facility. In the same way, in each position there must be exactly one allocation (2.4). We know that a client can be allocated to a facility only if this facility is open, i.e. $x_{ij}^k \leq y_j$ for all i, j, k. Furthermore, each allocation of client to facility can be placed in at most one position. Hence, $x_{ij}^k \leq y_j$ can be strengthened yielding constraint (2.5). The equality constraint (2.6) implies that there are exactly p open facilities. Inequality (2.7) imposes that the allocation cost in position $k-1$ cannot be greater than the one in position k. Finally, the variables are binary, see (2.8) and (2.9).

2.2.2 Two-index formulation

This formulation($DOMP_2$) was described for the first time in Puerto [2008] and Marín et al. [2009] and later applied to a hub problem in Puerto et al.. It considers a vector that contains all the different values in the cost matrix C, augmented with zero if it is not present in matrix C, as it is explained below.

Let C be a matrix and assume that it contains G different values such that $c_{ij} > 0$. Then, the $(G+1)$-dimensional vector $c_{(.)}$ is constructed as follows

$$c_{(0)} = 0 < c_{(1)} < c_{(2)} < \cdots < c_{(G-1)} < c_{(G)} = \max\{C_{ij} : i, j = 1, \ldots, n\}.$$

To formulate the problem we need to define the following binary variables. Variable $x_{ij} = 1$ if client i is served by facility j and 0 otherwise, variable $y_j = 1$ if $j \in J$ and 0 otherwise and variable $u_{kh} = 1$ if the k-th smallest allocation cost is greater than $c_{(h-1)}$ and 0 otherwise. Further, we set $u_{k0} = 1$ and $u_{k,G+1} = 0, k = 1, \ldots, n$.

The problem to solve is

$$(DOMP_2) \quad \min \sum_{k=1}^{n}\sum_{h=1}^{G} \lambda^k (c_{(h)} - c_{(h-1)}) u_{kh} \tag{2.10}$$

$$s.t. \quad \sum_{j=1}^{n} y_j = p \tag{2.11}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n \tag{2.12}$$

$$x_{ij} \leq y_j \qquad i, j = 1, \ldots, n \tag{2.13}$$

$$u_{kh} \geq u_{k,h+1} \; k = 1, \ldots, n, \; h = 1, \ldots, G-1 \tag{2.14}$$

$$u_{k+1,h} \geq u_{kh} \; k = 1, \ldots, n-1, \; h = 1, \ldots, G \tag{2.15}$$

$$\sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} > c_{(h-1)}}}^{n} x_{ij} = \sum_{k=1}^{n} u_{kh} \quad h = 1, \ldots, G \tag{2.16}$$

$$x_{ij} \in \{0, 1\} \qquad i, j = 1, \ldots, n \tag{2.17}$$

$$u_{kh} \in \{0, 1\} \qquad k = 1, \ldots, n, \; h = 1, \ldots, G \tag{2.18}$$

$$y_j \in \{0, 1\} \qquad j = 1, \ldots, n. \tag{2.19}$$

The objective function (2.10) is equivalent to (2.1). Assume that the k-th smallest allocation cost is equal to $c_{(h_k)}$ for some h_k; then by the definition of the variable u_{kh_k}

$$\sum_{h=1}^{G}(c_{(h)} - c_{(h-1)})u_{kh} = \sum_{h=1}^{h_k}(c_{(h)} - c_{(h-1)}) = c_{(h_k)} - c_{(0)} = c_{(h_k)}$$

provided that $u_{kh_k} = 1$ and $u_{k,h_k+1} = 0$. Finally,

$$\sum_{k=1}^{n}\sum_{h=1}^{G}\lambda^k(c_{(h)}-c_{(h-1)})u_{kh} = \sum_{k=1}^{n}\lambda^k\sum_{h=1}^{G}(c_{(h)}-c_{(h-1)})u_{kh} = \sum_{k=1}^{n}\lambda^k c_{(h_k)} = \sum_{i=1}^{n}\lambda_i c_i(J).$$

The equality constraint (2.11) ensures that there are exactly p facilities to be located. Constraints (2.12) and (2.13) state that each client is served by one open facility. Equality (2.16) ensures a good definition of the variable u_{kh} and it relates the sorting (u_{kh}) and design variables (x_{ij}). We need to impose some sorting constraints on the u_{kh} variables (2.15). Constraints (2.14) are redundant but they are included because, according to Marín et al. [2009], it significantly strengthen the formulation. Furthermore, all variables are binary, (2.17), (2.18) and (2.19).

Note that others two index formulations has been proposed in Marín et al. [2009, 2010]. However, they are only valid if $c_{ii} = 0\,\forall i = 1, \ldots, n$ (*free self-service*).

2.2.3 A new formulation for DOMP

There are several formulations for DOMP but they all have large integrality gap, as observed previously in the literature see Boland et al. [2006], Marín et al. [2009, 2010] and Puerto [2008]. The main motivation for addressing a new formulation for the DOMP relies on the attempt to reduce this gap.

First, we introduce the following notation.

$$ij \prec \hat{\imath}\hat{\jmath} \equiv \begin{cases} C_{ij} < C_{\hat{\imath}\hat{\jmath}} \\ \quad \text{or} \\ C_{ij} = C_{\hat{\imath}\hat{\jmath}} \text{ and } i < \hat{\imath} \\ \quad \text{or} \\ C_{ij} = C_{\hat{\imath}\hat{\jmath}}, i = \hat{\imath} \text{ and } j < \hat{\jmath} \end{cases} \qquad ij \succ \hat{\imath}\hat{\jmath} \equiv \begin{cases} C_{ij} > C_{\hat{\imath}\hat{\jmath}} \\ \quad \text{or} \\ C_{ij} = C_{\hat{\imath}\hat{\jmath}} \text{ and } i > \hat{\imath} \\ \quad \text{or} \\ C_{ij} = C_{\hat{\imath}\hat{\jmath}}, i = \hat{\imath} \text{ and } j > \hat{\jmath} \end{cases} \quad (2.20)$$

$$ij \preceq \hat{\imath}\hat{\jmath} \equiv \begin{cases} ij \prec \hat{\imath}\hat{\jmath} \\ \quad \text{or} \\ ij = \hat{\imath}\hat{\jmath} \end{cases} \qquad ij \succeq \hat{\imath}\hat{\jmath} \equiv \begin{cases} ij \succ \hat{\imath}\hat{\jmath} \\ \quad \text{or} \\ ij = \hat{\imath}\hat{\jmath}. \end{cases}$$

The above relationship induces a strict total order among the couples ij. Its use avoids to consider multiple, symmetric feasible solutions coming from the structure of the matrix C.

New three index formulation.

Our new formulation uses the same variables and constraints as in the three-index formulation, $DOMP_1$, except that (2.7) is replaced by (2.21) that are called *order constraints*.

The resulting formulation is denoted $DOMP_3$.

$$(DOMP_3) \quad \min \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} \lambda^k C_{ij} x_{ij}^k$$

$$s.t. \quad (2.3), (2.4), (2.5), (2.6), (2.8), (2.9)$$

$$\sum_{\hat{\imath}=1}^{n}\sum_{\substack{\hat{\jmath}=1:\\ \hat{\imath}\hat{\jmath}\succeq ij}}^{n} x_{\hat{\imath}\hat{\jmath}}^{k-1} + \sum_{\hat{\imath}=1}^{n}\sum_{\substack{\hat{\jmath}=1:\\ \hat{\imath}\hat{\jmath}\preceq ij}}^{n} x_{\hat{\imath}\hat{\jmath}}^{k} \leq 1 \quad i,j = 1,\ldots,n, k = 2,\ldots,n. \quad (2.21)$$

The rationale behind constraints (2.21) is illustrated by the following example.

Example 1. Consider the following matrix.

$$\begin{pmatrix} 0 & 2 & 7 & 4 \\ 1 & 0 & 5 & 5 \\ 3 & 6 & 0 & 2 \\ 9 & 4 & 1 & 0 \end{pmatrix}$$

The order of couples ij by means of the above preference order is

$$11 \prec 22 \prec 33 \prec 44 \prec 21 \prec 43 \prec 12 \prec 34 \prec 31 \prec 14 \prec 42 \prec 23 \prec 24 \prec 32 \prec 13 \prec 41.$$

The columns of Figure 2.1 represent the n^2 possible assignments of clients to facilities whereas its rows represent the n positions in DOMP objective function. Each point (bullet or circle) thus represents a variable x_{ij}^k and the bullets correspond to variables which cannot take value 1 simultaneously because, in any feasible solution, the cost of couples assigned to consecutive positions should be non-decreasing.

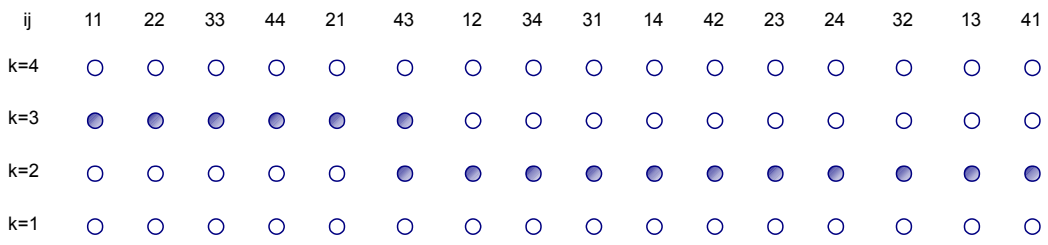


Figure 2.1: An order constraint for the case $\mathbf{n = 4}$

So, Figure 2.1 corresponds to the following inequality:

$$\sum_{i=1}^{n} \sum_{\substack{j=1: \\ \hat{i}\hat{j} \succeq 43}}^{n} x^2_{\hat{i}\hat{j}} + \sum_{i=1}^{n} \sum_{\substack{j=1: \\ \hat{i}\hat{j} \preceq 43}}^{n} x^3_{\hat{i}\hat{j}} \leq 1,$$

or equivalently

$$x^3_{11} + x^3_{22} + x^3_{33} + x^3_{44} + x^3_{21} + x^3_{43} + x^2_{43} + x^2_{12} + x^2_{34} + x^2_{31} + x^2_{14} + x^2_{42} + x^2_{23} + x^2_{24} + x^2_{32} + x^2_{13} + x^2_{41} \leq 1.$$

Note that the number of order constraints is $O(n^3)$. Further they can be seen as cliques of a conflict graph induced by the incompatibility among x^k_{ij} variables on three index formulations (Nemhauser and Trotter [1975], Fernández et al. [2013]).

In a similar way, we can choose several allocations which are sorted and identify a new type of constraints. Let s be a positive integer with $s \leq n-1$ and let $(i_1 j_1), (i_2 j_2), \ldots, (i_s j_s)$ be s couples of clients and facilities such that $i_r j_r \preceq i_{r+1} j_{r+1}$ for all $r = 1, \ldots, s-1$. Then the following family of inequalities, called *staircase inequalities* is valid for $k = s, \ldots, n$:

$$\sum_{i=1}^{n} \sum_{\substack{j=1: \\ ij \preceq i_1 j_1}}^{n} x^k_{ij} + \sum_{r=1}^{s-1} \sum_{i=1}^{n} \sum_{\substack{j=1: \\ ij \succeq i_r j_r \\ ij \preceq i_{r+1} j_{r+1}}}^{n} x^{k-r}_{ij} + \sum_{i=1}^{n} \sum_{\substack{j=1: \\ ij \succeq i_s j_s}}^{n} x^{k-s}_{ij} \leq 1. \tag{2.22}$$

Figure 2.2 provides an example of staircase inequality when $n = 5$.

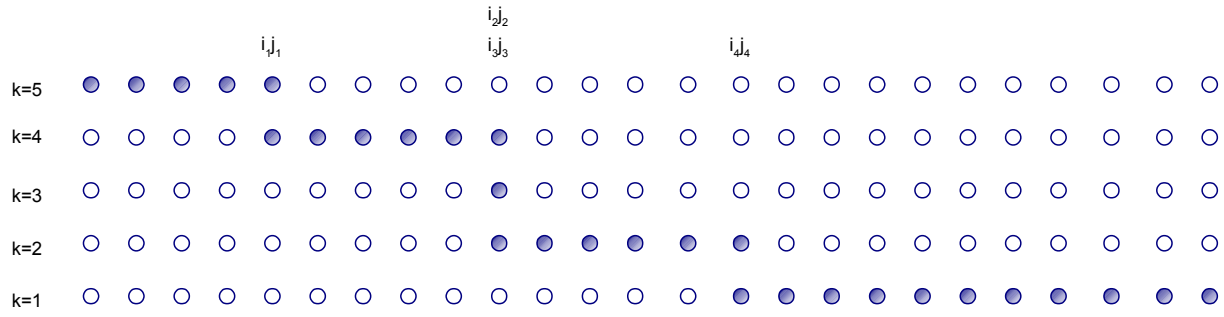


Figure 2.2: A staircase constraint for the case $\mathbf{n = 5}$

Notice that there exists an exponential number of additional staircase constraints. But the question is whether these new constraints actually strengthen our formulation. The answer is provided by the following result that states that all of them are implied by those with a single step, i.e. the order constraints.

Proposition 3. *Staircase inequalities (2.22) with $i_r j_r \preceq i_{r+1} j_{r+1}$ can be obtained as an affine combination of (2.21) and (2.4).*

Proof. Let s be a positive integer such that $s \leq n-1$, then by (2.21) we obtain that also the following s inequalities hold:

$$\sum_{i=1}^{n} \sum_{\substack{j=1: \\ ij \preceq i_r j_r}}^{n} x_{ij}^{k-(r-1)} + \sum_{i=1}^{n} \sum_{\substack{j=1: \\ ij \succeq i_r j_r}}^{n} x_{ij}^{k-r} \leq 1 \quad r = 1, \ldots, s. \tag{2.23}$$

Now, since $i_r j_r \preceq i_{r+1} j_{r+1}$ using (2.4), we obtain the following $s-1$ equations:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}^{k-r} = 1 \quad r = 1, \ldots, s-1. \tag{2.24}$$

Adding all inequalities (2.23) we obtain a new inequality:

$$\sum_{i=1}^{n} \sum_{\substack{j=1: \\ ij \preceq i_1 j_1}}^{n} x_{ij}^{k} + \sum_{i=1}^{n} \sum_{\substack{j=1: \\ ij \succeq i_1 j_1}}^{n} x_{ij}^{k-1} + \cdots + \sum_{i=1}^{n} \sum_{\substack{j=1: \\ ij \preceq i_s j_s}}^{n} x_{ij}^{k-(s-1)} + \sum_{i=1}^{n} \sum_{\substack{j=1: \\ ij \succeq i_s j_s}}^{n} x_{ij}^{k-s} \leq s.$$

After rearranging, we get

$$\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\preceq i_1j_1}}^{n} x_{ij}^{k}+\sum_{r=1}^{s-1}\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\succeq i_rj_r}}^{n} x_{ij}^{k-r}+\sum_{r=1}^{s-1}\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\preceq i_{r+1}j_{r+1}}}^{n} x_{ij}^{k-r}+\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\succeq i_sj_s}}^{n} x_{ij}^{k-s}\leq s.$$

Next, we conveniently split some terms of the above inequality to get:

$$\begin{aligned}&\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\preceq i_1j_1}}^{n} x_{ij}^{k}+\sum_{r=1}^{s-1}\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\succeq i_rj_r\\ ij\preceq i_{r+1}j_{r+1}}}^{n} x_{ij}^{k-r}+\sum_{r=1}^{s-1}\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\succ i_{r+1}j_{r+1}}}^{n} x_{ij}^{k-r}\\ &+\sum_{r=1}^{s-1}\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\preceq i_rj_r}}^{n} x_{ij}^{k-r}+\sum_{r=1}^{s-1}\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\succ i_rj_r\\ ij\preceq i_{r+1}j_{r+1}}}^{n} x_{ij}^{k-r}+\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\succeq i_sj_s}}^{n} x_{ij}^{k-s}\leq 1+(s-1).\end{aligned}\tag{2.25}$$

On the other hand, using equality (2.24) we can write

$$\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\preceq i_rj_r}}^{n} x_{ij}^{k-r}+\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\succ i_rj_r\\ ij\preceq i_{r+1}j_{r+1}}}^{n} x_{ij}^{k-r}+\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\succ i_{r+1}j_{r+1}}}^{n} x_{ij}^{k-r}=1\quad r=1,\dots,s-1.$$

Adding the above equations for all $r=1,\dots,s-1$, we obtain

$$\sum_{r=1}^{s-1}\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\preceq i_rj_r}}^{n} x_{ij}^{k-r}+\sum_{r=1}^{s-1}\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\succ i_rj_r\\ ij\preceq i_{r+1}j_{r+1}}}^{n} x_{ij}^{k-r}+\sum_{r=1}^{s-1}\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\succ i_{r+1}j_{r+1}}}^{n} x_{ij}^{k-r}=s-1.$$

Finally, using the above equation in (2.25) results in

$$\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\preceq i_1j_1}}^{n} x_{ij}^{k}+\sum_{r=1}^{s-1}\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\succeq i_rj_r\\ ij\preceq i_{r+1}j_{r+1}}}^{n} x_{ij}^{k-r}+\sum_{i=1}^{n}\sum_{\substack{j=1:\\ ij\succeq i_sj_s}}^{n} x_{ij}^{k-s}\leq 1,$$

which is a staircase inequality. □

A two index formulation with scheduling constraints.

Formulation $DOMP_3$ is rather efficient and tight whenever there are few ties in the structure of the allocation costs. This is for instance the case of problems with assignment costs based on flat costs (for instance randomly generated). However, if the number of ties in the allocation costs is large the number of binary variables and constraints is relatively large, as compared with similar numbers in formulation $DOMP_2$. In order to exploit this advantage without losing the usage of scheduling constraints, that relate the formulation with the stable set problem, we will develop in the following another formulation.

The new formulation, called $DOMP_{3C}$, is an extension of $DOMP_3$ using the rationale of $DOMP_2$. Moreover, it provides a compact form for representing ties in the allocation costs.

Consider a new set of binary variables, v_{kh} such that $v_{kh} = 1$ if the k-th smallest allocation cost is $c_{(h)}$ and 0 otherwise. Next, $DOMP_{3C}$ is a new valid formulation for DOMP:

$$(DOMP_{3C}) \quad \min \quad \sum_{k=1}^{n}\sum_{h=0}^{G} \lambda^k c_{(h)} v_{kh} \tag{2.26}$$

$$s.t. \quad (2.11), (2.12), (2.13)$$

$$\sum_{h=0}^{G} v_{kh} = 1 \qquad k = 1, \ldots, n \tag{2.27}$$

$$\sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij}=c_{(h)}}}^{n} x_{ij} = \sum_{k=1}^{n} v_{kh} \qquad h = 0, \ldots, G \tag{2.28}$$

$$\sum_{h'<h} v_{k+1,h'} + \sum_{h' \geq h} v_{kh'} \leq 1 \qquad \begin{array}{l} k = 1, \ldots, n-1 \\ h = 1, \ldots, G \end{array} \tag{2.29}$$

$$x_{ij}, y_j, v_{kh} \in \{0,1\} \qquad \begin{array}{l} i, j, k = 1, \ldots, n \\ h = 0, \ldots, G. \end{array} \tag{2.30}$$

Clearly, the objective function (2.26) accounts for the ordered weighted sum of the allocation costs. Constraints (2.28) state that the number of allocations that are attained at the value $c_{(h)}$ regardless of the level k that they occupy (v_{kh} variables) must be equal to the number of allocations of clients i to facility j with ij such that C_{ij} is equal to $c_{(h)}$ (see Figure 2.3).

Finally, constraints (2.29) are scheduling constraints based on costs values rather than in couples ij of client-facility (see Figure 2.4).

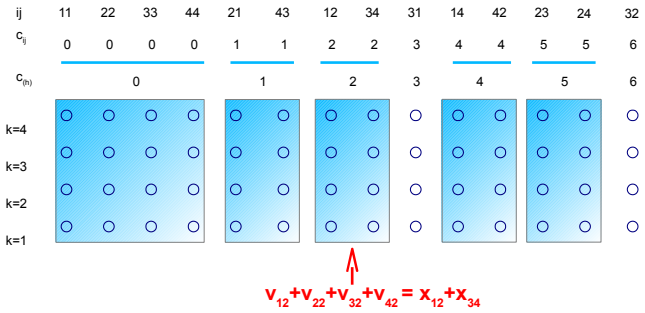


Figure 2.3: The rationale of Constraints (2.28)

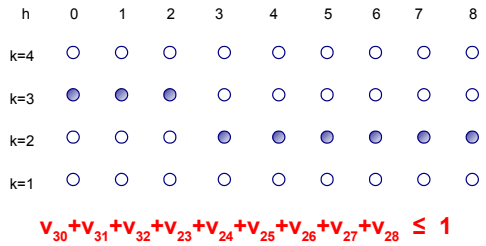


Figure 2.4: The rationale of Constraints (2.29)

2.2.4 An aggregated formulation

Here, we introduce another formulation $(DOMP_4)$ based on the aggregation of order constraints from $DOMP_3$ corresponding to the same position. It therefore requires a smaller number of constraints.

$$(DOMP_4) \quad \min \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} \lambda^k C_{ij} x_{ij}^k$$

$$s.t. \quad (2.3), (2.4), (2.5), (2.6), (2.8), (2.9)$$

$$\sum_{i}^{n}\sum_{j}^{n}\left(\sum_{i'=1}^{n}\sum_{\substack{j'=1: \\ i'j' \preceq ij}}^{n} x_{i'j'}^k + \sum_{i'=1}^{n}\sum_{\substack{j'=1: \\ i'j' \succeq ij}}^{n} x_{i'j'}^{k-1}\right) \leq n^2 \quad k = 2, \ldots, n. \quad (2.31)$$

The new constraints (3.6), that we call *weak order constraints*, ensure that if a couple ij occupies the k-th position then in $(k-1)$-th position there must be a more preferred allocation. It is due to the coefficients of each variable in the inequality. In each constraint there are two different positions, k and $k-1$, so that, by (3.3), two variables must take value one and all the others will be equal to zero. If we do not take into account the variables taking the value zero and we assume that the variables with value one for positions k and $k-1$ are in position s and t, respectively, we have the following:

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

which is valid if and only if $t < s$.

Aggregating the scheduling constraints in $DOMP_{3C}$ for the different values of the costs, namely in $h = 1, \ldots, G$, results in a new valid model. This model is the aggregated version of $DOMP_{3C}$ that we denote as $DOMP_{4C}$:

$$(DOMP_{4C}) \quad \min \quad \sum_{k=1}^{n} \sum_{h=0}^{G} \lambda^k c_{(h)} v_{kh}$$

$$s.t. \quad (2.11), (2.12), (2.13), (2.27), (2.28), (2.30)$$

$$\sum_{h=1}^{G} \left(\sum_{h'<h} v_{kh'} + \sum_{h' \geq h} v_{k-1h'} \right) \leq G \quad k = 2, \ldots, n \quad (2.32)$$

Using the rationale of (2.7), since there is only one binary variable v_{kh} in each position k by (2.27), the following constraints are valid inequalities for both formulations $DOMP_{3C}$ and $DOMP_{4C}$:

$$\sum_{h=1}^{G} c_{(h)} v_{k-1h} - \sum_{h=1}^{G} c_{(h)} v_{kh} \leq 0 \quad k = 2, \ldots, n. \quad (2.33)$$

In fact, these constraints can also define another valid formulation for DOMP replacing (2.32) by (2.33) in the above formulation. In our experiments, we will not use this last possibility and instead, we shall use (2.33) as valid inequalities to strengthen $DOMP_{3C}$ and $DOMP_{4C}$.

2.3 Theoretical results

In this section, we provide a theoretical comparison of the four formulations presented in Section 2.2 and some polyhedral results regarding our formulation $DOMP_3$. Our goal is to state the formal relationships between the lower bounds provided by the linear relaxations of the considered formulations. In addition, we also give some families of tight valid inequalities which are proven to be facets of the polytope defined by assignment constraints (see Section 2.3.2.)

2.3.1 Comparison of formulations

We denote by $z_l(\cdot)$ the value of the objective function of $DOMP_l$ evaluated at the point $(\cdot)$, by P_l the polytope defining the feasible set of the linear

relaxation of formulation $DOMP_l$, and by P_l^I the convex hull of the integer solutions within that polytope.

Consider the following mapping

$$\begin{array}{rcl} f : [0,1]^{n^3} \times [0,1]^n & \longrightarrow & [0,1]^{n^2} \times [0,1]^n \times [0,1]^{nG} \\ (x_{ij}^k, y_j) & \longmapsto & (x_{ij}, y_j, u_{kh}) \end{array}$$

defined by the following two equations

$$x_{ij} = \sum_{k=1}^{n} x_{ij}^k \quad i,j = 1,\ldots,n \tag{2.34}$$

and

$$u_{kh} = \sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} \geq c_{(h)}}}^{n} x_{ij}^k \quad k = 1,\ldots,n, h = 1,\ldots,G. \tag{2.35}$$

For $h = 1,\ldots,G$, let us define $k(h) = \min\{k : u_{kh} - u_{k,h+1} > 0\}$ being $u_{k,G+1} = u_{kG}$. We assume that if $u_{kh} - u_{k,h+1} = 0$ for all k, $k(h) = +\infty$. Next, for each $h = 1,\ldots,G$ let $x_{(ij)_h}$ be the non-null variable $x_{ij} > 0$ such that the couple (ij) is the most preferred, in the pairwise strict order introduced in (2.20), among those satisfying $C_{ij} = c_{(h)}$.

Observe that this couple can be formally defined as the minimal in the order induced by the relation $\prec$ among those with $C_{ij} = c_{(h)}$, namely:

$$(ij)_h = \min_{(\prec)} \{ij : x_{ij} > 0 \text{ and } C_{ij} = c_{(h)}\} \tag{2.36}$$

Based on the above couples, for any feasible solution $(x_{ij}, y_j, u_{kh}) \in P_2$ we construct, sequentially, a feasible solution of P_1. Indeed, for each $h = 1,\ldots,G$ and $C_{ij} = c_{(h)}$, we construct sequentially in the strict order given by (2.20) and from $k = 1$ until $k = n$:

$$x_{ij}^k = \begin{cases} 0 & \text{if } k < k(h) \text{ or } x_{ij} = 0, \\ \min\{x_{ij} - \sum_{\ell<k} x_{ij}^\ell, u_{kh} - u_{k,h+1}\} & \text{if } k \geq k(h) \text{ and } ij = (ij)_h, \\ \min\{x_{ij} - \sum_{\ell<k} x_{ij}^\ell, u_{kh} - u_{k,h+1} - \sum_{(i'j') \prec ij} x_{i'j'}^k\} & \text{if } k \geq k(h) \text{ and } ij \succ (ij)_h. \end{cases} \tag{2.37}$$

Now, using the above definition we introduce the mapping g:

$$\begin{array}{rcl} g : [0,1]^{n^2} \times [0,1]^n \times [0,1]^{nG} & \longrightarrow & [0,1]^{n^3} \times [0,1]^n \\ (x_{ij}, y_j, u_{kh}) & \longmapsto & (x_{ij}^k, y_j) \end{array}$$

where x_{ij}^k is given by (2.37).

These two mappings f and g relate the space of feasible solutions to $DOMP_1$, $DOMP_3$ and $DOMP_4$ with the space of feasible solutions to $DOMP_2$ and conversely.

Observation 1.

- *For any points* $(x_{ij}^k, y_j) \in P_l$ *and* $f(x_{ij}^k, y_j)$, $z_l(x_{ij}^k, y_j) = z_2(f(x_{ij}^k, y_j))$ *for* $l = 1, 3, 4$.
- *For any points* $(x_{ij}, y_j, u_{kh}) \in P_2$ *and* $g(x_{ij}, y_j, u_{kh})$, $z_2(x_{ij}, y_j, u_{kh}) = z_l(g(x_{ij}, y_j, u_{kh}))$ *for* $l = 1, 3, 4$.

We begin by analyzing the strength of the lower bounds provided by the continuous relaxation of formulations $DOMP_l$ for $l = 1, 2, 3, 4$.

Theorem 2. *Let* $p = (x_{ij}, y_j, u_{kh})$. *If* $p \in P_2$ *then* $g(p) \in P_1$.

Proof. Let $(x_{ij}, y_j, u_{kh}) \in P_2$. By construction of (2.37), we have that for any k, $\sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij}=c_{(h)}}}^{n} x_{ij}^k = u_{kh} - u_{k,h+1}$. Moreover, by (2.16), we get $\sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij}=c_{(h)}}}^{n} x_{ij} = \sum_{k=1}^{n} (u_{kh} - u_{k,h+1})$, and adding over k proves (2.35). Thus, it follows that $\sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij}=c_{(h)}}}^{n} x_{ij} = \sum_{k=1}^{n} \sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij}=c_{(h)}}}^{n} x_{ij}^k$ and then by the construction in (2.37) we obtain that $x_{ij} = \sum_{k=1}^{n} x_{ij}^k$. This argument proves that the point (x_{ij}^k, y_j), provided by (2.37), also satisfies (2.34).

Next, since $(x_{ij}, y_j, u_{kh}) \in P_2$, it satisfies (2.12), (2.13) and (2.11) and using that $x_{ij} = \sum_{k=1}^{n} x_{ij}^k$ it follows that (x_{ij}^k, y_j) fulfills (2.3), (2.5) and (2.6), respectively.

Further, note that

$$\sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij}=c_{(h)}}}^{n} x_{ij}^k = \sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij}\geq c_{(h)}}}^{n} x_{ij}^k - \sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij}>c_{(h)}}}^{n} x_{ij}^k = u_{kh} - u_{k,h+1}.$$

Hence,

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij}^{k} = \sum_{h=0}^{G}\left(\sum_{i=1}^{n}\sum_{\substack{j=1:\\ C_{ij}=c_{(h)}}}^{n} x_{ij}^{k}\right) = \sum_{h=0}^{G}(u_{kh}-u_{k,h+1}) = u_{k0}-u_{k,G+1} = 1-0 = 1,$$

i.e. the point (x_{ij}^{k}, y_j) satisfies (2.4).

Now we show that (2.15) and (2.35) imply (2.7). Replacing the variables in (2.15) using (2.35) we obtain: $\sum_{i=1}^{n}\sum_{\substack{j=1:\\ C_{ij}\geq c_{(h)}}}^{n} x_{ij}^{k} \geq \sum_{i=1}^{n}\sum_{\substack{j=1:\\ C_{ij}\geq c_{(h)}}}^{n} x_{ij}^{k-1}$, which implies that

$$(c_{(h)} - c_{(h-1)})\left(\sum_{\substack{i=1\\ :C_{ij}\geq}}^{n}\sum_{\substack{j=1\\ c_{(h)}}}^{n} x_{ij}^{k} - \sum_{\substack{i=1\\ :C_{ij}\geq}}^{n}\sum_{\substack{j=1\\ c_{(h)}}}^{n} x_{ij}^{k-1}\right) \geq 0 \quad \forall h = 1,\ldots,G.$$

Next, adding the above inequalities for all h yields

$$\sum_{h=1}^{G-1}\left[c_{(h)}\left(\sum_{i=1}^{n}\sum_{\substack{j=1:\\ C_{ij}\geq c_{(h)}}}^{n} x_{ij}^{k} - \sum_{i=1}^{n}\sum_{\substack{j=1:\\ C_{ij}\geq c_{(h)}}}^{n} x_{ij}^{k-1}\right) - c_{(h)}\left(\sum_{i=1}^{n}\sum_{\substack{j=1:\\ C_{ij}\geq c_{(h+1)}}}^{n} x_{ij}^{k} - \sum_{i=1}^{n}\sum_{\substack{j=1:\\ C_{ij}\geq c_{(h+1)}}}^{n} x_{ij}^{k-1}\right)\right] +$$
$$c_{(G)}\left(\sum_{i=1}^{n}\sum_{\substack{j=1:\\ C_{ij}\geq c_{(G)}}}^{n} x_{ij}^{k} - \sum_{i=1}^{n}\sum_{\substack{j=1:\\ C_{ij}\geq c_{(G)}}}^{n} x_{ij}^{k-1}\right) - c_{(0)}\left(\sum_{i=1}^{n}\sum_{\substack{j=1:\\ C_{ij}\geq c_{(1)}}}^{n} x_{ij}^{k} - \sum_{i=1}^{n}\sum_{\substack{j=1:\\ C_{ij}\geq c_{(1)}}}^{n} x_{ij}^{k-1}\right) \geq 0.$$

Given that $c_{(0)} = 0$, we get

$$\sum_{h=1}^{G}\left[c_{(h)}\left(\sum_{i=1}^{n}\sum_{\substack{j=1:\\ C_{ij}=c_{(h)}}}^{n} x_{ij}^{k} - \sum_{i=1}^{n}\sum_{\substack{j=1:\\ C_{ij}=c_{(h)}}}^{n} x_{ij}^{k-1}\right)\right] \geq 0$$

which is equivalent to

$$\sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij}x_{ij}^{k} - \sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij}x_{ij}^{k-1} \geq 0.$$

Finally, since $(x_{ij}, y_j, u_{kh}) \in P_2$ it satisfies $0 \leq x_{ij} \leq 1, 0 \leq u_{kh} \leq 1$. Hence, by (2.34) and (2.37) $0 \leq x_{ij}^{k} \leq 1$. Furthermore $0 \leq y_j \leq 1$ is satisfied. □

Next, we prove the relationship between the feasible regions of formulations $DOMP_3$ and $DOMP_2$.

Theorem 3. $f(P_3) \subset P_2$.

Proof. Let $(x_{ij}^k, y_j) \in P_3$, by (2.3) and (2.34), (2.12) is satisfied and by (2.5) and (2.34), (2.13) is satisfied.

We observe that (x_{ij}^k, y_j) satisfies

$$\sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} \geq c_{(h)}}}^{n} x_{ij}^k \geq \sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} \geq c_{(h+1)}}}^{n} x_{ij}^k.$$

Then using (2.35) we obtain $u_{kh} \geq u_{k,h+1}$ which proves (2.14).

In order to verify (2.15), by (2.4) we know that

$$\sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} < c_{(h)}}}^{n} x_{ij}^{k+1} + \sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} \geq c_{(h)}}}^{n} x_{ij}^{k+1} = 1$$

and by (2.21)

$$\sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} < c_{(h)}}}^{n} x_{ij}^{k+1} + \sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} \geq c_{(h)}}}^{n} x_{ij}^{k} \leq 1.$$

So,

$$\sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} \geq c_{(h)}}}^{n} x_{ij}^{k+1} \geq \sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} \geq c_{(h)}}}^{n} x_{ij}^{k}$$

and, using (2.35), constraint (2.15) is satisfied.

Equations (2.6) and (2.11) are the same.

It is clear that

$$\sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} > c_{(h-1)}}}^{n} \sum_{k=1}^{n} x_{ij}^k = \sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} \geq c_{(h)}}}^{n} \sum_{k=1}^{n} x_{ij}^k.$$

By (2.34), the LHS is

$$\sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} > c_{(h-1)}}}^{n} \sum_{k=1}^{n} x_{ij}^k = \sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} > c_{(h-1)}}}^{n} x_{ij},$$

and by (2.35) the RHS is

$$\sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} \geq c_{(h)}}}^{n} \sum_{k=1}^{n} x_{ij}^{k} = \sum_{k=1}^{n} u_{kh}.$$

By replacing both, (2.16) is satisfied.

In addition, all the variables are greater than or equal to zero and lower than or equal to one according with (2.34), (2.35), (2.3) and (2.4), since

$$x_{ij} = \sum_{k=1}^{n} x_{ij}^{k} \leq \sum_{j=1}^{n} \sum_{k=1}^{n} x_{ij}^{k} = 1,$$

$$u_{kh} = \sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} \geq c_{(h)}}}^{n} x_{ij}^{k} \leq \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}^{k} = 1.$$

□

As shown by Example 1 (Cont'd), the inclusion of $f(P_3)$ into P_2 is strict, i.e. there exists a point in P_2 which cannot be obtained as the image of a point from P_3 by mapping f.

Example 1 (Cont'd)**.** We consider matrix C of Example 1. Further, $n = 4$ and $p = 2$. We choose the optimal solution of the linear relaxation of $DOMP_2$ with $\lambda = (1, 1, 1, 3)$ and observe that it is a fractional vertex of P_2 (see Tables 2.1 and 2.2).

u_{kh}	$c_{(0)} = 0$	$c_{(1)} = 1$	$c_{(2)} = 2$	$c_{(3)} = 3$	$c_{(4)} = 4$	$c_{(5)} = 5$	$c_{(6)} = 6$	$c_{(7)} = 7$	$c_{(8)} = 9$
$k = 1$	1	$\frac{1}{2}$	0	0	0	0	0	0	0
$k = 2$	1	$\frac{1}{2}$	0	0	0	0	0	0	0
$k = 3$	1	$\frac{1}{2}$	0	0	0	0	0	0	0
$k = 4$	1	$\frac{1}{2}$	0	0	0	0	0	0	0

Table 2.1: **u**-values for a fractional feasible solution of Example 1 using formulation $DOMP_2$

Now, we show that there is no point in P_3 corresponding to that fractional solution in P_2. From $x_{ij} = \sum_{k=1}^{n} x_{ij}^{k}$ we deduce that $x_{ij}^{k} = 0$ for all i, j, k such

x_{ij}	$j=1$	$j=2$	$j=3$	$j=4$
$i=1$	1	0	0	0
$i=2$	1	0	0	0
$i=3$	0	0	1	0
$i=4$	0	0	1	0
y_j	1	0	1	0

Table 2.2: **x**-values and **y**-values for a fractional feasible solution of Example 1 using formulation $DOMP_2$

that $x_{ij} = 0$. Next, from the equation $u_{kh} = \sum_{\substack{i,j=1 \\ C_{ij} \geq c(h)}}^{n} x_{ij}^k$ we can conclude that point $(x_{ij}^k, y_j) \in P_3$ should satisfy:

$$x_{21}^1 + x_{43}^1 = \frac{1}{2},\ x_{21}^2 + x_{43}^2 = \frac{1}{2},\ x_{21}^3 + x_{43}^3 = \frac{1}{2},\ x_{21}^4 + x_{43}^4 = \frac{1}{2}. \tag{2.38}$$

Further, constraints (2.4) for $k = 2, 3$ state that: $x_{11}^2 + x_{33}^2 + x_{21}^2 + x_{43}^2 = 1$ and $x_{11}^3 + x_{33}^3 + x_{21}^3 + x_{43}^3 = 1$, respectively. Thus, combining these two equations with those above it results that $x_{11}^2 + x_{33}^2 = \frac{1}{2}$ and $x_{11}^3 + x_{33}^3 = \frac{1}{2}$.

On the other hand, the order constraints (2.21) for $i = 2, j = 1, k = 2$ and $i = 2, j = 1, k = 3$ require that x_{ij}^k fulfills $x_{11}^2 + x_{33}^2 + x_{21}^2 + x_{21}^1 + x_{43}^1 \leq 1$ and $x_{11}^3 + x_{33}^3 + x_{21}^3 + x_{43}^3 + x_{43}^2 \leq 1$, respectively. Combining these inequalities with the results above yields that $x_{21}^2 = x_{43}^2 = 0$ which contradicts the equation $x_{21}^2 + x_{43}^2 = \frac{1}{2}$ included in (2.38).

Our following result states the relationship between the feasible regions of the formulations $DOMP_3$ and $DOMP_1$.

Theorem 4. $P_3 \subset P_1$.

Proof. Using Theorems 2 and 3, it follows that

$$P_3 \subset g(P_2) \subset P_1.$$

□

Our next goal is to relate the polytope P_{3C} of the linear relaxation of $DOMP_{3C}$ with the previously considered polytopes.

Consider the following linear transformation

$$\begin{array}{rcl} L : [0,1]^{nG} & \longrightarrow & [0,1]^{nG}, \\ u_{kh} & \longmapsto & v_{kh} \end{array}$$

defined by the following equations

$$v_{kG} = u_{kG} \tag{2.39}$$

and

$$v_{kh} = u_{kh} - u_{k,h+1} \quad k = 1, \ldots, n, h = 0, \ldots, G-1. \tag{2.40}$$

Consider as well the inverse L^{-1}

$$u_{kh} = \sum_{h'=h}^{G} v_{kh'} \quad k = 1, \ldots, n, h = 0, \ldots, G. \tag{2.41}$$

Let us denote by z_l^{LP} the optimal value of the LP-relaxation of $DOMP_l$ for $l = 1, 2, 3, 4, 3C, 4C$.

Theorem 5. *The linear relaxation of formulation $DOMP_{3C}$ is equal to the linear relaxation of formulation $DOMP_2$ modulo the linear transformation L, i.e. $L(P_2) = P_{3C}$ and $z_2^{LP} = z_{3C}^{LP}$.*

Proof. First, we check that both objective functions are equivalent,

$$\sum_{k=1}^{n}\sum_{h=1}^{G} \lambda^k (c_{(h)} - c_{(h-1)}) u_{kh} = \sum_{k=1}^{n}\sum_{h=1}^{G} \lambda^k (c_{(h)} - c_{(h-1)}) \left(\sum_{h'=h}^{G} v_{kh'} \right) =$$

$$\sum_{k=1}^{n}\sum_{h=1}^{G} \lambda^k c_{(h)} \left(\sum_{h'=h}^{G} v_{kh'} \right) - \sum_{k=1}^{n}\sum_{h=1}^{G} \lambda^k c_{(h-1)} \left(\sum_{h'=h}^{G} v_{kh'} \right) =$$

$$\sum_{k=1}^{n}\sum_{h=1}^{G} \lambda^k c_{(h)} v_{kh} + \sum_{k=1}^{n}\sum_{h=1}^{G} \lambda^k c_{(h)} \left(\sum_{h'=h+1}^{G} v_{kh'} \right) - \sum_{k=1}^{n}\sum_{h=1}^{G} \lambda^k c_{(h-1)} \left(\sum_{h'=h}^{G} v_{kh'} \right) =$$

$$\sum_{k=1}^{n}\sum_{h=1}^{G} \lambda^k c_{(h)} v_{kh} = \sum_{k=1}^{n}\sum_{h=0}^{G} \lambda^k c_{(h)} v_{kh}.$$

Now, we show that the image $L^{-1}(P_{3C}) = P_2$. Then remarking that the linear transformation has full rank, the proof will be completed.

First, constraints (2.11), (2.12) and (2.13) are common to $DOMP3C$ and $DOMP2$.

Then, by definition, $u_{k0} = 1$, $k = 1, \ldots, n$. So

$$\sum_{h'=0}^{G} v_{kh'} = 1.$$

From (2.15),

$$\sum_{h'=h}^{G} v_{k+1,h'} \geq \sum_{h'=h}^{G} v_{kh'},$$

which is equivalent to

$$1 - \sum_{h'=0}^{h-1} v_{k+1,h'} \geq \sum_{h'=h}^{G} v_{kh'}.$$

And this is constraint (2.29). Writing (2.16) for h and $h+1$, we get

$$\sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} > c_{(h-1)}}}^{n} x_{ij} = \sum_{k=1}^{n} u_{kh} \quad \text{and} \quad \sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} > c_{(h)}}}^{n} x_{ij} = \sum_{k=1}^{n} u_{k,h+1},$$

and subtracting the second from the first we obtain

$$\sum_{i=1}^{n} \sum_{\substack{j=1: \\ C_{ij} = c_{(h)}}}^{n} x_{ij} = \sum_{k=1}^{n} u_{kh} - \sum_{k=1}^{n} u_{k,h+1} = \sum_{k=1}^{n} \sum_{h'=h}^{G} v_{kh'} - \sum_{k=1}^{n} \sum_{h'=h+1}^{G} v_{kh'} = \sum_{k=1}^{n} v_{kh},$$

which is equivalent to (2.28).

Finally, from (2.15) and (2.41) and the fact that $0 \leq u_{kh} \leq 1$, we obtain that $v_{kh} \in [0, 1]$. □

From the previous theorem one can easily obtain the relationship between the polytopes of feasible solutions of formulations $DOMP_{3C}$ and $DOMP_{4C}$.

Corollary 1. $P_{3C} \subset P_{4C}$.

We are now in position to present the overall relationship among the LP-relaxation values of all the considered formulations.

Corollary 2. $z_3^{LP} \geq z_{3C}^{LP} = z_2^{LP} \geq z_1^{LP}$, $z_3^{LP} \geq z_4^{LP}$ *and* $z_{3C}^{LP} \geq z_{4C}^{LP}$.

Proof. This is an immediate consequence of Observation 1 and Theorems 2, 3, 4 and 5. □

Observe that we have not proved a theoretical relationship between P_2 and P_4, but, as we will see in Section 2.4, empirically, we have observed that the solutions of the LP-relaxation of $DOMP_4$ provides in most cases a better bound than that of $DOMP_2$. However cases exist where it is the contrary. Theoretically, the formulation $DOMP_3$ gives a better or equal bound than $DOMP_2$ whose bound, as one can see in Section 2.4, concurs experimentally with the objective value of the linear relaxation of $DOMP_1$. Therefore, we know that $DOMP_3$ is the tightest formulation among the four presented in this chapter.

Using similar arguments as those developed in the proofs of Theorems 2 and 3 we can formally state the validity of our formulations $DOMP_3$ and $DOMP_4$.

Theorem 6.

1. *If* $(x_{ij}, y_j, u_{kh}) \in P_2^I$ *then* $g(x_{ij}, y_j, u_{kh}) \in P_3^I$. *Conversely, for any* $(x_{ij}^k, y_j) \in P_3^I$, $f(x_{ij}^k, y_j) \in P_2^I$.
2. *If* $(x_{ij}, y_j, u_{kh}) \in P_2^I$ *then* $g(x_{ij}, y_j, u_{kh}) \in P_1^I$. *Conversely, for any* $(x_{ij}^k, y_j) \in P_1^I$, $f(x_{ij}^k, y_j) \in P_2^I$.
3. $P_3^I = P_4^I$.

Proof. We prove Theorem 6 in three steps.

1. $\mathbf{g(P_2^I) = P_3^I}$
   (a) $\mathbf{g(P_2^I) \subset P_3^I}$
   Let $(x_{ij}, y_j, u_{kh}) \in P_2^I$ and consider its projection, (x_{ij}^k, y_j) defined by (2.34), (2.35) and (2.37) onto P_3. This point satisfies (2.3), (2.5), (2.6) and (2.21).
   In order to see that the point (x_{ij}^k, y_j) satisfies (2.4), by (2.35) we have $\sum_{i=1}^n \sum_{j=1}^n x_{ij}^k = u_{k0} \in \{0, 1\}$ for all $k = 1, \ldots, n$ and by (2.12) it follows that $\sum_{i=1}^n \sum_{j=1}^n x_{ij} = n$. Then using (2.34) we obtain $\sum_{k=1}^n \left(\sum_{i=1}^n \sum_{j=1}^n x_{ij}^k \right) = n$. Next, the only possibility for

the above sum to be equal to n is that $\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij}^{k} = 1$ for all $k = 1, \ldots, n$, and (2.4) is satisfied.

It remains to prove that (x_{ij}^{k}, y_j) is integer. It is clear that y_j is integer because it constitutes the same vector as in P_2. Finally, the x_{ij}^{k} variables are also binary due to the fact that they are the product of binary variables, see (2.37).

(b) $\mathbf{P_3^I \subset g(P_2^I)}$

To prove the converse, we observe by that every integer solution in P_3^I provides a projected integer solution and we have seen in Theorem 3 that this point satisfies the inequalities defining P_2.

2. $\mathbf{P_1^I = g(P_2^I)}$

(a) $\mathbf{g(P_2^I) \subset P_1^I}$

Every integer solution in P_2 is projected into an integer solution and, from Theorem 2 this point satisfies the inequalities defining P_1.

(b) $\mathbf{P_1^I \subset g(P_2^I)}$

Conversely, let $(x_{ij}^{k}, y_j) \in P_1^I$ and its projection (x_{ij}, y_j, u_{kh}) by means of (2.34) and (2.35).

It is easy to see that (2.3), (2.5) and (2.6) are equivalent to (2.12), (2.13) and (2.11), respectively.

Since all the variables are positive the point (x_{ij}^{k}, y_j) satisfies

$$\sum_{C_{ij} \geq c_{(h)}}^{n} x_{ij}^{k} \geq \sum_{C_{ij} \geq c_{(h+1)}}^{n} x_{ij}^{k},$$

and (2.14) holds.

Furthermore, (2.16) holds by the change of variable defined in (2.34) and (2.35).

In inequalities (2.7), we can order the cost without loss of generality. Next by (2.4), there will be a unique variable with value one in each position. Therefore we obtain the following constraint for each different cost,

$$\sum_{C_{ij} \geq c_{(h)}}^{n} x_{ij}^{k+1} \geq \sum_{C_{ij} \geq c_{(h)}}^{n} x_{ij}^{k} \quad h = 1, \ldots, G,\ k = 1, \ldots, n-1$$

and point (x_{ij}, y_j, u_{kh}) satisfies (2.15). Furthermore, this point is integer.

3. $\mathbf{P_3^I = P_4^I}$

   (a) $\mathbf{P_4^I \subset P_3^I}$
   Let $(x_{ij}^k, y_j) \in P_4^I$. In particular, the weak order constraint is satisfied. By (2.4) we have

$$\sum_{\substack{i'=1 \\ :i'j'}}^{n} \sum_{\substack{j'=1 \\ \preceq ij}}^{n} x_{i'j'}^k \leq 1$$

   and

$$\sum_{\substack{i'=1 \\ :i'j'}}^{n} \sum_{\substack{j'=1 \\ \succeq ij}}^{n} x_{i'j'}^{k-1} \leq 1,$$

   where both inequalities are a sum of binary variables. So there are two possibilities:

$$\sum_{i'=1}^{n} \sum_{\substack{j'=1: \\ i'j' \preceq ij}}^{n} x_{i'j'}^k + \sum_{i'=1}^{n} \sum_{\substack{j'=1: \\ i'j' \succeq ij}}^{n} x_{i'j'}^{k-1} \leq 1, \text{ or} \tag{2.42}$$

$$\sum_{i'=1}^{n} \sum_{\substack{j'=1: \\ i'j' \preceq ij}}^{n} x_{i'j'}^k + \sum_{i'=1}^{n} \sum_{\substack{j'=1: \\ i'j' \succeq ij}}^{n} x_{i'j'}^{k-1} = 2. \tag{2.43}$$

   In case (2.43), there will be two different costs $\hat{\imath}\hat{\jmath} \prec \tilde{i}\tilde{j}$ (note that $\hat{\imath}\hat{\jmath} = \tilde{i}\tilde{j}$ is not possible by (2.6)) such that $x_{\hat{\imath}\hat{\jmath}}^k = 1$ and $x_{\tilde{i}\tilde{j}}^{k-1} = 1$. This fact contradicts the ordering relationship. Thus, the only possibility is that case (2.42) holds and, in consequence, constraint (2.21) is satisfied.

   (b) $\mathbf{P_3^I \subset P_4^I}$
   Let (x_{ij}^k, y_j) be an integer point satisfying P_3^I. It is clear that this integer point also belongs to P_4^I.

□

Corollary 3. $P_1^I = P_3^I = P_4^I = g(P_2^I)$.

2.3.2 On the polytope defined by the assignment constraints

The goal of this section is to provide some results about the facial structure of the polytope corresponding to the assignment constraints of formulation $DOMP_3$. We restrict ourselves to the analysis of formulation $DOMP_3$ since, according to Corollary 2, it gives the tightest formulation to DOMP among those studied in this paper. Similar studies of (simpler) polytopes related to location problem have been carried out by e.g. Arbib et al. [2011], Guignard [1980], Cornuéjols and Thizy [1982], de Farias Jr. [2001] and Vasilyev et al. [2013].

For a given set J of p open facilities, we define the assignment polytope $P_3(J)$ of $DOMP_3$ as follows:

$$\sum_{j\in J}\sum_{k=1}^{n} x_{ij}^k \leq 1 \qquad i=1,\dots,n \tag{2.44}$$

$$\sum_{i=1}^{n}\sum_{j\in J} x_{ij}^k \leq 1 \qquad k=1,\dots,n \tag{2.45}$$

$$\sum_{k=1}^{n} x_{ij}^k \leq 1 \qquad i=1,\dots,n, j\in J \tag{2.46}$$

$$\sum_{i'j'\succeq ij} x_{i'j'}^{k-1} + \sum_{i'j'\preceq ij} x_{i'j'}^{k} \leq 1 \qquad i=1,\dots,n, j\in J, k=2,\dots,n \tag{2.47}$$

$$x_{ij}^k \geq 0 \qquad i,k=1,\dots,n, j\in J \tag{2.48}$$

We show which of the constraints that describe this polytope are facet inducing. Clearly, this contributes to the good quality of the LP-relaxation of $DOMP_3$, since this assignment polytope represents the underlying structure of the problem once the set of open facilities is determined.

We summarize the polyhedral properties of the convex hull of $P_3(J) \cap \{0,1\}^{n^2\times p}$, namely $P_3^I(J)$, in the following result.

Proposition 4.

1. *$dim(P_3(J)) = n^2p$.*
2. *Constraints (2.44) and (2.48) induce facets of $P_3^I(J)$.*

Proof. Constraints (2.44)-(2.48) define a particular packing polytope which has been studied by Padberg [1973] among others. The results are consequences of this observation and the fact that variables appearing in (2.44) define maximal cliques in the conflict graph associated to the problem, see Padberg [1973]. □

Observe that constraints (2.46) do not induce facets of $P_3(J)$ since they are dominated by constraints (2.44).

Let us denote by $(ij)^s, s = 1, \ldots, n^2$ the couple client i facility j where C_{ij} is the s-th lowest cost over the cost matrix C. For instance, $(ij)^1$ and $(ij)^{n^2}$ are, respectively, the most and the least preferred couples in the sorted list of costs of the cost matrix C.

Definition 7. *We call a pair ij generic if for some feasible set J such that $j \in J$, it satisfies*

1. *Let $\hat{\imath}\hat{\jmath}$ be the couple client $\hat{\imath}$ facility $\hat{\jmath} \in J$ such that no couple $i'j'$, $j' \in J$ satisfying $\hat{\imath}\hat{\jmath} \prec i'j' \prec ij$ does exist, i.e. $\hat{\imath}\hat{\jmath}$ is the couple immediately before ij. Then, $\hat{\imath} \neq i$.*

2. *Let $\tilde{\imath}\tilde{\jmath}$ be the couple client $\tilde{\imath}$ facility $\tilde{\jmath} \in J$ such that no couple $i'j'$, $j' \in J$ satisfying $ij \prec i'j' \prec \tilde{\imath}\tilde{\jmath}$ does exist, i.e. $\hat{\imath}\hat{\jmath}$ is the couple immediately after ij. Then, $\tilde{\imath} \neq i$.*

Intuitively, a pair ij is generic with respect to a feasible solution set J if the remaining feasible allocation costs are well distributed around it. That is, there are costs of different clients surrounding the ij cost (C_{ij}) in the sorted list of costs.

Proposition 5. *If ij is generic for the feasible set J then*

$$\sum_{i'j' \succeq ij} x^{k-1}_{i'j'} + \sum_{i'j' \preceq ij} x^{k}_{i'j'} + \sum_{l=k+1}^{n} x^{l}_{(ij)^1} + \sum_{l=1}^{k-2} x^{l}_{(ij)^{n^2}} \leq 1 \quad k = 2, \ldots, n \quad (2.49)$$

is facet defining for the assignment polytope $P_3^I(J)$.

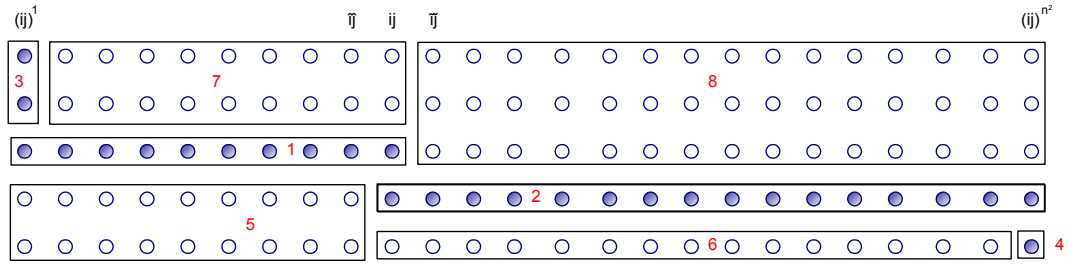


Figure 2.5: Constraints (2.49)'s scheme.

Proof. According to Definition 7, let us denote by îj and ĩj the couples immediately before and after the couple ij in the sorted list of costs (see Figure 2.5).

In the following we prove that this family of constraints are facet defining inequalities showing that they are maximal cliques. Specifically, we show that no additional variable can be added to those inequalities which is, at the same time, incompatible with all of those that already appear in it.

Next we prove the above claim. We show that for each variable that does not belong to the considered clique, there is one in the clique compatible with it.

1. For all $i'j' \prec ij$, $l \leq k-1$ if $i' \neq i$, variables $x^l_{i'j'}$ and x^k_{ij} are compatible. Otherwise, variables $x^l_{i'j'}$ and $x^k_{\tilde{ij}}$ are compatible. (Case 1 in Figure 2.5.)

2. For all $i'j' \succeq ij$, $l < k-1$ and $i'j' \neq (ij)^{n^2}$ if $i' \neq i$ variables $x^l_{i'j'}$ and x^k_{ij} are compatible. Otherwise, variables $x^l_{i'j'}$ and $x^k_{\hat{ij}}$ are compatible. (Case 2 in Figure 2.5.)

3. For all $i'j' \preceq ij$, $l = k+1, .., .n$ and $i'j' \neq (ij)^1$ if $i' \neq i$, variables $x^l_{i'j'}$ and x^{k-1}_{ij} are compatible. Otherwise, variables $x^l_{i'j'}$ and $x^{k-1}_{\tilde{ij}}$ are compatible. (Case 3 in Figure 2.5.)

4. For all $i'j' \succ ij$, $l = k, ..., n$ if $i' \neq i$ variables $x^l_{i'j'}$ and x^{k-1}_{ij} are compatible. Otherwise, variables $x^l_{i'j'}$ and $x^{k-1}_{\tilde{ij}}$ are compatible. (Case 4 in Figure 2.5.)

□

Remark that in our computational experiments, we did not reinforce constraints (2.47) because our preprocessing procedure (see Section 2.4) sets to zero all the variables appearing in these reinforcements associated with the costs of couples $(ij)^1$ and $(ij)^{n^2}$. Thus, in most cases in our formulation $DOMP_3$, after preprocessing those constraints are already facet inducing.

2.4 Computational study

In order to test the performance of our new formulations for DOMP, we have performed intensive computational tests comparing results with respect to previous available formulations of DOMP (see Boland et al. [2006], Domínguez-Marín [2003], Nickel [2001], Nickel and Puerto [2005]) and Marín et al. [2010]).

2.4.1 Description of the test instances

We use two different types of instances. First, we consider random instances in which the elements of the cost matrix are integer numbers randomly generated between 10000 and 100000. The second set of instances consists in p-median instances from OR_Lib, Beasley [2012].

Regarding the random instances; we vary the number of clients n in $\{10, 20, 30, 40, 50\}$ and for each n, we consider three possible values for the number of facilities to be open: $p = \left\lfloor \frac{n}{4} \right\rfloor, \left\lfloor \frac{n}{3} \right\rfloor, \left\lfloor \frac{n}{2} \right\rfloor$.

As for the p-median instances from Beasley's library, we have selected graphs corresponding to $p - med1, \ldots, p - med20$ with up to 400 nodes from the original data. Each set of nodes is divided in two disjoint subsets containing, respectively, the set of clients and the set of facilities. (The reader may observe that in these instances we force these two sets to be disjoint). Next, each cost C_{ij} is computed as the shortest path between client i and facility j in the resulting complete graph induced by the above described set of nodes.

Eight different types of λ-vectors are tested. Their description is provided in Table 2.3. We consider, among others, p-median, p-center, p-k-centrum, p-trimmed mean, random and p-α-centdian problems. In the k-centrum case, $k = \left\lfloor \frac{n}{2} \right\rfloor$. In the $(k_1 + k_2)$-trimmed mean, $k1 = k2 = \left\lfloor \frac{n}{10} \right\rfloor$. When λ is taken as a random vector we generate 5 instances which contain values randomly drawn between 1 and 100. Finally, we use $\alpha = 0.5$ in the centdian case.

Notation	λ-vector	Name
T1	$(1, 1, \ldots, 1, 1)$	p-median
T2	$(0, 0, \ldots, 0, 1)$	p-center
T3	$(0, 0, \ldots, 0, 0, \underbrace{1, 1, \ldots, 1, 1}_{k})$	k-centrum
T4	$(\underbrace{0, 0, \ldots, 0, 0}_{k_1}, 1, 1, \ldots, 1, 1, \underbrace{0, 0, \ldots, 0, 0}_{k_2})$	$(k_1 + k_2)$-trimmed mean
T5	$(0, 1, 0, 1, 0, 1, 0, 1, \ldots)$	–
T6	$(\ldots, 0, 0, 1, 0, 0, 1)$	–
T7	λ random	Random
T8	$(\alpha, \alpha, \ldots, \alpha, \alpha, 1)$	Centdian

Table 2.3: Types of λ-vectors used in experiments

For each type of data and each possible values of parameters n, p and vector λ, the results presented consist in average values over five instances. This results in an overall number of 900 tested instances.

2.4.2 Preprocessing

We present in this section two preprocessings that are based on a similar rationale to those already developed in Marín et al. [2009] and Puerto et al.; although adapted to the new formulations on this paper. The first one is based on feasibility and the second on optimality.

Claim 1. *(Feasibility based preprocessing)*

1. *Let* $l(h) = |\{i : \exists j \in \{1, \ldots, n\}$ *satisfying* $C_{ij} \leq c_{(h)}\}|$ *and* $u(h) = |\{i : \exists j \in \{1, \ldots, n\}$ *satisfying* $C_{ij} \geq c_{(h)}\}|$.

   Then,

   (a) $v_{kh} = 0$ *and* $u_{kh} = 1 \quad k = l(h) + 1, \ldots, n, h = 1 \ldots, G.$; *and*

   (b) $v_{kh} = 0$ *and* $u_{kh} = 0 \quad k = 1, \ldots, n - u(h), h = 1 \ldots, G.$

2. *Let* $l(ij) = |\{i' : \min_{\hat{\jmath}} i'\hat{\jmath} \prec ij, i' \neq i\}|$ *and* $u(ij) = |\{i' : \max_{\hat{\jmath}} i'\hat{\jmath} \succ ij, i' \neq i\}|$. *Then,*

   (a) $x_{ij}^k = 0$ *for all* $i, j = 1, \ldots, n,\ k = l(ij) + 2, \ldots, n$; *and*

(b) $x_{ij}^k = 0$ *for all* $i, j = 1, \ldots, n,\ k = 1, \ldots, n - u(ij) - 1$.

This claim formalizes the fact that a given cost C_{ij} can appear in position k of a feasible solution only if there are at least $k-1$ allocations costs lower than or equal to and $n-k$ greater than or equal to C_{ij}. We observe that this preprocessing can remove feasible solutions of the problem although it does not affect its solution since them cannot be optimal.

Claim 2. *Let* ij *be a couple client facility. If* $|\{j' : C_{ij} > C_{ij'}\}| > n - p$, *then* $x_{ij} = 0$ *and* $x_{ij}^k = 0$ *for* $k = 1, \ldots, n$.

This second claim removes some feasible solutions of the problems which cannot be optimal because they are dominated for other feasible solutions with smaller objective value.

Table 2.4 shows the percentage of variables v_{kh}, u_{kh}, x_{ij} and x_{ij}^k fixed by our preprocessing based on Claim 1 and 2. Notice that the percentage of two or three index variables fixed to zero in random instances is almost equal because assignment costs in random instances are almost all different. Furthermore, we observe that the percentage of fixed variables slightly decreases with n. Observe that Beasley instances have a larger number of ties in the distances (allocation costs). For this reason, these instances are solved using our compact formulations $DOMP_{3C}$ and $DOMP_{4C}$, whereas those with random data are solved with the formulations that do not take adventage of ties, namely $DOMP_3$ and $DOMP_4$. Hence, Beasley instances are preprocessed with Claim 1.1 (for v variables) and Random instances with Claim 1.2 (for x_{ij}^k variables).

n (Random instances)	10	20	30	40	50	**Average**
Claim 1.2	16.40%	9.17%	6.21%	4.67%	3.93%	**8.07%**
Claim 2	20.00%	25.00%	30.00%	29.99%	29.99%	**27.00%**
Total	29.68%	29.61%	30.43%	32.28%	31.97%	**31.40%**
n (Beasley instances)		50	100	150	200	**Average**
Claim 1.1		27.03%	31.86%	33.42%	31.01%	**30.83%**
Claim 2		28.91%	26.85%	25.87%	25.26%	**26.72%**
Total		27.38%	30.33%	29.67%	27.12%	**28.62%**

Table 2.4: Number of variables fixed by preprocessing

2.4.3 Computational results

All our experiments have been carried out on a PC with two Intel Xeon processors with 3.46 GHz and 48 GB of RAM. The models were written in Mosel and solved using Xpress IVE 7.3. To have a clean comparison of our solution approaches, all automatic cuts from Xpress have been disabled. We now report a summary of our computational experiments. Detailed information can be found in the material included in the Appendix A. In particular, we report results for the different types of lambda vectors from Table 2.3.

Random allocation costs data sets.

Table 2.5 provides a comparison of the LP-relaxations of models $DOMP_1$, $DOMP_2$, $DOMP_3$, $DOMP_4$ and $DOMP_{4\cap1}$, averaging for $n = 20$ and $n = 40$ and all possible values of p and λ. The last model $DOMP_{4\cap1}$ consists in $DOMP_4$ to which constraints (2.7) of $DOMP_1$ have been appended. Specifically, we report the integrality gap defined as $GAP = \frac{z^* - z_l^{LP}}{z^*}$, where z^* and z_l^{LP} represent the optimal value of $DOMP_l$ and its LP-relaxation, respectively. We observe that the values of the integrality gap vary between 2.29% and 7.34%, obtained in formulations $DOMP_3$ the best and $DOMP_2$ or $DOMP_1$ the worst. In all cases, the LP-gaps are good but specially in formulation $DOMP_3$ which seems to be rather tight. We remark also the small difference that is obtained adding constraints (2.7) to the formulation $DOMP_4$ in terms of integrality gap. Moreover, we point out that we have obtained the same LP-gap with formulations $DOMP_1$ and $DOMP_2$ for all the tested instances. (It is still an open question whether this is also theoretically true.) Thus, from Table 2.5 one could conclude that the best formulation is $DOMP_3$. In spite of that, the large number of inequalities ($O(n^3)$) used in the model makes it rather slow whenever the number of clients n is of moderate size ($n > 50$).

In order to define a solution approach which presents the best performance, we have conducted a preliminary computational test with instance sizes $n = 10, 20, 30$. Our first strategy consists in solving $DOMP_3$ with a pure branch-and-bound. Our second strategy, $DOMP_{4\cap1}(B\&B)$, solves $DOMP_{4\cap1}$ with a pure branch-and-bound. The third approach, $DOMP_{4\cap1}$ $(B\&C-3)$, starts by solving the LP-relaxation of $DOMP_4$ and then adds

inequalities (2.7) at the root node and *order constraints* (2.21) as long as they are violated by the current solution of the LP. The reader may observe that both families of inequalities are cliques in the conflict graph induced by the three index variables of our formulation. Therefore, *order constraints* could in principle be added by standard clique cuts generation techniques implemented in Xpress. Nevertheless, our own implementation is more efficient since the separation of the entire family of valid inequalities (2.21) can be performed in $O(n^3)$ by sequentially updating the l.h.s. value of the *order constraints* when switching from a couple ij to the adjacent one in the same position k. The fourth and last strategy $DOMP_4(B\&C-3)$ is a branch and cut algorithm based upon $DOMP_4$ adding only valid inequalities from (2.21).

Formulation	**GAP**
$DOMP_1$	7,34%
$DOMP_2$	7,34%
$DOMP_3$	2,29%
$DOMP_4$	6,27%
$DOMP_{4\cap1}$	6,06%

Table 2.5: Average integrality gaps

Solution approach	**Time**	**#nodes**
$DOMP_3(B\&B)$	463.24(8)	46.20
$DOMP_{4\cap1}(B\&B)$	18.64	1389.51
$DOMP_{4\cap1}(B\&C-3)$	52.25	24.94
$DOMP_4(B\&C-3)$	39.39	29.37

Table 2.6: CPU-Time and Number of nodes of the different formulations for $n = 10, 20, 30$.

Our results are reported in Table 2.6. There we have included the CPU-times, in seconds, and the number of nodes in the B&B tree for solving instances to optimality within 2 hours of CPU-time. The numbers between parentheses indicate the number of unsolved instances within the time limit. We observe that on average $DOMP_{4\cap1}(B\&B)$ is the strategy that solves problems faster even though it has to visit the largest number of nodes in the B&B tree. This is explained by the fact that it is the most compact formulation (with the smallest number of inequalities) and therefore, it can be easily solved at each node. On the other hand, $DOMP_3(B\&B)$ is the heaviest one (in terms of LP representation) giving rise to worse CPU-times although it visits few nodes in the searching phase. In between, we found the two branch-and-cut procedures that we have tested $DOMP_{4\cap1}(B\&C-3)$ and $DOMP_4(B\&C-3)$. In the implementation of this two $B\&C$ approaches

we have tested the separation of maximal clique inequalities over the conflict graph as an alternative to our own separation procedure. Nevertheless, our separation algorithm applied on inequalities (7) and (21) gives better results. From the above two tables, we can conclude that the best strategies to be tested in the intensive computational tests are $DOMP_{4\cap1}(B\&B)$ and, at times, $DOMP_4(B\&C-3)$.

To finish, Table 2.7 allows us to compare our best strategy, i.e. $DOMP_{4\cap1}$ $(B\&B)$, with a branch-and-bound approach based on $DOMP_2$, as well as to determine the size of instances that can be solved within a reasonable time limit of two hours. This table is organized in four columns. The first two columns show the size n and p of the instances. The last two columns show the average CPU-time in seconds necessary for solving those instances applying formulation $DOMP_2(B\&B)$ and $DOMP_{4\cap1}(B\&B)$. The numbers between parentheses indicate the number of unsolved instances within the time limit of 2 hours.

From Table 2.7 we remark that $DOMP_{4\cap1}(B\&B)$ performs similarly as $DOMP_2(B\&B)$ (it improves the behavior of $DOMP_2(B\&B)$ only for some instance sizes). It is better for data instances with $n < 40$ in all combinations of p. In addition, for larger n, i.e. $n = 40$, 50, it is also better except if p is relatively small as compared with n. Furthermore, we also observe that for $n = 50$ both models fail to solve some instances for the smallest tested value of $p = 12$. Finally, Table 2.8 allows us to conclude that three index with aggregated scheduling constraints performs the best whenever the number of facilities to be located is not too small compared to the number of possible locations.

n	p	Time (# unsolved)	
		DOMP$_{\mathbf{2}}$(B&B)	**DOMP$_{\mathbf{4\cap 1}}$(B&B)**
10	2	1.12	0.42
10	3	0.54	0.25
10	5	0.18	0.09
20	5	22.21	5.80
20	6	9.05	4.09
20	10	3.33	1.54
30	7	161.32	121.04
30	10	66.85	22.70
30	15	33.15	11.87
40	10	449.11	625.68
40	13	232.79	174.62
40	20	116.99	49.13
50	12	1870.80(2)	3184.54(6)
50	16	988.08	804.36
50	25	771.51(2)	157.44
Average		**315.14**	**344.24**

Table 2.7: Summary of results with random matrices

p	**DOMP$_{\mathbf{2}}$(B&B)**	**DOMP$_{\mathbf{4\cap 1}}$(B&B)**
$\lfloor \frac{n}{4} \rfloor$	500.91(2)	787.49(6)
$\lfloor \frac{n}{3} \rfloor$	259.46	201.20
$\lfloor \frac{n}{2} \rfloor$	185.03(2)	44.01

Table 2.8: CPU-Time of the different formulations for different values of p.

Beasley's data set.

The large number of ties within the cost matrices of this data set suggests that on this second part of the study $DOMP_{4C}$ is the appropriate formulation to solve the problems. For each cost matrix we solve each instance

Problem	n	p	**Time (#unsolved)**	
			$\mathbf{DOMP_2(B\&B)}$	$\mathbf{DOMP_{4C}(B\&C-3C)}$
pmed1	50	5	94.65	94.66
pmed2	50	10	79.24	40.58
pmed3	50	10	103.84	37.58
pmed4	50	20	34.94	8.57
pmed5	50	33	20.44	3.90
pmed6	100	5	805.02	4757.27(5)
pmed7	100	10	617.60	2708.85(2)
pmed8	100	20	848.95(1)	543.26
pmed9	100	40	130.28	45.87
pmed10	100	67	45.84	22.01
pmed11	150	5	1353.85(1)	3702.40(2)
pmed12	150	10	1667.06(1)	4235.25(4)
pmed13	150	30	2030.82(1)	3808.45(3)
pmed14	150	60	509.60	302.93
pmed15	150	100	88.48	52.17
pmed16	200	5	2760.99(1)	5940.66(6)
pmed17	200	10	2940.22(1)	5670.05(6)
pmed18	200	40	1830.27(1)	4129.70(3)
pmed19	200	80	358.11	264.35
pmed20	200	133	256.21	150.76

Table 2.9: Summary of results using Beasley's data set

with $DOMP_2(B\&B)$ and $DOMP_{4C}(B\&C-3C)$, i.e. formulation $DOMP_{4C}$ within a branch and cut scheme separating inequalities (2.29). For each value of n we solve those problems for the number of open facilities p, suggested in the original data from Beasley's library, and for all the considered vectors of λ shown in Table 2.3.

Table 2.9 shows the average results for these instances. Detailed information for each λ can be found in the Appendix A.3. This table is organized in five columns. The first three columns show the name of the instance problem and its size n and p. The last two columns show the CPU-time for solving those instances applying strategies $DOMP_2(B\&B)$ and $DOMP_{4C}(B\&C-3C)$. The numbers between parentheses indicate the number of instances that could not be solved to optimality within the time limit of 2 hours. We can see that in 11 out of 20 instances $DOMP_{4C}(B\&C-3C)$ is

faster than $DOMP_2(B\&B)$. This behavior confirms that both formulations have a rather similar performance. In spite of that, we observe that the use of our new formulation outperforms $DOMP_2$ provided that n is of moderate size $n < 50$ or whenever the size of p relative to n is not too small, namely $p/n \geq 0.2$. This behavior allows us to conclude that $DOMP_{4C}(B\&C - 3C)$ is advisable to be used, at least, in those cases.

Chapter 3

A Branch-and-Cut-and-Price procedure for the discrete ordered median problem

3.1 Introduction

In Chapter 2, we propose new formulations for DOMP, based on a set covering approach (Labbé et al. [2016]), that give rise to rather tight integrality GAP's and that are reasonably efficient to solve medium size instances when embedded in a Branch-and-Cut-and-Bound scheme. In this chapter we explore a different paradigm for solving DOMP based on an extended formulation with an exponential number of variables coming from a set partition model where each element (in the partition) is a set of pairs (client, position). These clients are served by the same facility and their allocation costs must be sorted in the given positions in any feasible solutions that they belong to. To handle the exponential number of variables we use a column generation approach that is embedded in a Branch-and-Cut algorithm. This idea has never been applied to DOMP and it opens new avenues of research.

This chapter is organized as follows. After this introduction, Section 3.2 introduces a new set partition formulation for DOMP. This formulation uses an exponential number of variables where each element of the partition is a set of clients together with their sorted positions that are assigned to the same server. Section 3.3 describes the column generation algorithm that we have designed to overcome the large number of variables in the model. We prove

that the pricing subproblem is solvable efficiently in polynomial time by using an *ad hoc* dynamic programming algorithm. In addition, we also develop a stabilization routine, based in Pessoa et al. [2010], that reduces considerably the number of iterations of the column generation approach. Section 3.4 presents the branch-and-cut-and-price algorithm for DOMP. This is a branch-and-cut scheme that solves the linear relaxation of each node of the branching tree with the column generation algorithm previously described in Section 3.3. This section contains 5 subsections where we have developed the building blocks of this approach: preprocessing, warm-start phase (initial pool of columns), initial feasible solution (grasp heuristic), branching rule and valid inequalities to be embedded in the branch-and-cut-and-price algorithm. The final section, namely Section 3.5 is devoted to report on the final computational experiments of this chapter. Here, we report on how the performance of the solution (CPU times and number of explored nodes in the branching tree) is improved by using the cuts introduced before. Besides, we also compare the performance of branch-and-cut-and-price algorithm presented in this chapter against the formulation $DOMP_4(B\&B-3)$.

3.2 The problem and its formulation

In this chapter, we elaborate upon a formulation that uses binary variables x_{ij}^k equal to 1 if client i is allocated to facility j in position k; and y_j that assumes value 1 if facility j is open and zero otherwise. This formulation is defined and explained in Section 2.2.4.

$$(DOMP_4)\ \min \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} \lambda^k C_{ij} x_{ij}^k \tag{3.1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n}\sum_{k=1}^{n} x_{ij}^k = 1 \qquad i = 1,\ldots,n \tag{3.2}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij}^k = 1 \qquad k = 1,\ldots,n \tag{3.3}$$

$$\sum_{k=1}^{n} x_{ij}^k \le y_j \qquad i,j = 1,\ldots,n \tag{3.4}$$

$$\sum_{j=1}^{n} y_j = p \tag{3.5}$$

$$\sum_{i}^{n}\sum_{j}^{n}\left(\sum_{i'=1}^{n}\sum_{\substack{j'=1:\\ i'j'\preceq ij}}^{n} x_{i'j'}^k + \sum_{i'=1}^{n}\sum_{\substack{j'=1:\\ i'j'\succeq ij}}^{n} x_{i'j'}^{k-1}\right) \le n^2, \quad k = 2,\cdots,n \tag{3.6}$$

$$x_{ij}^k \in \{0,1\} \qquad i,j,k = 1,\ldots,n \tag{3.7}$$

$$y_j \in \{0,1\} \qquad j = 1,\ldots,n \tag{3.8}$$

From a linear programming relaxation point of view this formulation is not the strongest one but it provides an interesting compromise by requiring a reasonable solution time. Further, it allows to solve to optimality problems of moderate size. One of its drawbacks is its use of a cubic number of variables, which can be prohibitive for large n. This is especially true because the linear programming relaxation can be extremely fractional. A second important problem of all known formulations for DOMP is the high degree of symmetry in case of equal costs or lambda.

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

Let us consider sets of couples (i,k) where the first component refers to client i and the second to position k, namely $S = \{(i,k) :$ for some $i, k = 1,\ldots,n\}$. Associated with each set S and facility j, we define variables

$$y_S^j = \begin{cases} 1 & \text{if set } S \text{ is part of a feasible solution, i.e. all its clients} \\ & \text{are allocated to facility } j \text{ that must be open.} \\ 0 & \text{otherwise.} \end{cases}$$

We observe that in any feasible solution each client i must occupy a unique sorted position k and must be allocated to a unique facility j, thus the following relationship holds $x_{ij}^k = \sum_{S \ni (i,k)} y_S^j$, for all i, j, k.

Next, assuming that all clients in S are allocated to facility j and that the positions that appear in the second entry of the couples (i, k) of the set S satisfy the sorting among their allocation costs, i.e. $C_{ij}^k \leq C_{i'j}^{k'}$ whenever $k \leq k'$, we can evaluate the cost c_S^j induced by the set S provided that its clients are assigned to facility j in a feasible solution:

$$c_S^j = \sum_{(i,k) \in S} \lambda^k C_{ij}. \tag{3.9}$$

To simplify the presentation in the following we denote by $(i, \cdot)$ the couples whose first entry is i regardless of the value of the second entry. Analogously, $(\cdot, k)$ denotes the couples whose second entry is k regardless of the value of the first entry. We give next a valid formulation for DOMP using this set of variables and based on the above partition in disjoint pairs idea. This will be our Master Problem in Section 3.3, so we name it MP.

$$\textbf{(MP)}\ \min \quad \sum_{j=1}^{n} \sum_{S} c_S^j y_S^j \tag{3.10}$$

$$s.t. \quad \sum_{j=1}^{n} \sum_{S \ni (i,\cdot)} y_S^j = 1, \forall\, i \tag{3.11}$$

$$\sum_{j=1}^{n} \sum_{S \ni (\cdot,k)} y_S^j = 1, \forall\, k \tag{3.12}$$

$$\sum_{S} y_S^j \leq 1, \forall\, j \tag{3.13}$$

$$\sum_{j=1}^{n} \sum_{S} y_S^j \leq p, \tag{3.14}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \left(\sum_{\substack{S \ni (\hat{\imath},k) \\ :C_{\hat{\imath}\hat{\jmath}} \leq C_{ij}}} y_S^{\hat{\jmath}} + \sum_{\substack{S \ni (\hat{\imath},k-1) \\ :C_{\hat{\imath}\hat{\jmath}} \geq C_{ij}}} y_S^{\hat{\jmath}} \right) \leq n^2, k = 2, \ldots, n \tag{3.15}$$

$$y_S^j \in \{0,1\}, \forall\, S, j, \tag{3.16}$$

The objective function (3.10) accounts for the sorted weighted cost of any feasible solution. Observe that it corresponds to the translation of (3.1). Next, constraints (3.11) ensure that each client appears exactly once in a set S as a part of a feasible solution. Constraints (3.12) ensure that each position is taken exactly once by a client in a set S as a part of a feasible solution. Constraints (3.13) assures that each facility j serves at most one set S of clients. The inequality (3.14) states that at most p facilities will be opened. By the following family of inequalities (3.15) we enforce the correct sorting of the costs in any feasible solution. Finally, the variables are binary. We note in passing that this formulation is not a Dantzig-Wolfe reformulation but a new formulation based on the properties of the problem.

Constraints (3.13) can be removed from the above formulation without affecting its validity. A naive approach suggests that in most instances the LP-bound will not be worsened, although this may happen with some of them. To test this fact we performed a computational study with 90 instances and we found some, although very few, instances where the linear relaxation with and without the family of constraints (3.13) is different. However, the

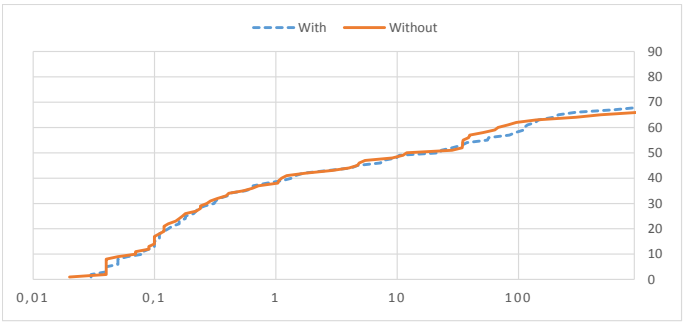


Figure 3.1: Number of solved problems per time with or without constraint (3.13).

average CPU time to solve the problems in both cases (including or not the family of constraints (3.13)) is similar, as one can see on the performance profile shown in Figure 3.1. Base on this profile, it is convenient to include this family of constraints and thus, they are included in the approach followed in the rest of the chapter.

One can prove that the linear programming relaxation of MP is stronger than that of $(DOMP_4)$. Let P_{MP} and P_{DOMP_4}, denote, respectively, the polyhedra defined by the feasible domains of MP and $DOMP_4$ relaxing the integrality constraints. Moreover, let N be the dimension of the space of variables y_S^j defined above and consider the following mapping

$$\begin{array}{rcl} f : [0,1]^N & \longrightarrow & [0,1]^{n^3} \times [0,1]^n \\ (y_s^j) & \longmapsto & (x_{ij}^k, y_j) \end{array}$$

defined by the following two equations

$$x_{ij}^k = \sum_{S \ni (i,k)} y_S^j \quad i,j,k = 1,\ldots,n \tag{3.17}$$

and

$$y_j = \sum_{S} y_S^j \quad j = 1,\ldots,n. \tag{3.18}$$

Proposition 6. *Let $p = (y_S^j)$ if $p \in P_{MP}$ then $f(p) \in P_{DOMP_4}$.*

Proof. Let us assume that $p \in P_{MP}$. We prove that $f(p)$ satisfies (3.2)-(3.6). To prove (3.2) observe that according with the definition of x_{ij}^k in (3.17)

$\sum_{S \ni (i,.)} y_S^j = \sum_{k=1}^{n} x_{ij}^k$. Therefore, substituting in (3.11) we get the result. Checking the validity of (3.3) is analogue.

Now, we prove (3.4). Observe that by (3.17) $\sum_{k=1}^{n} x_{ij}^k = \sum_{k=1}^{n} \sum_{S \ni (i,k)} y_S^j = \sum_{S \ni (i,\cdot)} y_S^j$ and then

$$\sum_{S \ni (i,\cdot)} y_S^j \leq \sum_{k=1}^{n} \sum_{S} y_S^j \leq 1$$

this last inequality holds by (3.13) which proves (3.4). To check (3.5) we replace (3.18) on (3.14) to obtain $\sum_{j=1}^{n} y_j \leq 1$. The equality follows because setting extra y_j variables to 1 do not worsen the objective function since all costs $C_{ij}^k \geq 0$. Finally, (3.15) follows analogously substituting (3.17) in (3.6). □

Hence, it is clear that the bound obtained by the linear relaxation of MP, from now on LRMP, is at least as good as the bound provided by the linear relaxation of $DOMP_4$.

Due to the fact that MP can have a number of variables too large to be handled directly, in the next section we describe a column generation approach to solve this problem.

3.3 Column generation to solve LRMP

The exponential number of variables that define LRMP makes it difficult to be directly handled by linear programming solvers and thus we will solve it using column generation. We begin by obtaining the dual of LRMP. In order to do that let $(\alpha, \beta, \gamma, \delta, \epsilon)$ be the dual variables associated, respectively, to constraints (3.11), (3.12), (3.13), (3.14) and (3.15). Then, DP, the dual

problem of LRMP is

$$
\begin{aligned}
\textbf{(DP)}\ \max\ & \sum_{i=1}^{n}\alpha_i+\sum_{k=1}^{n}\beta_k-\sum_{j=1}^{n}\gamma_j-p\delta-\sum_{k=2}^{n}n^2\epsilon_k \\
s.t.\ & \sum_{\substack{i=1\\:(i,\cdot)\in S}}^{n}\alpha_i+\sum_{\substack{k=1\\:(\cdot,k)\in S}}^{n}\beta_k-\gamma_j-\delta \\
& -\sum_{k=2}^{n}\sum_{\hat{\imath}=1}^{n}\sum_{\hat{\jmath}=1}^{n}\left(\sum_{\substack{(i,k)\in S\\:C_{\hat{\imath}\hat{\jmath}}\geq C_{ij}}}\epsilon_k+\sum_{\substack{(i,k-1)\in S\\:C_{\hat{\imath}\hat{\jmath}}\leq C_{ij}}}\epsilon_k\right) \leq c_S^j \quad \forall\, j,S \\
& \gamma_j \geq 0 \quad \forall\, j \\
& \delta \geq 0 \\
& \epsilon_k \geq 0 \quad \forall\, k.
\end{aligned}
$$

In order to apply the column generation procedure let us assume that we are given a set of columns that defines a Reduced Master Problem, ReMP. This problem is solved to optimality and we get its dual optimal variables $(\alpha^*,\beta^*,\gamma^*,\delta^*,\epsilon^*)$. See example 2. Then, the reduced cost, $\overline{c}_S^j$, of the column y_S^j, namely $\overline{c}_S^j=c_S^j-z_S^j$ is given as:

$$
\overline{c}_S^j=c_S^j+\gamma_j^*+\delta^*+\sum_{k=2}^{n}\sum_{\hat{\imath}=1}^{n}\sum_{\hat{\jmath}=1}^{n}\left(\sum_{\substack{(i,k)\in S\\:C_{\hat{\imath}\hat{\jmath}}\geq C_{ij}}}\epsilon_k^*+\sum_{\substack{(i,k-1)\in S\\:C_{\hat{\imath}\hat{\jmath}}\leq C_{ij}}}\epsilon_k^*\right)-\sum_{\substack{i=1\\:(i,\cdot)\in S}}^{n}\alpha_i^*-\sum_{\substack{k=1\\:(\cdot,k)\in S}}^{n}\beta_k^*.
$$

Example 2. *Consider the following cost matrix:*

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

and the vector $\lambda=(2,1,0.5)$. *For* $n=3$, *there are 33 different sets of pairs*

(i,k).

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

We consider as initial pool of columns the variables y_8^1, y_8^2 and y_{18}^3. With this set of variables, the ReMP is

$$
\begin{array}{rrrrrrl}
\textbf{(ReMP)} \min & +6y_8^1 & +8y_8^2 & +y_{18}^3 & & & \\
s.t. & & & +y_{18}^3 & \geq 1 & i=1 \\
 & & & +y_{18}^3 & \geq 1 & i=2 \\
 & +y_8^1 & +y_8^2 & & \geq 1 & i=3 \\
 & & & +y_{18}^3 & \geq 1 & k=1 \\
 & +y_8^1 & +y_8^2 & & \geq 1 & k=2 \\
 & & & +y_{18}^3 & \geq 1 & k=3 \\
 & -y_8^1 & & & \geq -1 & j=1 \\
 & & -y_8^2 & & \geq -1 & j=2 \\
 & & & -y_{18}^3 & \geq -1 & j=3 \\
 & -y_8^1 & -y_8^2 & -y_{18}^3 & \geq -2 & \\
 & -4y_8^1 & -2y_8^2 & -7y_{18}^3 & \geq -9 & k=2 \\
 & -6y_8^1 & -8y_8^2 & -y_{18}^3 & \geq -9 & k=3 \\
 & & & y & \geq 0 &
\end{array}
$$

Actually, we are interested in its dual problem:

$$
\begin{array}{rccccccccccccll}
\textbf{(DP)} \max & +\alpha_1 & +\alpha_2 & +\alpha_3 & +\beta_1 & +\beta_2 & +\beta_3 & -\gamma_1 & -\gamma_2 & -\gamma_3 & -2\delta & -9\epsilon_2 & -9\epsilon_3 & & \\
s.t. & & & +\alpha_3 & & +\beta_2 & & -\gamma_1 & & & -\delta & -4\epsilon_2 & -6\epsilon_3 & \leq 6 & (y_8^1) \\
 & & & +\alpha_3 & & +\beta_2 & & & -\gamma_2 & & -\delta & -2\epsilon_2 & -8\epsilon_3 & \leq 8 & (y_8^2) \\
 & +\alpha_1 & +\alpha_2 & & +\beta_1 & & +\beta_3 & & & -\gamma_3 & -\delta & -7\epsilon_2 & -\epsilon_3 & \leq 18.5 & (y_{18}^3)
\end{array}
$$

$$\alpha, \beta, \gamma, \delta, \epsilon \geq 0$$

Solving (DP) the solution is $\alpha_1 = 25.5, \alpha_3 = 10, \epsilon_2 = 1$ and the value of the objective function is $f = 26.5$.

Next, if $\overline{c}_S^j \geq 0$ for all S, j the current solution of ReMP is also optimal for the LRMP and the column generation procedure is finished.

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

$$z_{ReMP} + p \cdot \min_{j,S} \overline{c}_S^j \leq z_{MP} \leq z_{ReMP},$$

or

$$z_{ReMP} + \sum_{j=1}^{n} \min_{S} \overline{c}_S^j \leq z_{MP} \leq z_{ReMP}.$$

3.3.1 Solving the pricing subproblem

Although any column y_S^j with negative reduced cost may be added to ReMP, we will follow a strategy that identifies the most negative reduced cost for each facility j. This approach may give rise to several candidate columns (multiple pricing, see Chvátal [1983]), which is advantageous for this procedure.

In order to do that, we solve for each facility j a subproblem to find the column with minimum reduced cost associated with a feasible set S, namely a solution that satisfies that there are at most one pair $(i, \cdot)$ for each client i and one pair $(\cdot, k)$ for each position k. Furthermore, the set S must satisfy that the allocation costs of its couples are ranked accordingly. We solve this problem by the following dynamic programming algorithm. The reader may gain some intuition interpreting the algorithm as a shortest path in a graph built upon the matrix D_j defined in (3.19).

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

d_{ij}^k is given as

$$d_{ij}^k = \begin{cases} \lambda^k C_{ij} + \sum_{\hat{\imath}=1}^{n} \sum_{\substack{\hat{\jmath}=1 \\ :C_{\hat{\imath}\hat{\jmath}} \le C_{ij}}} \epsilon_{k+1} - \alpha_i - \beta_k & \text{if } k = 1 \\ \lambda^k C_{ij} + \sum_{\hat{\imath}=1}^{n} \sum_{\substack{\hat{\jmath}=1 \\ :C_{\hat{\imath}\hat{\jmath}} \ge C_{ij}}}^{n} \epsilon_k + \sum_{\hat{\imath}=1}^{n} \sum_{\substack{\hat{\jmath}=1 \\ :C_{\hat{\imath}\hat{\jmath}} \le C_{ij}}} \epsilon_{k+1} - \alpha_i - \beta_k & \text{if } k = 2, \ldots, n-1 \\ \lambda^k C_{ij} + \sum_{\hat{\imath}=1}^{n} \sum_{\substack{\hat{\jmath}=1 \\ :C_{\hat{\imath}\hat{\jmath}} \ge C_{ij}}}^{n} \epsilon_k - \alpha_i - \beta_k, & \text{if } k = n. \end{cases}$$

Now for each facility j, we define the matrix $D_j = (d_{i_l j}^k)_{i_l k}$, namely

$$D_j = \begin{pmatrix} d_{i_1 j}^1 & d_{i_1 j}^2 & \cdots & d_{i_1 j}^n \\ d_{i_2 j}^1 & & & \\ \vdots & & \ddots & \\ d_{i_n j}^1 & & & d_{i_n j}^n \end{pmatrix} \tag{3.19}$$

where $i_1, i_2, \ldots, i_n$ is a permutation which ensures $C_{i_1 j} \le C_{i_2 j} \le \cdots \le C_{i_n j}$.

Example 2 (Cont'd)**.** *Next, we show the procedure that computes the elements d_{ij}^k for all $i, k = 1, \ldots, n$ of the matrix D_1. (**j=1**)*

$$d_{11}^1 = \lambda^1 C_{11} + \sum_{\hat{\imath}=1}^{n} \sum_{\substack{\hat{\jmath}=1 \\ :C_{\hat{\imath}\hat{\jmath}} \le C_{11}}} \epsilon_2 - \alpha_1 - \beta_1 = 2 + 1 - 25.5 - 0 = -22.5$$

$$d_{11}^2 = \lambda^2 C_{11} + \sum_{\hat{\imath}=1}^{n} \sum_{\substack{\hat{\jmath}=1 \\ :C_{\hat{\imath}\hat{\jmath}} \ge C_{11}}}^{n} \epsilon_2 + \sum_{\hat{\imath}=1}^{n} \sum_{\substack{\hat{\jmath}=1 \\ :C_{\hat{\imath}\hat{\jmath}} \le C_{11}}} \epsilon_3 - \alpha_1 - \beta_2 = 1 + 9 + 0 - 25.5 - 0 = -15.5$$

$$d_{11}^3 = \lambda^3 C_{11} + \sum_{\hat{\imath}=1}^{n} \sum_{\substack{\hat{\jmath}=1 \\ :C_{\hat{\imath}\hat{\jmath}} \ge C_{11}}}^{n} \epsilon_3 - \alpha_1 - \beta_3 = 0.5 + 0 - 25.5 - 0 = -25$$

$$d_{21}^1 = \lambda^1 C_{21} + \sum_{\hat{\imath}=1}^{n} \sum_{\substack{\hat{\jmath}=1 \\ :C_{\hat{\imath}\hat{\jmath}} \le C_{21}}} \epsilon_2 - \alpha_2 - \beta_1 = 8 + 4 - 0 - 0 = 12$$

$$d_{21}^2 = \lambda^2 C_{21} + \sum_{\hat{\imath}=1}^{n} \sum_{\substack{\hat{\jmath}=1 \\ :C_{\hat{\imath}\hat{\jmath}} \ge C_{21}}}^{n} \epsilon_2 + \sum_{\hat{\imath}=1}^{n} \sum_{\substack{\hat{\jmath}=1 \\ :C_{\hat{\imath}\hat{\jmath}} \le C_{21}}} \epsilon_3 - \alpha_2 - \beta_2 = 4 + 6 + 0 - 0 - 0 = 10$$

$$d_{21}^3 = \lambda^3 C_{21} + \sum_{\hat{\imath}=1}^{n} \sum_{\substack{\hat{\jmath}=1 \\ :C_{\hat{\imath}\hat{\jmath}} \geq C_{21}}}^{n} \epsilon_3 - \alpha_2 - \beta_3 = 2 + 0 - 0 - 0 = 2$$

$$d_{31}^1 = \lambda^1 C_{31} + \sum_{\hat{\imath}=1}^{n} \sum_{\substack{\hat{\jmath}=1 \\ :C_{\hat{\imath}\hat{\jmath}} \leq C_{31}}} \epsilon_2 - \alpha_3 - \beta_1 = 12 + 6 - 10 - 0 = 8$$

$$d_{31}^2 = \lambda^2 C_{31} + \sum_{\hat{\imath}=1}^{n} \sum_{\substack{\hat{\jmath}=1 \\ :C_{\hat{\imath}\hat{\jmath}} \geq C_{31}}}^{n} \epsilon_2 + \sum_{\hat{\imath}=1}^{n} \sum_{\substack{\hat{\jmath}=1 \\ :C_{\hat{\imath}\hat{\jmath}} \leq C_{31}}} \epsilon_3 - \alpha_3 - \beta_2 = 6 + 4 + 0 - 10 - 0 = 0$$

$$d_{31}^3 = \lambda^3 C_{31} + \sum_{\hat{\imath}=1}^{n} \sum_{\substack{\hat{\jmath}=1 \\ :C_{\hat{\imath}\hat{\jmath}} \geq C_{31}}}^{n} \epsilon_3 - \alpha_2 - \beta_3 = 3 + 0 - 10 - 0 = -7$$

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

$$D_1 = \begin{pmatrix} -22.5 & -15.5 & -25.0 \\ -12.0 & 10.0 & 2.0 \\ 8.0 & 0.0 & -7.0 \end{pmatrix} \begin{matrix} i = 1 \\ i = 2 \\ i = 3 \end{matrix}$$

We present next the recursion to get the minimum reduced cost $\overline{c}_S^j$ for each $j = 1, \ldots, n$. For each element (i_l, k) we need to compute $g_{i_l}^k$, the minimum value of the set of pairs in the rectangular block of D_j with $(i_1, 1)$ as upper left corner and (i_l, k) as lower right corner satisfying that there are at most one couple per row and column and that for any two couples $(i_{\hat{l}}, \hat{k})$ and $(i_{\tilde{l}}, \tilde{k})$ such that $\hat{k} < \tilde{k}$, then $C_{i_{\hat{l}}j} < C_{i_{\tilde{l}}j}$.

Observe that $g_{i_l}^k$ can be computed by means of the following algorithm:

- **Step 0**

  $$g_{i_1}^1 = \min\{0, d_{i_1}^1\}$$

  If $g_{i_1}^1 = d_{i_1}^1 < 0$ we store the pair $(i_1, 1)$.

- **Step 1**. For $k = 1, \ldots, n$.

  $$g_{i_1}^k = \min\{d_{i_1}^k, g_{i_1}^{k-1}\}$$

  If $g_{i_1}^k = g_{i_1}^{k-1}$ we store the pairs related with $g_{i_1}^{k-1}$.

  Else, we store the pair (i_1, k).

- **Step 2**. For $l = 1, \ldots, n$.

$$g^1_{i_l} = \min\{d^1_{i_l}, g^1_{i_{l-1}}\}$$

If $g^1_{i_l} = g^k_{i_{l-1}}$ we store the pairs related with $g^k_{i_{l-1}}$.
Else, we store the pair $(i_l, 1)$.

- **Step 3**. For $l, k = 2, \ldots, n$.

$$g^k_{i_l} = \min\{g^{k-1}_{i_{l-1}}, d^k_{i_l} + g^{k-1}_{i_{l-1}}, g^{k-1}_{i_l}, g^k_{i_{l-1}}\}$$

If $g^k_{i_l} = g^{k-1}_{i_{l-1}}$ we store the pairs related with $g^{k-1}_{i_{l-1}}$.
Else, if $g^k_{i_l} = g^{k-1}_{i_l}$ we store the pairs related with $g^{k-1}_{i_l}$.
Else, if $g^k_{i_l} = g^k_{i_{l-1}}$ we store the pairs related with $g^k_{i_{l-1}}$.
Else, $g^k_{i_l} = g^{k-1}_{i_{l-1}} + d^k_{i_l}$ we store the pairs related with $g^{k-1}_{i_{l-1}}$ and (i_l, k).

Finally $g^{i_n}_n$ contains enough information to recover the set S_j, namely the candidate set with the minimum reduced cost among those served by facility j. The associated reduced cost is

$$\overline{c}^j_{S_j} = g^n_{i_n} + \delta + \gamma_j,$$

which is the minimum value associated with any feasible set served by the considered facility j. Obviously, if this value is negative the variable $y^j_{S_j}$ is a good candidate to be chosen in the next iteration of the column generation scheme.

If we solve this problem for all j, we get $\overline{c}^j_R = \min_S \overline{c}^j_S$ and if $\overline{c}^j_R < 0$, we can activate (at least) y^j_R. Next, we solve a new reduced master problem (ReMP) with this (these) new activated variable(s).

Example 2 (Cont'd)**.** *We show the computation of the g^k_i elements derived from D_1.*

$g^1_1 = \min\{0, -22.5\} = -22.5$. *The minimum is attained by pair* $(1, 1)$.
$g^2_1 = \min\{-22.5, -15.5\} = -22.5$ *and we keep pair* $(1, 1)$.
$g^3_1 = \min\{-22.5, -25\} = -25$ *and we store pair* $(1, 3)$.
$g^1_2 = \min\{-22.5, -12\} = -22.5$ *and we keep pair* $(1, 1)$.
$g^1_3 = \min\{-22.5, 8\} = -22.5$ *and we keep pair* $(1, 1)$.

$g_2^2 = \min\{-22.5, -22.5, -22.5, -22.5 + 10\} = -22.5$ *and we keep pair* $(1, 1)$.

$g_2^3 = \min\{-22.5, -25, -22.5, -22.5 + 2\} = -25$ *and we keep pair* $(1, 3)$.

$g_3^2 = \min\{-22.5, -22.5, -22.5, -22.5 + 0\} = -22.5$ *and we keep pair* $(1, 1)$.

$g_3^3 = \min\{-22.5, -25, -22.5, -22.5 - 7\} = -29.5$ *and we keep pair* $(1, 1)$ *and store pair* $(3, 3)$.

Once, we have obtained g_3^3 *we can recover the potential set to be used, if the reduced cost is negative. This set is* S_{13} *with reduced cost* $\bar{c}_{13}^1 = g_3^3 + \delta + \gamma_1 = -29.5 + 0 + 0 = -29.5 < 0$. *Hence, we active variable* y_{13}^1.

Next, the process continues with the following facilities, i.e. $j = 2, 3$. *In this example the optimal solution can be certified after eight complete iterations of the above process. The following table shows the objective function values and the negative reduced costs per facility obtained in each iteration.*

		$\min_S c_S^j$		
	f	*j=1*	*j=2*	*j=3*
Iteration 0	*26.5*	*-29.5*	*-23.0*	*-22.0*
Iteration 1	*26.5*	*-23.5*	*-24.5*	*-22.0*
Iteration 2	*26.5*	*-42.0*	*-38.5*	*-37.0*
Iteration 3	*11.6*	*-3.3*	*-10.4*	*-8.5*
Iteration 4	*7.6*	*-1.1*	*-5.6*	*-3.4*
Iteration 5	*7.6*	*-2.7*	*-2.2*	*0.0*
Iteration 6	*7.5*	*-0.5*	*-0.5*	*0.0*
Iteration 7	*7.0*	*0.0*	*0.0*	*0.0*

3.3.2 Dealing with infeasibility

One important issue when implementing a column generation procedure to solve a linear problem is how to deal with infeasibility. This is specially crucial if the procedure is being to be used within a branch and bound scheme to solve the linear relaxation of the problem in every node. In order to handle it, we resort to the so called Farkas pricing.

According with Farkas' Lemma, a reduced master problem is infeasible if and only if its associated dual problem is unbounded. Thus, to recover feasibility in the ReMP we have to revoke the certificate of unboundedness in the dual problem what can be done by adding constraints to it. Since we are only interested in recovering feasibility in ReMP, one can proceed in the

same way that the usual pricing, but with null coefficients in the objective function of the primal. In this way, the Farkas dual problem is

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{n} \alpha_i + \sum_{k=1}^{n} \beta_k - \sum_{j=1}^{n} \gamma_j - p\delta - \sum_{k=2}^{n} n^2 \epsilon_k \\
s.t. \quad & \sum_{\substack{i=1 \\ :(i,\cdot)\in S}}^{n} \alpha_i + \sum_{\substack{k=1 \\ :(\cdot,k)\in S}}^{n} \beta_k - \gamma_j - \delta \\
& - \sum_{k=2}^{n} \sum_{\hat{\imath}=1}^{n} \sum_{\hat{\jmath}=1}^{n} \left(\sum_{\substack{(i,k)\in S \\ :C_{\hat{\imath}\hat{\jmath}} \geq C_{ij}}} \epsilon_k + \sum_{\substack{(i,k-1)\in S \\ :C_{\hat{\imath}\hat{\jmath}} \leq C_{ij}}} \epsilon_k \right) \leq 0 \quad \forall\, j, S \\
& \gamma_j \geq 0 \quad \forall\, j \\
& \delta \geq 0 \\
& \epsilon_k \geq 0 \quad \forall\, k.
\end{aligned}
$$

and we proceed to identify new variables that make the reduced master problem feasible using the dynamic programming approach replacing c_S^j by zeros.

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

3.3.3 Stabilization

In a column generation procedure, most of the columns that are presented in the solution of the corresponding linear relaxation, i.e. nonzero variables, are generated in the last iterations. An explanation for this fact is that the algorithm is dual based and the dual vector of variables is quite different from an iteration to the next. For this reason, sometimes the stabilization is a critical step in order to reduce the number of variables and iterations needed to solve each reduced master problem.

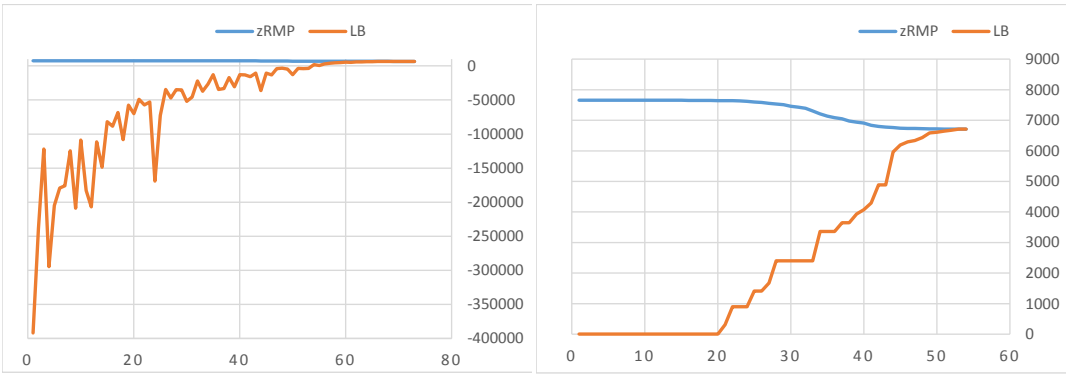
In this framework stabilization means the process of accelerating the slow convergence of dual multipliers to the optimal solutions of the dual problem (du Merle et al. [1999]). In our approach, we follow the procedure in Pessoa et al. [2010]. This paper describes a stabilization procedure which just depends of one parameter. This algorithm is based on using a vector of

dual variables which is a convex combination of the previous vector and the current solution of the dual problem.

Let $\pi = (\alpha, \beta, \gamma, \delta, \epsilon)$ be a generic vector of dual multipliers, $\overline{\pi}$ be the best known vector of dual multipliers (found so far) and π_{ReMP} be the current solution of the dual problem. Let $\overline{c}_S^j(\pi)$ be the reduced cost of y_S^j computed with the dual variable π and $LB(\pi)$ the lower bound provided by the same vector of dual multipliers, namely π. Finally, let b be the resource vector of ReMP. The stabilization algorithm that we have implemented is described by the following pseudocode:

Algorithm 1 Stabilization in $LRMP$.

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



(a) Stabilization disabled (b) Stabilization enabled

Figure 3.2: Bound's behavior at the root node in Problem n40p10v1 on successive iterations.

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

In order to show the performance of the algorithm we report in Figure 3.2 the evolution of the lower and upper bound with respect to number of iterations. Results reported here correspond to a single example, but when stabilization is applied generally results in the best behavior observed in practice. One can realize that the dual bound is not infinity at iteration 0 and that it does not improve until certain iteration. The reason is because we start with a feasible solution of the problem. See subsection 3.4.2 for more details.

The control over the dual variables significantly improves the necessary number of iterations and the number of variables used to certify optimality. Note that this improvement becomes more important on a branch-and-bound

procedure where the number of variables should be small in every node.

To decide the way to use the stabilization within our framework we solve 110 instances with sizes ranging from 10 to 30 clients. Firstly, we show -in Table 3.1- the behavior on the linear relaxation of the Master Problem MP when the stabilization is applied. This table compares the CPU time and number of variables if stabilization is or is not enabled. One can see that the number of variables decreases and only a small increment of CPU time is appreciated on these instances whenever stabilization is enabled.

	No stabilization	Stabilization		
		$\Delta_{init} = 0.2$	$\Delta_{init} = 0.4$	$\Delta_{init} = 0.6$
Time	1.15	1.21	1.19	1.18
$\|Vars\|$	849	640	635	631

Table 3.1: Average CPU-Time and number of variables to solve the linear relaxation of the tested instances.

In the actual (discrete) version of the problem we use 50 instances, out of those described above, to test three different approaches for the stabilization framework as well as several options of Δ_{init}. The difference between these three approaches lies in the intensity of the use of the stabilization procedure: not using, using at the root node or using in all the nodes of the B&B tree.

In table 3.2 we summarize the results. One can see that the best strategy is to apply stabilization at all nodes and with an initialization parameter $\Delta_{inint} = 0.2$.

	No stabilization	Stabilization(Root node)		
		$\Delta_{init} = 0.2$	$\Delta_{init} = 0.4$	$\Delta_{init} = 0.6$
Time	47.37	30.40	31.29	80.16
$\|Vars\|$	9628	7265	7841	10689
#Unsolved	2	1	1	4
		Stabilization(All nodes)		
		$\Delta_{init} = 0.2$	$\Delta_{init} = 0.4$	$\Delta_{init} = 0.6$
Time		12.33	30.07	12.76
$\|Vars\|$		4530	7171	4700
#Unsolved		0	1	0

Table 3.2: Average CPU-Time, number of variables and number of unsolved problems with different strategies of stabilization.

In this analysis, we set the CPU-time limit in 15 minutes. In order to avoid disturbances as a consequence of the unsolved instances we show in Figure 3.3 the performance profile of the three approaches. This figure represents the number of solved problems up to certain amount of time. Note that the Time axis is in a logarithmic scale. Among the options of Δ_{init}, we choose $\Delta_{init} = 0.2$ for this representation. Although for short time limit there is no clear dominance of one of the approaches over the others, once the time limit is bigger the strategy of applying stabilization in all the nodes dominates since it is able to solve to optimality more instances within the specified time limit.

3.4 Branch-and-cut-and-price

3.4.1 Preprocessing

In order to improve the performance of the algorithm we use two different preprocessing to set some variables to zero. Our preprocessing is based on Claims 1 and 2 in Chapter 2. The reader may observe that although those results fix original x_{ij}^k variables we can translate the variable fixing to the new setting by the relation $x_{ij}^k = \sum_{S \ni (i,k)} y_S^j$.

Hence, variables y_S^j such that $(i,k) \in S$ and $x_{ij}^k = 0$ will not be considered

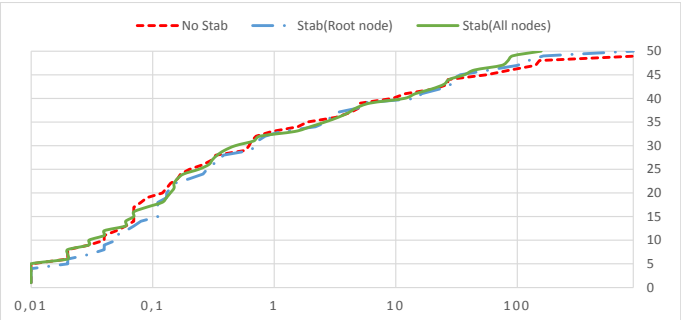


Figure 3.3: Number of solved problems per time for some options of stabilization.

to be added to the ReMP. For this purpose, we simply force that $d_{ij}^k = 0$ in every pricing subproblem.

3.4.2 Initialization

Since we are solving the linear relaxation of the ReMP without its entire set of variables, using the primal simplex algorithm, the goal of the initialization phase is to find an initial set of columns that allows solving the ReMP performing a small number of iterations in the column generation routine. To do this, we first provide an initial pool of columns which tries to make feasible this problem by extracting a set obtained from, $DOMP_4^{LP}$, the linear relaxation of $DOMP_4$.

Algorithm 2 Warm-start for MP.

```
1: Input();
2: Solve DOMP_4^LP;
3: for j : y_j > 0 do
4:    S̄ = ∅;
5:    for i = 1, ..., n do
6:       for k = 1, ..., n do
7:          if x_ij^k > 0 then
8:             S̄ = S̄ + {(i, k)};
9:          end if
10:      end for
11:   end for
12:   Let i_1 ..., i_n be a permutation which ensures c_{i_1 j} ≤ ··· ≤ c_{i_n j}
13:   while S̄ ≠ ∅ do
14:      S = ∅;
15:      k_min = 1
16:      for l = 1, ..., n do
17:         for k = k_min, ..., n do
18:            if (i_l, k) ∈ S̄ then
19:               S = S + {(i_l, k)};
20:               S̄ = S̄ − {(i_l, k)};
21:               k_min = k + 1
22:               break;
23:            end if
24:         end for
25:      end for
26:      Add y_S^j to the problem.
27:   end while
28: end for
```

Let (x, y) be the optimal solution of $DOMP_4^{LP}$. The initial set of columns is obtained by means of Algorithm 2 that chooses fractional variables which are compatible with constraints (3.6) in the relaxed solution of $DOMP_4$.

To show the convenience of this technique we present a computational study with 70 instances. In Table 3.3 it is shown how the average time to solve the instances decreases when the heuristic initialization pool of columns is applied. It might be even more significant to remark that the maximum time

to solve all the instances is rather small when this initialization is enabled. Furthermore, the overall number of variables necessary to certify optimality is lower in average. We also point out that when the initialization is enabled the number of variables added by means of the warm-start (initialization) is rather small and it represents in average only a 0.32% over the total number of variables to certify optimality.

	Disabled	**Enabled**
Time	33.9	26.6
Max Time	1003.0	400.4
#vars	5 408	5 349
#initial vars	–	17

Table 3.3: Summary of the comparison of using variables to initialize or not.

3.4.3 Upper bound for the Master Problem: A GRASP heuristic

In order to start the problem with a feasible integer solution, we present a heuristic algorithm to generate a feasible solution for MP. This feasible solution will provide a good upper bound as well.

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

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

Algorithm 3 GRASP for DOMP.

```
1: Input();
2: for n replications do
3:    ConstructGreedyRandomizedSolution (Solution);
4:    for p iterations do
5:       LocalSearch(Solution);
6:       UpdateSolution (Solution, BestSolutionFound);
7:    end for
8: end for
```

First of all we would like to point out the remarkable behavior of our GRASP heuristic in the application to this problem. In order to illustrate the appropriateness of our heuristic we have solved to optimality a number of instances of the problem (using the MIP formulations) to be compared with those given by the heuristic approach. The reader can observe the results on table 3.4. In this table we have solved 20 instances per each size and in all cases, except one instance for $n = 60$, the GRASP heuristic has reached the optimal solution of the problem. Furthermore, in the only case that the optimal solution was not obtained the GAP was 0.056%.

n	**p**	**#Instances**	**#Optimal**
20	10	20	20
30	15	20	20
40	20	20	20
45	15	20	20
50	25	20	20
55	25	20	20
60	20	20	19

Table 3.4: Number of instances for which GRASP heuristic reaches the optimal solution by size (n).

Moreover, it is not only advisable to use the GRASP heuristic because it provides a very good upper bound but it also helps in solving the problem since this solution prunes many branches of the branch-and-bound tree. According with Table 3.5 and Figure 3.4 it is clearly advisable using the bound

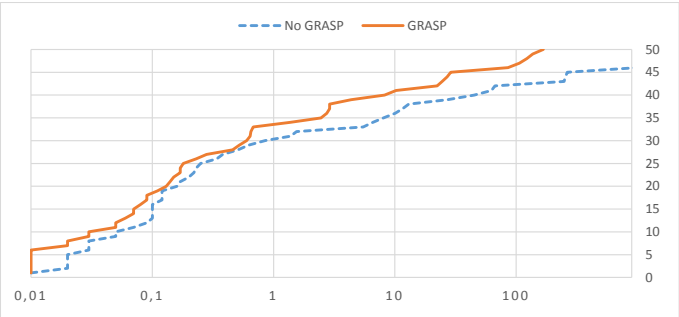


Figure 3.4: Number of solved problems per time using or not GRASP heuristic.

obtained by means of the GRASP heuristic. Its main benefit lies on the improvement of the number of nodes, i.e. on the reduction of the size of the branch-and-bound tree.

GRASP	**Time**	**#nodes**	$\|Vars\|$
Disabled	110.67(2)	106.68	13 083
Enabled	15.26	53.40	4 930

Table 3.5: CPU-Time, Number of nodes and Number of variables using or not GRASP heuristic for $n = 10, 15$.

3.4.4 Branching

The MP is defined in the y_S^j variables so that a first natural option would be to branch on them. This standard branching presents an important drawback: on the zero branch, a variable fixed to zero could be desirable in terms of reduced cost (see Barnhart et al. [1998]) but it does not fix any allocation in the underlying location problem making this branch, possibly rather unbalanced. In addition, branching on the y_S^j variables would imply to add constraints $y_S^j = 1$ or $y_S^j = 0$ to the problem in each node of the branching tree. In order to simplify the resolution of the linear relaxation of the problem in each node, we would try to keep the structure of the Master Problem transferring the new constraint to the pricing subproblem. This would be

easily doable in the branch where $y_S^j = 1$ since it could be done by simply removing some rows and columns in the corresponding matrix D_j. On the other hand, in the branch where $y_S^j = 0$ we would have to remove this variable in the pricing subproblem. Nevertheless, this process can not be done sequentially using the dynamic programming recursion (some backtracking would be necessary), which makes the resolution of the pricing very inefficient. For this reason we have decided not to branch on the y_S^j variables, rather we branch on in the original variables x_{ij}^k.

Branching on original variables

In our algorithm, we have implemented branching on the original x_{ij}^k variables. Branching on original variables is a common option on Mixed Integer Master Problems where some set partition constraints are involved. See for instance Johnson [1989]. Recall that $x_{ij}^k = \sum_{S \ni (i,k)} y_S^j$, thus, a way to branch on the fractional solution can be derived directly from satisfying integrality conditions of original variables.

Proposition 7. *If $x_{ij}^k \in \{0,1\}$ for $i,j,k = 1,\ldots,n$, then $y_S^j \in \{0,1\}$.*

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

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

$$1 \geq \sum_{S \ni (i_1,k_1)} y_S^{j'} > \sum_{S \ni (i_2,k_2)} y_S^{j'} > 0.$$

The first inequality comes directly from the formulation. The second inequality is strict because the term $\sum_{S \ni (i_2,k_2)} y_S^{j'}$ has at least one fractional variable less than the the term $\sum_{S \ni (i_1,k_1)} y_S^{j'}$. Te third inequality is strict because of the election of (i_2,k_2). Finally, a contradiction is found because $x_{i_2k_2}^{j'}$ is not binary. □

The reader may note that this branching can be seem as a SOS1 branching since at most one of the above y_S^j variables can assume the value 1.

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

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

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

- $x_{ij}^k = 0 \Rightarrow d_{ij}^k = 0$.
- $x_{ij}^k = 1 \Rightarrow \begin{cases} d_{i\hat{\jmath}}^k = 0, & \forall \hat{\jmath} \neq j. \\ d_{\hat{\imath}\hat{\jmath}}^k = 0, & \forall \hat{\jmath} \forall \hat{\imath} \neq i. \\ d_{i\hat{\jmath}}^{\hat{k}} = 0, & \forall \hat{\jmath} \forall \hat{k} \neq k. \end{cases}$

Note that when we fix y_S^j variables to one the resulting subproblem is easier because we have fixed to zero a larger number of variables. Moreover, it is also well-known that branching on SOS constraints (original variables) gives rise to more balanced branching trees (see Chapter 7 of Wolsey [1998]) than doing it in the variables of the problem.

Selecting a variable to branch on

Among the fractional original variables one has to decide which variable to branch on. One of the easiest techniques for this choice is the *most fractional variable*. This is not difficult to implement but it is not better than choosing randomly (Achterberg et al. [2005]).

Alternative techniques are *pseudocost branching* (Benichou et al. [1971]) or *strong branching* (Applegate et al. [1995]). They are based on the idea of reducing the tree size by means of increasing the lower bound of the newly created nodes, as much as possible. Strong branching has a high computational cost. This drawback is even more pronounced on a column generation approach. Pseudocost branching has not the same computational cost but on the other hand, the initialization of the pseudocosts it is not a trivial task in our case.

These drawbacks have motivated that here we propose another rule to select the variable to branch on, based on the improvement of the bounds in

each of the new created nodes. We use the following indices corresponding to the down and up branches of the variable x_{ij}^k:

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

They account, respectively, for the unitary contribution to the objective function due to fixing the variable x_{ij}^k either to zero (down branching) or to one (up branching). Branching down clearly stimulates the improvement of the lower bound, whereas branching up helps the problem to find good integer solutions. Once the measures are defined we can define several strategies to determine the variable to be chosen.

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

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

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

The reader can see that on the one hand, using Strategy 1 and choosing $\theta = 1$, one expects to increase the lower bound; with $\theta = 0.5$, we are essentially reproducing a scaled version of the most fractional branching; and $\theta = 0$ focuses on getting integer solutions. On the second hand, Strategy 2 puts the strength in the improvement of the lower bounds. Finally, Strategy 3 tries that none of new branches are bad in their bounds.

We compare the different strategies described above to decide the next variable to branch. This analysis is supported by 50 instances. Table 3.6 shows that Strategy 1 ($\theta = 1$) is the most promising branching rule. Since we have a good initial upper bound because of the GRASP heuristic (Subsection 3.4.3), that strategy helps us to improve the lower bound which is the bottleneck in our implementation.

Branching strategy	**Time**	**#nodes**	$\|Vars\|$
Strategy 1($\theta = 0$)	250.41(13)	106.94	31 177
Strategy 1($\theta = 0.1$)	190.22(10)	47.36	28 427
Strategy 1($\theta = 0.5$)	127.22(6)	44.46	21 259
Strategy 1($\theta = 0.9$)	20.26	52.92	5 084
Strategy 1($\theta = 1$)	15.26	53.40	4 930
Strategy 2	171.58(8)	65.00	24 846
Strategy 3	102.26(5)	49.18	16 524

Table 3.6: CPU-Time, Number of nodes and Number of variables of the different branching strategies for $n = 10, 15$.

3.4.5 Valid inequalities

The formulation $DOMP_4$ can be reinforced by adding some families of valid inequalities that translate also to the set partition reformulation MP. The final goal is to use them within the Branch & Price algorithm to improve the performance.

Observe that constraints (3.6) are the aggregation over i, j of inequalities of the form

$$\sum_{i'=1}^{n} \sum_{\substack{j'=1: \\ i'j' \preceq ij}}^{n} x_{i'j'}^{k} + \sum_{i'=1}^{n} \sum_{\substack{j'=1: \\ i'j' \succeq ij}}^{n} x_{i'j'}^{k-1} \leq 1,\ i, j = 1, \cdots, n,\ k = 2, \cdots, n \qquad (3.20)$$

These inequalities are the so called *order constraints* which are known to be valid for DOMP. One can also express these inequalities in terms of the y_S variables so that they can be used in the set partition formulation of DOMP. Indeed, the translation of (3.20) results in:

$$\sum_{\substack{S \ni (\hat{\imath},k) \\ :C_{\hat{\imath}\hat{\jmath}} \leq C_{ij}}} y_S^{\hat{}} + \sum_{\substack{S \ni (\hat{\imath},k-1) \\ :C_{\hat{\imath}\hat{\jmath}} \geq C_{ij}}} y_S^{\hat{}} \leq 1,\ i, j = 1, \ldots, n, k = 2, \ldots, n. \qquad (3.21)$$

Clearly, the addition of valid inequalities (3.21) to MP modifies the structure of the master problem and thus the pricing must be modified accordingly. Let us denote by ζ_{ij}^{k} the dual variable associated with valid inequality (3.21) for indices i, j, k. After some calculation, one can work out the new

expression of the reduced costs of variable y_S^j which results in:

$$\bar{c}_S^j = c_S^j + \gamma_j^* + \delta^* + \sum_{k=2}^{n}\sum_{\hat{\imath}=1}^{n}\sum_{\hat{\jmath}=1}^{n}\left(\sum_{\substack{(i,k)\in S\\ :C_{\hat{\imath}\hat{\jmath}}\geq C_{ij}}}(\epsilon_k^* + \zeta_{\hat{\imath}\hat{\jmath}}^{k*}) + \sum_{\substack{(i,k-1)\in S\\ :C_{\hat{\imath}\hat{\jmath}}\leq C_{ij}}}(\epsilon_k^* + \zeta_{\hat{\imath}\hat{\jmath}}^{k*})\right) - \sum_{\substack{i=1\\ :(i,\cdot)\in S}}^{n}\alpha_i^* - \sum_{\substack{k=1\\ :(\cdot,k)\in S}}^{n}\beta_k^*.$$

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

$$d_{ij}^k = \begin{cases} \lambda^k C_{ij} + \sum_{\hat{\imath}=1}^{n}\sum_{\substack{\hat{\jmath}=1\\ :C_{\hat{\imath}\hat{\jmath}}\leq C_{ij}}}(\epsilon_{k+1} + \zeta_{\hat{\imath}\hat{\jmath}}^{k+1}) - \alpha_i - \beta_k & \text{if } k = 1 \\ \lambda^k C_{ij} + \sum_{\hat{\imath}=1}^{n}\sum_{\substack{\hat{\jmath}=1\\ :C_{\hat{\imath}\hat{\jmath}}\geq C_{ij}}}^{n}(\epsilon_k + \zeta_{\hat{\imath}\hat{\jmath}}^{k}) + \sum_{\hat{\imath}=1}^{n}\sum_{\substack{\hat{\jmath}=1\\ :C_{\hat{\imath}\hat{\jmath}}\leq C_{ij}}}^{n}(\epsilon_{k+1} + \zeta_{\hat{\imath}\hat{\jmath}}^{k+1}) - \alpha_i - \beta_k & \text{if } k = 2,\ldots,n-1 \\ \lambda^k C_{ij} + \sum_{\hat{\imath}=1}^{n}\sum_{\substack{\hat{\jmath}=1\\ :C_{\hat{\imath}\hat{\jmath}}\geq C_{ij}}}^{n}(\epsilon_k + \zeta_{\hat{\imath}\hat{\jmath}}^{k}) - \alpha_i - \beta_k, & \text{if } k = n. \end{cases}$$

These new elements allow us to apply the adapted column generation algorithm to solve the linear relaxation of MP, reinforced with valid inequalities (3.21).

The resulting methodology can be used within the following Branch & Cut & Price algorithm: 1) Apply Branch & Bound with the branching strategy described in Section 3.4.4 on the MP formulation strengthen with some valid inequalities of (3.21) and 2) in each node of the branching tree solve the linear relaxation using column generation algorithm, then add cuts from the family (3.21) and reoptimize applying again the column generation algorithm. This solution scheme has been implemented and tested in the next section.

3.5 Computational results

To develop the branch-and-cut-and-price procedure described on this chapter we use SCIP (Achterberg [2009]). The experiments have been carried out on a PC with a 3Ghz-processor i7 and 32Gb of RAM. The models were solved using SCIP version 3.0.2 and Soplex 1.7.2 as LP solver.

Once we have justified in previous sections some parameters of the implementation (GRASP heuristic, warm-start, branching strategy or stabilization) in this section we report the computational study of formulation MP.

First we report on the best strategy to use the cuts showed in Section 3.4.5. Next we present results that compare formulation $DOMP_4$ and its column generation version, namely formulation MP.

3.5.1 Cuts

The cuts described in Section 3.4.5 are implemented using an efficient separation procedure (see Section 2.4.3) and they have been also included in the column generation routine. We test here two main strategies to take advantage of this family of cuts: adding cuts only at the root node and adding cuts in bigger depth. In Table 3.7 one can see how the number of necessary nodes decreases significantly. Over 90 instances using cuts, we are able to solve 18 additional problems when cuts are applied. According with Figure 3.5 adding cuts is very advisable, but it is not clear if we have to continue adding cuts after the root node is solved. Our decision in this regard was influenced by two main aspects that affect to the application to bigger size instances.

1. In our computational experiment, 75% of the finally considered cuts has been added on the root node.

2. Since we have $O(n^3)$ cuts this number could become intractable for bigger size problems.

Hence, our strategy for the next subsection will be to add cuts only at the root node and to keep them (but not adding more) in the later linear relaxations generated by the branching.

Cuts	Disabled	Root node	All nodes
Time	417.86	257.13	274.77
$\|Vars\|$	43491	6126	6513
$\|Nodes\|$	415.5	26	20
$\|Cuts\|$	–	230	308
#Unsolved	41	23	24

Table 3.7: Average CPU-Time, number of variables and number of problems not solved with different strategies of separation algorithms.

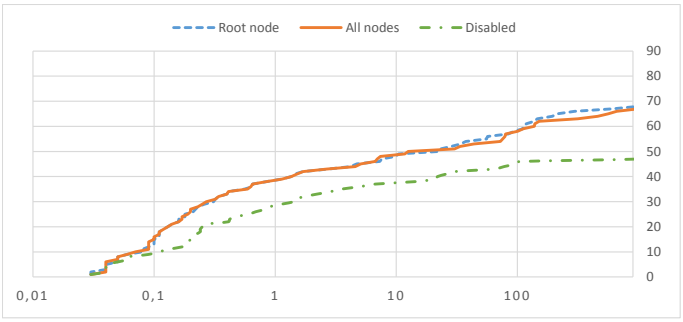


Figure 3.5: Number of solved problems per time using different cut strategies.

3.5.2 Comparing with $DOMP_4(B\&C-3)$

In this section we compare the branch-and-cut implementations of formulations $DOMP_4(B\&C-3)$ and MP, both with the family of valid inequalities described in Section 3.4.5. Separation is made using our efficient oracle presented in Chapter 2 and cuts are only added in the root node.

	n=20,p=10		n=30,p=15		n=40,p=20	
	$DOMP_4$	MP	$DOMP_4$	MP	$DOMP_4$	MP
GAP (%)	0.87	0.82	1.26	1.20	0.66	0.65
Time	0.59	11.48	4.41	23.16	32.46	194.92
Max Time	2.39	131.02	412.29	213.35	75.77	775.38
$\|Vars\|$	3475	1741	12850	2448	30469	6684

Table 3.8: Average GAP, CPU-Time and number of variables for solving to optimality.

Table 3.8 reports the average results over 20 instances per each combination of clients (n) and open facilities (p) and distinguishing between formulation $DOMP_4$ and MP. Row *GAP (%)* gives the gap of the linear relaxations of both formulations before the valid inequalities are included in the root node. Row *Time* shows the CPU time to solve the problems to optimality, whereas row *Max Time* reports the maximum among the resolution times needed to solve the 20 instances. We also report, in row $|Vars|$ the number of variables needed in each formulation. As expected, according to

Proposition 6, the integrality gap of formulation MP outperforms the one by $DOMP_4$.

Analyzing further the results in this table, we conclude that on average our implementation of MP is slower than the one in Soplex for $DOMP_4$. We could explain this behavior by several reasons. On the one hand, we have to solve each linear relaxation by the column generation algorithm which is costly because the pricing uses a dynamic programming algorithm which is not linear. On the other hand, we are using an implementation that may not be professional as compared with the one by Soplex in solving each linear relaxation. Of course, here there is some room for improvement. We could use some faster heuristic algorithm for solving the pricing subproblems so that each intermediate iteration is done faster and we only apply the exact algorithm to certify optimality. In addition, we can also optimize our code with the help of an expert computer scientist that improves that internal plugins and the data handling, among other aspects in our implementation. However, although CPU time is important, in this problem it is even more crucial to control its size since the number of variables in the original formulation $DOMP_4$ is of order of $O(n^3)$. This number can be observed in the table even after the application of the preprocessing phase. One important feature of our MP formulation is that it needs a much smaller number of variables that $DOMP_4$. This allows solving larger size instances with MP that could not be affordable for the original $DOMP_4$.

Chapter 4

The monotone ordered median problem

4.1 Introduction

Previous chapters have been devoted to the analysis of the general version of the Ordered Median Problem (OMP). The goal of this chapter is to analyze a particular family of OMP, namely the so called, *Monotone Ordered Median Problem* (MOMP). This class of problems appears whenever it is imposed that the lambda coefficients satisfy the monotonic, non-decreasing property $0 \leq \lambda^1 \leq \cdots \leq \lambda^n$. On the one hand, it is clear that this constraint reduces the number of problems that can be cast within the framework. In spite of this, even with this restriction, it is still possible to formulate most of the well-known classical problems in location theory such as p-centdian, p-center, p-k-centrum, p-median, etc. On the other hand, it will be evident along the chapter that under this hypothesis the problem gains some extra structure so that new properties can be proven and more efficient formulations can be derived allowing the resolution of larger problem sizes.

This chapter is devoted to the study of Monotone Discrete Ordered Median Problems (MDOMP) and it is composed of four sections. Section 4.2 deals with the first known model that exploited the special monotonic structure of this problem: the so called Ogryczak-Tamir model (OT), based on an efficient representation of k-sums (Ogryczack and Tamir [2003], Nickel and Puerto [2005]). Section 4.3 analyzes another model for the MDOMP that comes out from a different rationale. It appears for the first time ap-

plied to location problems in the paper by Blanco et al. [2014], and by that reason we will refer to it as the Blanco-El Haj-Puerto model (BHP). We compare, in Section 4.4, these two formulations between them and also with all the formulations previously analyzed in Chapter 2. We present next, in Section 4.5, some computational results comparing empirically the proposed formulations.

4.2 The model by Ogryczak-Tamir (OT)

In a remarkable paper by Ogryczack and Tamir [2003] these authors introduce a novel linear time algorithms to compute the sum of the q-largest entries ($q \leq n$) of an arbitrary vector of n components. This idea was later exploited by Kalcsics et al. [2002] to develop an efficient representation for the q-centrum location problem and more generally extended to deal with the MDOMP.

Assume that we are given a non-negative n vector $d = (d_1, \ldots, d_n)$. Following the idea in Ogryczack and Tamir [2003] and Kalcsics et al. [2002], the reader can check that the following problem returns as its optimal value $S_q(d) = \sum_{k=n-q+1}^{n} d_{(k)}$, the sum of the q-largest values out of the n components of the given vector d.

$$\begin{aligned} S_q(d) = \quad & \min \quad qt + \sum_{i=1}^{n} z_i \\ & \text{s.t.} \quad d_i - t \leq z_i \; i = 1, \ldots, n \\ & \qquad z_i \geq 0, \; \forall \, i, k. \end{aligned} \tag{4.1}$$

Following the proof in Ogryczack and Tamir [2003] one can prove that, under the assumption of nonnegative d values, the free variable t can be further strengthen to be nonnegative.

Now, to represent the ordered median value of d for the lambda weights $0 \leq \lambda^1 \leq \ldots \lambda^n$, namely $\sum_{k=1}^{n} \lambda^k d_{(k)}$, we can resort to the following observation whose proof is direct and it is left to the reader.

Lemma 1.

$$\sum_{k=1}^{n} \lambda^k d_{(k)} = \sum_{k=1}^{n} (\lambda^k - \lambda^{k-1}) S_k(d), \tag{4.2}$$

where for convenience $\lambda^0 = 0$.

Based on this idea we can formulate the discrete ordered median problem for nondecreasing lambda weights by means of a the following MILP. We need to apply the above representation to the assignment costs induced by the points in the location polytope, namely

$$X = \left\{ \sum_{j=1}^{n} y_j = p,\ \sum_{j=1}^{n} x_{ij} = 1,\ \forall i,\ x_{ij} \leq y_j\ \forall i, j,\ y_j \in \{0,1\} \forall j,\ x_{ij} \geq 0\ \forall i, j \right\}.$$

It is straightforward to realize that for a given feasible point (x, y), its assignment costs are $Cx := (\sum_{j=1}^{n} C_{1j}x_{1j}, \ldots, \sum_{j=1}^{n} C_{nj}x_{nj})$. Hence, using (4.2) replacing $S_k(Cx)$ by its valid formulation (4.1), it results in the following formulation for the MDOMP.

$$(\mathbf{MDOMP_{OT_0}}) \min \sum_{k=1}^{n} (\lambda^k - \lambda^{k-1}) \left[(n-k+1)t_k + \sum_{i=1}^{n} z_{ik} \right] \tag{4.3}$$

$$\text{s.t.} \sum_{j=1}^{n} y_j = p \tag{4.4}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall\, i \tag{4.5}$$

$$x_{ij} \leq y_j \qquad \forall\, i, j \tag{4.6}$$

$$\sum_{j=1}^{n} C_{ij}x_{ij} - t_k \leq z_{ik} \qquad \forall\, i, k \tag{4.7}$$

$$x_{ij}, y_j \in \{0,1\} \qquad \forall\, i, j \tag{4.8}$$

$$z_{ik} \geq 0 \qquad \forall\, i, k \tag{4.9}$$

Once again, under the assumption of nonnegative C_{ij} cost coefficients, we can assume that $t_k \geq 0$, for all $k = 1, \ldots, n$.

The above formulation can be strengthened taking advantage of the ties in the consecutive values of the lambda vector. Let us assume that there are Q different "blocks", each one with q_i replications, $i = 1, \ldots, Q$:

$$\lambda = (\underbrace{\lambda^1, \ldots, \lambda^1}_{q_1}, \underbrace{\lambda^2, \ldots, \lambda^2}_{q_2}, \cdots, \underbrace{\lambda^Q, \ldots, \lambda^Q}_{q_Q})$$

The following is also a valid formulation for the MDOMP:

$$
\begin{aligned}
(\mathbf{MDOMP_{OT}}) \quad \min \quad & \sum_{k=1}^{Q}(\lambda^k - \lambda^{k-1}) \left[\left(\sum_{k'=k}^{Q} q_{k'} \right) t_k + \sum_{i=1}^{n} z_{ik} \right] \\
\text{s.t.} \quad & \sum_{j=1}^{n} y_j = p \\
& \sum_{j=1} x_{ij} = 1 \quad \forall\, i \\
& x_{ij} \leq y_j \quad \forall\, i, j \\
& \sum_{j=1} C_{ij} x_{ij} - t_k \leq z_{ik} \quad \forall\, i, k \\
& x_{ij}, y_j \in \{0, 1\} \quad \forall\, i, j \\
& z_{ik} \geq 0 \quad \forall\, i, k.
\end{aligned}
$$

Remark 1. *Observe that for some particular problems where the first values of the λ-vector are zeros, e.g. p-center problem, one can adapt the formulation $MDOMP_{OT}$ taking*

$$\lambda = (0, \ldots, 0, \underbrace{\lambda^1, \ldots, \lambda^1}_{q_1}, \underbrace{\lambda^2, \ldots, \lambda^2}_{q_2}, \cdots, \underbrace{\lambda^Q, \ldots, \lambda^Q}_{q_Q})$$

with minor modifications.

4.2.1 Further extensions: New formulations for DOMP

The extraordinary performance that is obtained by the above formulations in the *convex* cases, leads us to extend the rationale behind the k-sum representations to be used in a more general framework where monotonicity of the lambda vector is lost.

Assume that the costs $C_{ij} \geq 0$ for all i, j, $\lambda^k \geq 0$ for all k, $\lambda^0 = 0$, $\Delta_k = \lambda^k - \lambda^{k-1}$, for $k = 1, \ldots, n$ and $S_k(x) = \sum_{j=k}^{n} (Cx)_{(j)} : (Cx)_{(1)} \leq \ldots \leq (Cx)_{(n)}$, $k = 1, \ldots, n$. Based on the previous Lemma 1 we have the validity of the following equation:

$$\min_{x \in X} \sum_{k=1}^{n} \lambda^k (Cx)_{(k)} = \min_{x \in X} \sum_{k=1}^{n} (\lambda^k - \lambda^{k-1}) S_k(x). \tag{4.10}$$

This problem in the r.h.s can be rewritten as:

$$\min_{x\in X} \sum_{k:\Delta_k>0} \Delta_k S_k(x) + \sum_{k:\Delta_k<0} \Delta_k S_k(x).$$

Therefore, using again the transformation for the k-sum terms with positive coefficient, this problem can be formulated as:

$$\min \quad \sum_{k:\Delta_k>0} \Delta_k \Big(k t_k + \sum_{i=1}^{n} z_{ik} \Big) + \sum_{k:\Delta_k<0} \Delta_k S_k(x) \tag{4.11}$$

$$s.t. \quad z_{ik} \geq \sum_{j=1}^{n} C_{ij} x_{ij} - t_k, \quad i,k = 1,\ldots,n : \Delta_k > 0 \tag{4.12}$$

$$z_{ik},\ t_k \geq 0, \quad i,k = 1,\ldots,n : \Delta_k > 0$$

$$(x,y) \in X = \Big\{ \textstyle\sum_{j=1}^{n} y_j = p,\ \sum_{j=1}^{n} x_{ij} = 1,\ \forall i,$$

$$x_{ij} \leq y_j\ \forall i,j,\ y_j \in \{0,1\} \forall j,\ x_{ij} \geq 0\ \forall i,j \Big\}.$$

To proceed further we need a representation of the problem that minimizes the negation of the sum of the k-largest $\sum_{j=1}^{n} C_{ij} x_{ij}$ cost coefficients. Ob-

serving that we assume non-negative C_{ij}, it results in

$$\begin{aligned}
\min_{(x,y)\in X} -S_k(x) = \min_{(x,y)\in X} \quad & \min \quad \sum_{i=1}^{n} - \left(\sum_{j=1}^{n} C_{ij} x_{ij} \right) u_i \\
& \text{s.t.} \quad \sum_{i=1}^{k} u_i \leq n-k+1, \\
& \qquad 0 \leq u_i \leq 1, \ \forall i = 1, \ldots, n, \\
= \min_{(x,y)\in X} \quad & \max \quad \Big((n-k+1)t_k + \sum_{i=1}^{n} z_{ik} \Big) \\
& \text{s.t.} \quad t_k + z_{ik} \leq - \sum_{j=1}^{n} C_{ij} x_{ij}, \ \forall i = 1, \ldots, n, \\
& \qquad t_k, z_{ik} \leq 0, \ \forall i = 1, \ldots, n, \\
= \min_{(x,y)\in X} \quad & \max \quad \Big(-(n-k+1)t_k - \sum_{i=1}^{n} z_{ik} \Big) \\
& \text{s.t.} \quad t_k + z_{ik} \geq \sum_{j=1}^{n} C_{ij} x_{ij}, \ \forall i = 1, \ldots, n, \\
& \qquad t_k, z_{ik} \geq 0, \ \forall i = 1, \ldots, n.
\end{aligned}$$

Nevertheless, we cannot exploit the above formulation to be included as a linear program into (4.11) since the objective function is the maximum of negative terms. To avoid this inconvenience we need an alternative expression that can be achieved, for instance, using a representation for the k-centrum, $S_k(x)$, via the u_{kh} variables that we apply in $DOMP_2$. Recall that using $DOMP_2$,

$$\begin{aligned}
S_k(x) = \quad & \min \quad \sum_{h=1}^{G-1} (u_{kh} - u_{kh+1}) c_{(h)} \qquad (4.13) \\
& \text{s.t.} \quad u_{k-1,h} \leq u_{k,h}, \quad k = 2, \ldots, n, \ h = 1, \ldots, G \\
& \qquad \sum_{k=1}^{n} (1 - u_{k,h}) = \sum_{\substack{i,j=1 \\ C_{ij} < c_{(h)}}}^{n} x_{ij}, \quad h = 1, \ldots, G \\
& \qquad z_{ik}, \ t_k \geq 0, \quad i, k = 1, \ldots, n, \\
& \qquad u_{kh} \in \{0, 1\}, \quad k = 1, \ldots, n, \ h = 1, \ldots, G.
\end{aligned}$$

However, if the objective function appears with a negative coefficient, it is not true anymore that the x variables will take the assignment given by the smallest possible costs. This must be enforced by the formulation adding some new constraints. In this case, we can use the following set of inequalities that enforce closest assignment of customers to open facilities.

$$x_{ij} \leq (1 - y_{\hat{\jmath}}), \quad \forall\, i, j, \hat{\jmath} = 1, \ldots, n,\, C_{ij} \geq C_{i\hat{\jmath}}. \tag{4.14}$$

Next, let $\alpha = \min\{k : \Delta_k < 0\}$. The DOMP with general lambda can be written as:

$$\begin{aligned}
(\mathbf{DOMP_{OTG}}) \min_{(x,y)\in X} \quad & \Big\{ \sum_{k:\Delta_k>0} \Delta_k((n-k+1)t_k + \sum_{i=1}^{n} z_{ik}) \\
& + \sum_{k:\Delta_k<0} (-\Delta_k) \sum_{h=1}^{G-1} (u_{kh} - u_{kh+1}) c_{(h)} \Big\} && (4.15)\\
\text{s.t.} \quad & t_k + z_{ik} \geq \textstyle\sum_{j=1}^{n} c_{ij} x_{ij}, \quad i, k = 1, \ldots, n,\, \Delta_k > 0 && (4.16)\\
& x_{ij} \leq (1 - y_{\hat{\jmath}}), \quad i, j, \hat{\jmath} = 1, \ldots, n,\, C_{ij} \geq C_{i\hat{\jmath}} && (4.17)\\
& u_{k-1,h} \leq u_{k,h}, \quad k = 2, \ldots, n,\; h = 1, \ldots, G && (4.18)\\
& \alpha - 1 + \textstyle\sum_{k=\alpha}^{n} (1 - u_{k,h}) \geq \displaystyle\sum_{\substack{i,\, j\, =\, 1 \\ c_{ij} < c_{(h)}}}^{n} x_{ij}, \quad h = 1, \ldots, G && (4.19)\\
& z_{ik},\; t_k \geq 0, \quad i, k = 1, \ldots, n,\, \Delta_k > 0 && (4.20)\\
& u_{kh} \in \{0, 1\}, \quad k \geq \alpha,\; h = 1, \ldots, G. && (4.21)
\end{aligned}$$

This is a formulation that gives good results whenever the first non monotonic occurrence of a lambda value occurs close to the n-th entry, i.e. $n - \alpha$ is small. To analyze that we have generated some random lambda vectors for which α is a percentage of the number of clients. Hence, we force that the $\Delta_k \geq 0$ if $k < \alpha$ and for the remaining ones we let them freedom to be positive or negative coefficients. In Table 4.1, we compare this new general formulation $DOMP_{OTG}$ with $DOMP_2$. In this table we show the average CPU-time and the average integrality GAP obtained from 900 instances varying the number of clients, the number of available servers to be open, the cost matrix and the randomly generated ordered weighted vectors. These λ-vectors has been randomly generated ensuring that the first α components are monotone. The reader can observe that the new formulation $DOMP_{OTG}$

outperforms $DOMP_2$ for three out of the four values of α tested in the analysis. Our conclusion is that for λ-vectors with a high percentage of monotone entries it may be advisable to use $DOMP_{OTG}$ since its CPU times to optimality are rather small. This behavior is observed, at least, up to 50 % of monotone components in the lambda vector. Appendix B.1 reports detailed results clustered by n and p.

	DOMP$_{\mathbf{OTG}}$		**DOMP$_{\mathbf{2}}$**	
	Time	GAP	Time	GAP
$\alpha \geq \lfloor 0.9n \rfloor$	7.61	2.17	79.24	13.29
$\alpha \geq \lfloor 0.7n \rfloor$	12.27	2.30	59.84	10.77
$\alpha \geq \lfloor 0.5n \rfloor$	16.84	2.82	37.41	7.47
$\alpha \geq \lfloor 0.3n \rfloor$	62.98	28.19	20.04	5.22

Table 4.1: CPU-Time and integrality GAP for $n = 10, 20, 30$.

4.3 The Blanco-El Haj-Puerto model (BHP)

There exists another specific formulation for the monotone case of the DOMP which is based on a different rationale from the one used to derive $DOMP_{OT}$. This formulation was first presented within the scope of location analysis in the paper by Blanco et al. [2014]. The formulation is based on two lemmas that exploit the non-negativity and monotonicity of the λ-vector used in the objective function. We reproduce them for the sake of completeness.

Lemma 2. *Let $x \in \mathbb{R}^n$ and $\lambda \geq 0$ then $\sum_{k=1}^{n} \lambda^k x_{(k)}$ is a monotonically non-decreasing function of x.*

Lemma 3. *If $0 \leq \lambda^1 \leq \cdots \leq \lambda^n$ then $\sum_{k=1}^{n} \lambda^k x_{(k)} = \max_{\sigma \in \mathcal{P}(n)} \sum_{i=1}^{n} \lambda^k x_{\sigma(i)}$.*

The proof of both results can be found, for instance, in Nickel and Puerto [2005]. The reader may observe that using Lemma 3 one can write down the evaluation of the function $\sum_{k=1}^{n} \lambda^k x_{(k)}$ by means of the optimal value of an

integer linear problem. Indeed,

$$\begin{array}{lllll} \sum_{k=1}^{n} \lambda^k x_{(k)} = & \max & \sum_{k=1}^{n}\sum_{i=1}^{n} \lambda^k p_{ik} x_i & & \textbf{Dual Multipliers} \\ & s.t. & \sum_{i=1}^{n} p_{ik} & = 1 \quad \forall\, k & u_k \text{ no restricted} \\ & & \sum_{k=1}^{n} p_{ik} & = 1 \quad \forall\, i & v_i \text{ no restricted} \\ & & p_{ik} & \in \{0,1\} \quad \forall\, i,k & \end{array}$$

We observe that the constraints of the problem above are those that model permutations, i.e. assignment constraints. Thus, variables p_{ik} are enforced to be binary by total unimodularity and could be relaxed to be non negative. This fact allows to compute the dual and its value will also be a valid form of evaluation for $\sum_{k=1}^{n} \lambda^k x_{(k)}$. Hence,

$$\begin{array}{lllll} \sum_{k=1}^{n} \lambda^k x_{(k)} & = \min & \sum_{k=1}^{n} u_k + \sum_{i=1}^{n} v_i & & \\ & \text{s.t.} & u_k + v_i & \geq \lambda^k x_i \quad \forall\, i,k. & \end{array}$$

Remark 2. *Under nonnegativity conditions of the lambda vector, assumed on this work, the assignment problem could be equivalently rewritten as follows:*

$$\begin{array}{lllll} \max & \sum_{\substack{k=1 \\ :\lambda^k>0}}^{n}\sum_{i=1}^{n} \lambda^k p_{ik} x_i & & & \textbf{\textit{Dual Multipliers}} \\ s.t. & \sum_{i=1}^{n} p_{ik} & \leq 1 \quad \forall\, k & u_k \geq 0 & \\ & \sum_{k=1}^{n} p_{ik} & \leq 1 \quad \forall\, i & v_i \geq 0 & \\ & p_{ik} & \geq 0 \quad \forall\, i,k & & \end{array}$$

The implications are that the representation of $\sum_{k=1}^{n} \lambda^k x_{(k)}$ is modeled by

a simplified problem

$$\begin{aligned}
\sum_{k=1}^{n} \lambda^k x_{(k)} \quad = \min \quad & \sum_{\substack{k=1 \\ \lambda^k \neq 0}}^{n} u_k + \sum_{i=1}^{n} v_i & & \\
\text{s.t.} \quad & u_k + v_i & \geq \lambda^k x_i \quad & \forall\, i,k : \lambda^k > 0, \\
& u_k, v_i \geq 0 & & \forall\, i,k : \lambda^k > 0.
\end{aligned}$$

Our goal is now to formulate the DOMP based on the above representation for an ordered weighted average. Recall that we are interested in combinatorial objects defined by the points (x, y) belonging to the location polytope, namely $(x, y) \in X = \{\sum_{j=1}^{n} y_j = p,\ \sum_{j=1}^{n} x_{ij} = 1,\ \forall i,\ x_{ij} \leq y_j\ \forall i, j\}$. Therefore, to model the DOMP we can apply the representation above to the assignment costs cx induced by the points in that polytope. This results in

$$\begin{aligned}
(\mathbf{MDOMP_{BHP_0}}) \quad \min \quad & \sum_{k=1}^{n} u_k + \sum_{i=1}^{n} v_i & & \\
\text{s.t.} \quad & \sum_{j=1}^{n} y_j & = p & \\
& \sum_{j=1}^{n} x_{ij} & = 1 \quad & \forall\, i \\
& x_{ij} & \leq y_j \quad & \forall\, i,j \\
& u_k + v_i & \geq \lambda^k \sum_{j=1}^{n} C_{ij} x_{ij} \quad & \forall\, i,k \\
& x_{ij}, y_j & \in \{0,1\} \quad & \forall\, i,j
\end{aligned}$$

Remark 3. *The way used to calculate the assignment cost associated to client i induced by the location polytope allows to relax the integrity conditions for variables x_{ij}.*

Hence, in the rest of this section we use the following formulation for the

MDOMP.

$$
\begin{array}{lllll}
(\mathbf{MDOMP_{BHP}}) & \min & \displaystyle\sum_{k=1}^{n} u_k + \sum_{i=1}^{n} v_i & & \\
& \text{s.t.} & \displaystyle\sum_{j=1}^{n} y_j & = p & \\
& & \displaystyle\sum_{j=1}^{n} x_{ij} & = 1 & \forall\, i \\
& & x_{ij} & \leq y_j & \forall\, i,j \\
& & u_k + v_i & \geq \lambda^k \displaystyle\sum_{j=1}^{n} C_{ij} x_{ij} & \forall\, i,k : \lambda_k > 0 \\
& & y_j & \in \{0,1\} & \forall\, i,j \\
& & x_{ij}, u_k, v_i & \geq 0 & \forall\, i,j,k
\end{array}
$$

4.4 Theoretical results

This section compares the different formulations from a theoretical point of view. Our goal is to compare the strength of the linear relaxations of the different MIP valid formulations for the MDOMP. For the sake of readability, let us denote by z_l^{LP} the optimal value of the LP-relaxation of $DOMP_l$.

We begin by establishing the relation between the two particular specific for the MDOMP previously developed in sections 4.2 and 4.3.

Theorem 7. $z_{OT}^{LP} = z_{BHP}^{LP}$

Proof. Let us assume a vector λ that satisfies the monotonicity assumption. We observe that for a given feasible point $(x, y) \in X = \{\sum_{j=1}^{n} y_j = p,\ \sum_{j=1}^{n} x_{ij} = 1,\ \forall i,\ x_{ij} \leq y_j\ \forall i,j,\ x_{ij} \geq 0\ \forall i,j,\ 0 \leq y_j \leq 1\ \forall j\}$ the evaluation of the ordered median function for the assignment cost, cx, is equivalently given as:

$$
\begin{array}{lllll}
cx = & \min & \displaystyle\sum_{k=1}^{n} u_k + \sum_{i=1}^{n} v_i & = \min & \displaystyle\sum_{k=1}^{n} (\lambda^k - \lambda^{k-1}) \left[(n-k+1) t_k + \sum_{i=1}^{n} z_{ik} \right] \\
& \text{s.t.} & u_k + v_i \geq \lambda^k \displaystyle\sum_{j=1}^{n} C_{ij} x_{ij} & \text{s.t.} & \displaystyle\sum_{j=1}^{n} C_{ij} x_{ij} - t_k \leq z_{ik} \quad \forall\, i,k \\
& & u_k, v_i \geq 0 \quad \forall\, i,j,k & & z_{ik} \geq 0 \quad \forall\, i,k.
\end{array}
$$

Therefore, the linear relaxations of $MDOMP_{OT}$ and $MDOMP_{BHP}$ coincide. □

We consider next an alternative formulation for $MDOMP_{BHP}$ using three index variables x_{ij}^k.

$$
\begin{aligned}
(\mathbf{MDOMP_{BHP'}})\ \min & \sum_{k=1}^{n} u_k + \sum_{i=1}^{n} v_i & & & \\
s.t. & \sum_{j=1}^{n} y_j & = & p & \\
& \sum_{j=1}^{n}\sum_{k=1}^{n} x_{ij}^k & = & 1 & \forall\, i \\
& \sum_{j=1}^{n}\sum_{i=1}^{n} x_{ij}^k & = & 1 & \forall\, k \\
& \sum_{k=1}^{k} x_{ij}^k & \leq & y_j & \forall\, i,j \\
& u_k + v_i & \geq & \lambda^k \sum_{j=1}^{n}\sum_{k'=1}^{n} C_{ij} x_{ij}^{k'} & \forall\, i,k \\
& \sum_{k=1}^{n} u_k + \sum_{i=1}^{n} v_i & \geq & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} \lambda^k C_{ij} x_{ij}^k & (4.22) \\
& x_{ij}^k, y_j & \in & \{0,1\} & \forall\, i,j,k
\end{aligned}
$$

Theorem 8. *Let $MDOMP_{BHP'}$ be the alternative formulation using three-index variables then $z_{BHP}^{LP} = z_{BHP'}^{LP}$*

Proof. The proof follows applying Theorem 7 and the relationship between the variables x_{ij} and x_{ij}^k, namely $x_{ij} = \sum_{k=1}^{n} x_{ij}^k$. □

We can now prove that any of the specialized formulations for MDOMP, namely $MDOMP_{OT}$, $MDOMP_{BHP}$ or $MDOMP_{BHP'}$ are stronger than $DOMP_1$.

Theorem 9. $z_{BHP}^{LP} \geq z_1^{LP}$

Proof. Let $(x_{ij}, y, u, v) \in P_{BHP}$

$$x_{ij}^k = \begin{cases} x_{ij} & \text{if } \sum_{j'=1}^{n} C_{ij'} x_{ij'} \text{ is ordered in k-th position, i.e. } x_{(k)j}{}^1 \\ 0 & \text{otherwise} \end{cases}$$

Since the order is satisfied

$$\sum_{j=1}^{n} C_{(k-1)j} x_{(k-1)j} \leq \sum_{j=1}^{n} C_{(k)j} x_{(k)j} \quad k = 2, \ldots, n.$$

Applying the mapping

$$\sum_{j=1}^{n} C_{(k-1)j} x_{(k-1)j}^{k-1} \leq \sum_{j=1}^{n} C_{(k)j} x_{(k)j}^{k} \quad k = 2, \ldots, n.$$

By construction of variables x_{ij}^k, we add terms which value is zero

$$\sum_{j=1}^{n} C_{(k-1)j} x_{(k-1)j}^{k-1} + \sum_{j=1}^{n} \sum_{i:i\neq(k-1)}^{n} C_{ij} x_{ij}^{k-1} \leq \sum_{j=1}^{n} C_{(k)j} x_{(k)j}^{k} + \sum_{j=1}^{n} \sum_{i:i\neq(k)}^{n} C_{ij} x_{ij}^{k} \quad k = 2, \ldots, n$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} x_{ij}^{k-1} \leq \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} x_{ij}^{k} \quad k = 2, \ldots, n.$$

Other constraints from $DOMP_1$ are in model $DOMP_{BHP'}$ and by constraint (4.22) we have

$$z_{BHP}^{LP} = \sum_{k=1}^{n} u_k + \sum_{i=1}^{n} v_i \geq \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \lambda^k C_{ij} x_{ij}^k = z_1^{LP}$$

□

Corollary 4. $z_{OT}^{LP} \geq z_1^{LP}$

We have observed experimentally that in all our computational experiments (for all instances) $z_{BHP}^{LP} \geq z_2^{LP}$. Therefore, we suspect that this is also true in general. However, we have not found a valid proof of this fact.

Finally, we would like to report that the remaining formulations, namely $DOMP_3$ and $DOMP_4$ do not compare with $MDOMP_{BHP}$ since, as we report in the following examples, there are instances where all relationships can happen.

[1] Further we use this notation for the sake of simplicity

Notation	λ-vector	Name
T1	$(1, 1, \ldots, 1, 1)$	p-median
T2	$(0, 0, \ldots, 0, 1)$	p-center
T3	$(0, 0, \ldots, 0, 0, \underbrace{1, 1, \ldots, 1, 1}_{k})$	k-centrum
T7	λ random	Random
T8	$(\alpha, \alpha, \ldots, \alpha, \alpha, 1)$	Centdian

Table 4.2: Types of λ-vectors used in experiments

4.5 Computational experiments

This section is devoted to test whether it is advisable to apply specialized formulations for the MDOMP as compared with the general ones already presented in previous chapters. Moreover, we will also report a detailed computational study of formulations $MDOMP_{OT}$ and $MDOMP_{BHP}$ on bigger size instances. Detailed information can be found in the material included in the Appendix B. All our experiments have been carried out on a PC with two Intel Xeon processors with 3.46 GHz and 48 GB of RAM. The models were written in Mosel and solved using Xpress IVE 7.7.

For the first analysis, namely to test whether specialized formulations are more efficient for monotone problems, we use the cost matrices considered in Chapter 1 and the family of lambda vectors described in Table 4.2. The reader may note that we are also using the same notation as in Chapter 2, although we omit those lambdas that are non-monotone. We have built five different configuration of monotone random lambdas per size. For each considered size (number of clients) in this section we report on the average of 5 instances, thus the overall number of problems that we have solved 270 instances (2 different sizes of instances, 3 different number of servers, 5 cost matrices and 9 lambda vectors).

Table 4.3 reports on the average of the 270 instances for each of the formulations $DOMP_1$, $DOMP_2$ and $DOMP_3$, described in Chapter 1 and $MDOMP_{OT}$ and $MDOMP_{BHP}$. The computational experiment illustrates the theoretical result obtained in Theorem 7: integrality GAP of $MDOMP_{OT}$ and $MDOMP_{BHP}$ is the same. Moreover, on average, it is better than the best among the ones obtained with the formulations studied in Chap-

ter 1. Specifically, comparing the monotone formulations with the best non monotone one, namely $DOMP_3$, results in 155 out of 270 instances where $MDOMP_{OT}$ and $MDOMP_{BHP}$ have better gap that $DOMP_3$, 46 instances where both have the same gap and 69 cases where $DOMP_3$ gives the best lower bound. Finally, we observe that $DOMP_1$ and $DOMP_2$ always provide the worse linear programming relaxation (among the considered formulations) and in our results both formulations always got the same values in all the tested instances. This observation leads us to conjecture that both formulations are equivalent from a continuous relaxation point of view. This question is open so far and it will be part of our future research.

Next, we compare in Table 4.4 the CPU time to solve the same battery of problems for the two monotone formulations and the two best ones studied in Chapter 2, namely $DOMP_2$ and $DOMP_4(B\&C-3)$. We observe that the specialized formulations $MDOMP_{OT}$ and $MDOMP_{BHP}$ take advantage of the monotone structure of the λ-vectors, which results in an important reduction of the CPU time as compared with the general formulations. The comparison between $MDOMP_{OT}$ and $MDOMP_{BHP}$ does not show any significative difference, at least in medium size instances. For this reason we have designed another computational experiment where we will compare these two formulations on bigger size instances of MDOMP.

Formulation	**GAP**
$DOMP_1$	12,60%
$DOMP_2$	12,60%
$DOMP_3$	3.70%
$MDOMP_{BHP}$	1,79%
$MDOMP_{OT}$	1,79%

Table 4.3: Average integrality gaps of the different formulations for $n = 20, 40$.

Solution approach	**Time**	$\mathbf{T_{max}}$
$MDOMP_{BHP}(B\&B)$	0.13	0.93
$MDOMP_{OT}(B\&B)$	0.10	0.96
$DOMP_2(B\&B)$	56.75	914.35
$DOMP_4(B\&C-3)$	338.97(11)	7200.00

Table 4.4: CPU-Time of the different formulations for $n = 10, 20, 30$.

In the following, we wish to test the size limit that can be handled with the monotone formulations. In this analysis, we fix a CPU time limit of 7200 seconds and consider instances with a number of clients ranging from 100 to 180 because already for this size there are some cases that we could not solve

to optimality.

Overall, we have solved a number of 450 instances in this second computational experiment. The combinations of parameters were the following: 5 different number of clients, 5 cost matrices per each client size, 2 different number of servers and 9 different lambdas, as before.

We report the results in three tables (4.5-4.7). In addition, in Appendix B.4 the reader can found the results with more level of detail. All the tables show the same information. First, they contain the CPU time to solve the problems whenever the optimal solution is found and in those cases that some instances are not solved, we report in parenthesis the number of instances not solved to optimality. Second, they also report the average number of nodes explored in the B&B tree.

Table 4.5 shows the behavior of the formulations when the number of clients (n) increases. We can observe that, as the number of clients n increases, the problem becomes more difficult for both formulations, although $MDOMP_{OT}$ outperforms $MDOMP_{BHP}$ in CPU time and number of problems solved for $n \geq 160$.

	MDOMP$_{\mathbf{OT}}$		**MDOMP$_{\mathbf{BHP}}$**	
n	**Time**	**Nodes**	**Time**	**Nodes**
100	11.76	363	12.28	448
120	28.52	599	34.38	850
140	72.58	1880	103.52	2655
160	262.41	13722	622.29(1)	15572
180	518.95	13364	901.86(2)	15486

Table 4.5: Average CPU-Time and number of nodes by number of clients.

Moving to the next consideration, we also wish to know the dependence of formulations with respect to the number of facilities to be open. Table 4.6 compares the behavior of the formulations $MDOMP_{OT}$ and $MDOMP_{BHP}$, for two different number of facilities to be open. We observe that again $MDOMP_{OT}$ reports better performance, although in this case for large p $MDOMP_{BHP}$ outperforms $MDOMP_{OT}$. We note in passing that in both formulations we have applied the same preprocessing defined by Claim 2 in Chapter 2.

	$\mathbf{MDOMP_{OT}}$		$\mathbf{MDOMP_{BHP}}$	
	Time	**Nodes**	**Time**	**Nodes**
$p = \lfloor \frac{n}{2} \rfloor$	47.95	951	29.18	1655
$p = \lfloor \frac{n}{3} \rfloor$	309.74	11020	640.55(3)	12349

Table 4.6: Average CPU-Time and number of nodes by available servers.

In the final part of our computational results, we compare the results classifying the problems by the corresponding lambda vector in Table 4.7. Here we should distinguish two different patterns: we observe that the resolution times are rather competitive for the cases of the median ($T1$), center ($T2$) and cent-dian ($T8$); nevertheless, this is not the case for the k-centrum ($T5$) and the monotone random lambda ($T7$) where the problems get more difficult and they require very large branch-and-bound trees. At the moment, we do not have any clear explanation for this different behavior concerning the different lambda vectors.

	$\mathbf{MDOMP_{OT}}$		$\mathbf{MDOMP_{BHP}}$	
	Time	**Nodes**	**Time**	**Nodes**
T1	0.59	1	6.89	1
T2	1.26	13	1.26	13
T3	204.44	47657	946.37(2)	56429
T7	280.54	1240	410.49(1)	1316
T8	0.64	1	6.85	1

Table 4.7: Average CPU-Time and number of nodes by order weighted vector type.

As a general conclusion of our computational experiments we can state that it is advisable to use the formulations that exploit the monotonicity in the lambda vector. In general, formulation $MDOMP_{OT}$ performs better than $MDOMP_{BHP}$ since it requires shorter CPU-times with the only exception when the number of servers is large (around $p = n/2$). In this last case one should use $MDOMP_{BHP}$ because of its best performance and smaller CPU time.

Chapter 5

Conclusions and Future Research

This thesis has been devoted to the analysis of the Discrete Ordered Median Problem. The DOMP was introduced in the nineties as a powerful tool to model location problems. Since the very beginning it has attracted the interest of researchers in the field due to its applicability and also to its challenging theoretical characteristics. Among the different possibilities that one could have chosen to advance the knowledge on this problem we have concentrate on this thesis on the general problem applied to pure location problems: We have analyzed the discrete location problem with ordering requirements in the non-free self service case.

For this problem we have developed new formulations and studied properties of the polyhedral structure of some of these formulations. We have also developed *ad-hoc* resolution schemes based on those formulations, more specifically a branch-and-cut and a branch-and-cut-and-price algorithms. In addition, we have also analyzed a special case of DOMP where the vector of lambda weights is monotonic. This thesis has been structured in 5 chapters that cover the topics described above.

Chapter 1 is an introduction. There we have presented the state of the art on three of the main fields that are used in the thesis: Location theory, Combinatorial optimization, and Column generation. We included some of the main results that have been later applied in the rest of the chapters. In particular, we have introduced the main object of this thesis, namely the Ordered Median function and the Ordered Median Problem, in Section 1.1.1.

Chapter 2 presents new formulations for the Discrete Ordered Median

Problem based on order constraints (2.21) that are valid for the general non free self-service case. Furthermore, in this chapter we have proved theoretical relationships, in terms of their LP-gap, for different formulations of DOMP. According to the theoretical and computational results obtained, the main quality of the new formulations is that they provide substantial improvement of the integrality gap with respect to previously known ones. The most promising formulation in terms of its LP-gap is $DOMP_3$. For this reason, we have analyzed its polyhedral structure characterizing some facets of its assignment polytope. We have compared theoretically the different formulations proving the relationships among their LP-gaps and set of feasible solutions. Moreover, we have tested and compared empirically these formulations on several sets of instances giving some recommendations on which one should be used depending of the characteristics of the considered instances. We have observed that the LP-gap of $DOMP_1$ and $DOMP_2$ is always equal. It is currently an open question whether this property holds in general. This question will be a subject of our future research. This chapter has also opened another interesting line of research that consists in finding extensions of some of the existing formulations to exploit special structures of the lambda coefficients, as for instance the one in Marín et al. [2010]. Extensions based on the results in Marín et al. [2010] seem to require additional variables to handle the non free self-service case. A similar rationale can be also applied to the some of the new formulations in this chapter. Further theoretical and computational comparisons of the above mentioned new approach will be part of the subject of our future research.

Chapter 3 has been devoted to the development of a new approach to solve DOMP. We have develop a branch-and-cut-and-price approach for this problem. In Chapter 2 we have found that formulation $DOMP_4$ is rather promising in terms of its LP-gap. However, it needs a cubic number of variables, which can be prohibitive for large n. This is especially true because the linear programming relaxation can be extremely fractional. A second important problem of all known formulations for DOMP is the high degree of symmetry in case of equal costs or lambda. The reasons above lead us to introduce, in Chapter 3, a new formulation based on a different rationale. We have observed that a solution for DOMP is a partition of the clients together with their positions in the sorted vector of costs so that each subset of clients in the partition is allocated to the same facility. Therefore, based on formulation $DOMP_4$ we have developed a set partition formulation for DOMP. We have observed that this formulation is not a Dantzig-Wolfe reformulation

of the problem. In spite of that, we have proved that it always provides stronger LP-bounds that the original $DOMP_4$ formulation. From a theoretical point of view set partitioning formulations have an exponential number of variables, although in actual problems the number of variables necessary to represent a basic solution is much more reduced. In order to handle this implicit representation we have developed a column generation algorithm to solve the linear relaxation of this formulation of DOMP. It is worth mentioning that the pricing subproblem is solved rather efficiently since we have proved that its solution can be obtained by a dynamic programming algorithm with $O(n^3)$ worse case complexity. This approach has allowed us to address the resolution of DOMP with this formulation. We have also implemented a stabilization method based in Pessoa et al. [2010] which helps in reducing the number of variables to certify optimality. The column generation approach is embedded within a branch-and-cut-and-price. To improve its performance we have implemented a warm-start phase to choose an initial pool of columns that reduces significantly the final number of iterations of the column generation approach in each node of the branching tree. We have also developed a metaheuristic to find an initial feasible solution to the problem. A GRASP heuristic is applied for the first time to the DOMP and, according with the computational results, is a very useful one to obtain a good initial feasible solution before starting to solve the mixed integer linear problem. We have also developed a new adaptive branching rule and have adapted a family of valid inequalities from Chapter 2 that have been included in our algorithm. The computational results are rather promising. On top of that, this set partitioning reformulation opens a new avenue of research allowing new approaches to some other Ordered Median Problems beyond the pure discrete location one.

Chapter 4 deals with a special case of DOMP where the lambda vectors exhibit a monotonic property, namely $0 \leq \lambda_1 \leq \ldots \leq \lambda_n$. It is known that under this configuration of lambda vectors the ordered median operator (the ordered median function -OMf- as introduced in Chapter 1) is convex. This fact has motivated the interest of several researchers who tried to push further the sizes of well-solved instances (Ogryczack and Tamir [2003] and Blanco et al. [2014]). Nevertheless, the different approaches available in the literature have never been compared nor theoretically analyzed. This gap motivated our study in Chapter 4. In this chapter we have proceed by introducing the main two specific formulations that take advantage of the monotone structure of the lambda vectors: $MDOMP_{OT}$ and $MDOMP_{BHP}$. We

have compared between them these two specific formulations and also with all the others introduced in previous chapters. This comparison has stated the relationships in terms of their LP-bounds: $z^{LP}_{OT} = z^{LP}_{BHP} \leq z^{LP}_{DOMP_1}$ and also that z^{LP}_{OT} and $z^{LP}_{DOMP_3}$ do not compare because there are instances where each of them is superior to the other. Our computational experiments, in this chapter, have shown that the monotone formulations are more efficient than the general ones, applied to those families of problems where it makes sense, namely those with non-decreasing lambda vectors. Among the two that we compare $MDOMP_{OT}$ has shown to be more effective for medium to large size instances. In this regards, we have been able to solve instances up to 200 clients within a 7200 seconds of CPU time. Motivated by the excellent behavior exhibited by $MDOMP_{OT}$, we have introduced a new formulation $DOMP_{OTG}$ for the general DOMP problem (without monotone lambda vectors). The rationale is to represent the monotone part of the lambda vector using telescopic k-sums and the non-monotone one with sets of variables and constraints borrowed from $DOMP_2$. The results are promising and our preliminary computational tests show that $DOMP_{OTG}$ outperforms $DOMP_2$ for those lambda structures where the percentage of monotone positions in the lambda vector is at least 50 %. This idea of combining two rationale from two different paradigms in DOMP is new and it opens some avenues of research that may help in enlarging the limits of the sizes of problems solved efficiently.

The general conclusions and the statements of future research lines open by this thesis are included in this chapter 5. The thesis also includes several appendices including detailed information about our computational results.

Appendix A

Supplementary material for the Chapter 2 "New formulations and comparative study for DOMP"

This appendix reports detailed integrality GAP and CPU times for each of the eight types of λ-vector that have been considered in the computational results of Chapter 2. We follow the same notation so that $T1, ..., T8$, refer respectively to p-median, p-center, p-k-centrum, ($k1$+$k2$)-trimmed mean, the sequence of $0, 1, 0, 1...$, the sequence of $\ldots, 0, 0, 1, 0, 0, 1$, random elements and p-α-centdian (see Table 2.3).

A.1 GAP

In this section, we detail the results of Table 2.5 comparing the average integrality gap of the formulations in Chapter 2 for the different types of λ-vectors $T1, \ldots, T8$.

	Formulation				
λ-vector	**DOMP$_1$**	**DOMP$_2$**	**DOMP$_3$**	**DOMP$_4$**	**DOMP$_{4\cap1}$**
T1	0,14%	0,14%	0,14%	0,14%	0,14%
T2	36,04%	36,04%	18,81%	31,59%	31,57%
T3	16,38%	16,38%	4,53%	13,07%	13,07%
T4	5,53%	5,53%	0,14%	3,29%	3,29%
T5	2,16%	2,16%	0,91%	3,42%	2,16%
T6	4,01%	4,01%	1,78%	5,38%	3,99%
T7	3,32%	3,32%	0,77%	3,26%	3,00%
T8	3,69%	3,69%	1,02%	3,30%	3,30%

Table A.1: Average integrality GAP

A.2 Random instances: CPU times

This section shows eight tables reporting the detailed average CPU times of formulations $DOMP_2(B\&B)$ and $DOMP_{4\cap1}(B\&B)$ for the random data instances and for the the different types of λ-vectors: $T1, \ldots, T8$. The reader may note that these tables complement the information provided by Table 2.7 in Chapter 2. The numbers in parentheses report the number of instances that could not be solved to optimality within the time limit.

n	p	**Time (# unsolved)**		n	p	**Time (# unsolved)**	
		DOMP$_2$(B&B)	**DOMP$_{4\cap1}$(B&B)**			**DOMP$_2$(B&B)**	**DOMP$_{4\cap1}$(B&B)**
10	2	0.19	0.25	10	2	1.72	0.76
10	3	0.13	0.15	10	3	0.55	0.39
10	5	0.11	0.06	10	5	0.24	0.21
20	5	3.48	5.18	20	5	32.33	14.33
20	6	1.09	2.83	20	6	18.36	14.51
20	10	1.03	1.21	20	10	3.00	3.90
30	7	57.97	63.44	30	7	152.87	341.49
30	10	8.10	28.60	30	10	47.00	51.56
30	15	7.01	8.74	30	15	26.38	27.07
40	10	82.01	582.99	40	10	321.07	1762.57
40	13	35.93	203.66	40	13	140.96	778.25
40	20	21.99	39.74	40	20	66.24	106.23
50	12	237.49	2763.46	50	12	2873.03	4729.62(2)
50	16	97.04	1126.62	50	16	1964.89	1210.69
50	25	70.30	123.22	50	25	3762.76(2)	359.90
Average		**41.59**	**330.01**	**Average**		**627.43**	**626.76**

Table A.2: Results with random matrices: T1

Table A.3: Results with random matrices: T2

n	p	Time (# unsolved)	
		DOMP$_2$(B&B)	**DOMP$_{4\cap1}$**(B&B)
10	2	2.37	0.56
10	3	1.27	0.39
10	5	0.23	0.11
20	5	48.67	5.79
20	6	22.64	4.24
20	10	5.65	1.65
30	7	276.08	426.44
30	10	135.13	22.89
30	15	73.40	15.33
40	10	539.16	448.60
40	13	258.19	147.33
40	20	226.37	61.26
50	12	1710.53	4297.03(1)
50	16	1356.36	981.74
50	25	575.57	174.70
Average		**348.77**	**439.20**

Table A.4: Results with random matrices: T3

n	p	Time (# unsolved)	
		DOMP$_2$(B&B)	**DOMP$_{4\cap1}$**(B&B)
10	2	0.25	0.33
10	3	0.21	0.24
10	5	0.15	0.06
20	5	3.70	4.34
20	6	2.58	3.76
20	10	1.73	1.31
30	7	101.21	41.61
30	10	37.99	19.68
30	15	20.90	10.02
40	10	174.21	218.46
40	13	83.29	85.09
40	20	32.98	41.33
50	12	488.87	2038.03
50	16	319.03	305.56
50	25	122.41	138.23
Average		**92.63**	**193.87**

Table A.5: Results with random matrices: T4

n	p	**Time (# unsolved)**		n	p	**Time (# unsolved)**	
		DOMP$_{\mathbf{2}}$(B&B)	**DOMP$_{\mathbf{4\cap 1}}$(B&B)**			**DOMP$_{\mathbf{2}}$(B&B)**	**DOMP$_{\mathbf{4\cap 1}}$(B&B)**
10	2	2.21	0.41	10	2	2.11	0.63
10	3	0.55	0.23	10	3	1.24	0.33
10	5	0.23	0.08	10	5	0.24	0.09
20	5	20.76	6.42	20	5	28.93	6.59
20	6	7.91	3.00	20	6	12.92	2.94
20	10	3.05	1.38	20	10	7.42	1.41
30	7	140.47	72.14	30	7	325.47	116.99
30	10	71.33	15.55	30	10	110.88	21.96
30	15	44.92	10.36	30	15	32.67	10.71
40	10	594.70	752.32	40	10	1061.33	1127.77
40	13	239.04	125.60	40	13	453.66	140.17
40	20	62.04	41.42	40	20	209.93	43.81
50	12	4270.85(2)	4015.14	50	12	3440.55	4880.46(1)
50	16	1259.66	1014.72	50	16	2228.90	2646.65
50	25	480.13	133.00	50	25	1003.50	139.59
Average		**479.86**	**412.78**	**Average**		**594.65**	**609.34**

Table A.6: Results with random matrices: T5

Table A.7: Results with random matrices: T6

n	p	Time (# unsolved)		n	p	Time (# unsolved)	
		DOMP$_2$(B&B)	**DOMP$_{4\cap1}$**(B&B)			**DOMP$_2$**(B&B)	**DOMP$_{4\cap1}$**(B&B)
10	2	0.80	0.35	10	2	0.55	0.37
10	3	0.40	0.21	10	3	0.51	0.20
10	5	0.16	0.07	10	5	0.17	0.06
20	5	23.00	4.28	20	5	13.64	5.52
20	6	8.13	3.07	20	6	2.50	2.53
20	10	3.24	1.29	20	10	1.90	1.23
30	7	143.70	59.31	30	7	163.24	93.76
30	10	74.08	16.99	30	10	21.38	27.21
30	15	35.01	10.25	30	15	17.50	9.00
40	10	426.09	462.05	40	10	486.36	305.21
40	13	291.19	82.41	40	13	126.39	203.27
40	20	144.41	43.66	40	20	62.24	37.42
50	12	1588.92	2453.27(2)	50	12	1483.75	3224.39
50	16	826.26	335.67	50	16	499.76	687.93
50	25	605.61	139.36	50	25	215.36	123.81
Average		**278.07**	**240.82**	**Average**		**206.35**	**314.79**

Table A.8: Results with random matrices: T7

Table A.9: Results with random matrices: T8

A.3 Beasley instances: CPU times

This section shows the tables reporting the detailed average CPU times of formulations $DOMP_2(B\&B)$ and $DOMP_{4C}(B\&C-3)$ for Beasley's data instances and for the the different types of λ-vectors: $T1,\ldots,T8$. The reader may note that these tables complement the information provided by Table 2.9 in Chapter 2. The numbers in parentheses report the number of instances that could not be solved to optimality within the time limit.

Problem	**Time (#unsolved)**		**Problem**	**Time (#unsolved)**	
	$\mathbf{DOMP_2(B\&B)}$	$\mathbf{DOMP_{4C}(B\&C-3C)}$		$\mathbf{DOMP_2(B\&B)}$	$\mathbf{DOMP_{4C}(B\&C-3C)}$
pmed1	6.92	55.31	pmed1	108.27	226.46
pmed2	8.34	38.56	pmed2	26.53	77.63
pmed3	5.85	36.28	pmed3	16.41	70.54
pmed4	6.03	3.34	pmed4	42.13	53.30
pmed5	4.40	2.87	pmed5	5.61	11.07
pmed6	24.24	7200.00(1)	pmed6	516.44	1688.10
pmed7	47.44	347.24	pmed7	421.79	876.15
pmed8	20.14	4.36	pmed8	546.06	629.94
pmed9	17.50	4.29	pmed9	346.25	449.83
pmed10	6.56	2.43	pmed10	165.08	227.87
pmed11	50.95	108.90	pmed11	1879.64	3807.53
pmed12	352.21	896.05	pmed12	590.87	2444.95
pmed13	25.24	1194.59	pmed13	140.61	1704.65
pmed14	33.86	36.86	pmed14	424.07	1343.37
pmed15	14.41	3.20	pmed15	49.83	542.24
pmed16	761.88	2058.53	pmed16	726.70	7199.21(1)
pmed17	848.49	2116.27	pmed17	678.12	6983.10
pmed18	43.60	62.18	pmed18	224.94	4347.78
pmed19	18.22	48.25	pmed19	130.84	1714.46
pmed20	22.91	4.51	pmed20	95.45	1639.34

Table A.10: Results using Beasley's data set: T1

Table A.11: Results using Beasley's data set: T2

Problem	Time (#unsolved)	
	DOMP$_2$(B&B)	**DOMP$_{4C}$(B&C − 3C)**
pmed1	211.46	303.66
pmed2	204.31	139.96
pmed3	192.41	69.19
pmed4	77.74	14.83
pmed5	42.51	6.22
pmed6	1171.65	7199.39(1)
pmed7	510.68	1285.17
pmed8	384.37	604.27
pmed9	179.87	37.93
pmed10	70.04	10.77
pmed11	963.52	1520.28
pmed12	1542.15	3890.51
pmed13	765.99	999.69
pmed14	586.23	178.80
pmed15	96.97	50.17
pmed16	3014.77	7200.00(1)
pmed17	1345.46	3373.99
pmed18	402.20	770.66
pmed19	272.31	291.62
pmed20	551.30	114.13

Table A.12: Results using Beasley's data set: T3

Problem	Time (#unsolved)	
	DOMP$_2$(B&B)	**DOMP$_{4C}$(B&C − 3C)**
pmed1	19.60	38.89
pmed2	23.97	48.88
pmed3	86.73	34.00
pmed4	10.73	4.02
pmed5	7.34	3.23
pmed6	157.15	1027.20
pmed7	351.69	435.35
pmed8	53.92	140.88
pmed9	29.03	16.04
pmed10	12.13	2.58
pmed11	407.99	672.51
pmed12	875.06	1115.44
pmed13	236.48	370.46
pmed14	128.35	152.61
pmed15	39.94	4.34
pmed16	1409.52	1486.37
pmed17	667.46	1208.51
pmed18	237.20	287.65
pmed19	120.04	72.91
pmed20	71.41	10.50

Table A.13: Results using Beasley's data set: T4

Problem	Time (#unsolved)	
	DOMP$_2$(B&B)	**DOMP$_{4C}$(B&C − 3C)**
pmed1	137.99	54.50
pmed2	46.82	44.18
pmed3	67.40	25.38
pmed4	16.42	3.37
pmed5	22.00	2.90
pmed6	465.36	7200.00(1)
pmed7	269.49	2941.78
pmed8	381.89	349.08
pmed9	96.33	3.91
pmed10	35.83	2.72
pmed11	295.80	7200.00(1)
pmed12	1109.52	5103.53
pmed13	2216.15	7199.03(1)
pmed14	499.75	482.92
pmed15	49.41	3.26
pmed16	2699.31	6290.68
pmed17	2334.41	7199.84(1)
pmed18	601.00	7199.46(1)
pmed19	113.96	66.77
pmed20	77.59	5.28

Table A.14: Results using Beasley's data set: T5

Problem	Time (#unsolved)	
	DOMP$_2$(B&B)	**DOMP$_{4C}$(B&C − 3C)**
pmed1	207.40	206.21
pmed2	135.93	36.07
pmed3	411.76	88.81
pmed4	41.77	3.56
pmed5	43.77	3.03
pmed6	3595.93	7200.00(1)
pmed7	2108.41	7199.51(1)
pmed8	7200.00(1)	4160.82
pmed9	228.79	14.91
pmed10	43.40	2.88
pmed11	7200.00(1)	7199.00(1)
pmed12	7200.09(1)	7200.00(1)
pmed13	7199.57(1)	7198.94(1)
pmed14	1471.95	485.10
pmed15	112.41	3.58
pmed16	3348.32	7200.00(1)
pmed17	3679.42	7198.95(1)
pmed18	7200.00(1)	7199.35(1)
pmed19	1736.89	313.48
pmed20	1091.33	6.87

Table A.15: Results using Beasley's data set: T6

Problem	Time (#unsolved)	
	DOMP$_2$(B&B)	**DOMP$_{4C}$(B&C − 3C)**
pmed1	84.52	39.77
pmed2	98.40	19.56
pmed3	90.84	18.19
pmed4	40.26	3.42
pmed5	20.89	2.91
pmed6	695.69	4851.31(1)
pmed7	711.95	3753.20(1)
pmed8	288.05	125.07
pmed9	122.38	3.97
pmed10	39.36	2.47
pmed11	1030.77	4490.05
pmed12	1519.32	5717.80(3)
pmed13	2707.22	5289.07(1)
pmed14	579.99	181.29
pmed15	129.90	3.21
pmed16	4064.69(1)	7049.42(4)
pmed17	4841.21(1)	7020.19(4)
pmed18	2596.90	5639.34(1)
pmed19	372.41	128.10
pmed20	192.35	4.72

Table A.16: Results using Beasley's data set: T7

Problem	Time (#unsolved)	
	DOMP$_2$(B&B)	**DOMP$_{4C}$(B&C − 3C)**
pmed1	21.59	52.00
pmed2	12.95	3.84
pmed3	11.32	35.77
pmed4	23.11	3.27
pmed5	15.17	2.88
pmed6	251.02	1315.97
pmed7	141.96	654.99
pmed8	160.73	4.40
pmed9	53.71	3.68
pmed10	20.24	2.48
pmed11	294.44	1470.30
pmed12	738.29	1583.50
pmed13	249.74	588.62
pmed14	71.06	49.06
pmed15	49.21	3.17
pmed16	847.88	4605.99
pmed17	1523.26	4858.95
pmed18	269.84	1492.55
pmed19	42.97	24.20
pmed20	202.77	4.89

Table A.17: Results using Beasley's data set: T8

Appendix B

Supplementary material for the Chapter 4 "The monotone ordered median problem"

Section B.1 details the study done in Section 4.2.1. Sections B.2, B.3 and B.4 report detailed integrality GAP and CPU times respectively for each of the five type of λ-vector that have been considered in the computational results of Section 4.5. We follow the same notation so that $T1, T2, T3, T7, T8$, refer respectively to p-median, p-center, p-k-centrum, random elements and p-α-centdian (see Table 4.2).

B.1 Detailed results for comparison between formulations $DOMP_{OTG}$ and $DOMP_2$

Next tables detail the information of Table 4.1.

	$\mathbf{DOMP_{OTG}}$		$\mathbf{DOMP_2}$	
	Time	GAP	Time	GAP
$\alpha \geq \lfloor 0.9n \rfloor$	1.28	5.58	1.32	16.22
$\alpha \geq \lfloor 0.7n \rfloor$	1.31	5.52	1.12	13.11
$\alpha \geq \lfloor 0.5n \rfloor$	1.32	4.85	0.89	10.06
$\alpha \geq \lfloor 0.3n \rfloor$	3.16	19.54	0.98	7.46

Table B.1: CPU-Time and integrality GAP for $n \geq 10, p = 2$.

	$\mathbf{DOMP_{OTG}}$		$\mathbf{DOMP_2}$	
	Time	GAP	Time	GAP
$\alpha \geq \lfloor 0.9n \rfloor$	0.57	2.48	0.65	13.53
$\alpha \geq \lfloor 0.7n \rfloor$	0.90	2.81	0.44	10.96
$\alpha \geq \lfloor 0.5n \rfloor$	0.78	2.11	0.34	8.05
$\alpha \geq \lfloor 0.3n \rfloor$	2.36	18.13	0.48	5.87

Table B.2: CPU-Time and integrality GAP for $n = 10, p = 3$.

	$\mathbf{DOMP_{OTG}}$		$\mathbf{DOMP_2}$	
	Time	GAP	Time	GAP
$\alpha \geq \lfloor 0.9n \rfloor$	0.12	0.29	0.28	13.86
$\alpha \geq \lfloor 0.7n \rfloor$	0.25	0.68	0.28	10.86
$\alpha \geq \lfloor 0.5n \rfloor$	0.21	0.35	0.26	7.82
$\alpha \geq \lfloor 0.3n \rfloor$	1.70	14.65	0.22	5.29

Table B.3: CPU-Time and integrality GAP for $n = 10, p = 5$.

	$\mathbf{DOMP_{OTG}}$		$\mathbf{DOMP_2}$	
	Time	GAP	Time	GAP
$\alpha \geq \lfloor 0.9n \rfloor$	4.15	3.93	47.80	14.98
$\alpha \geq \lfloor 0.7n \rfloor$	5.34	3.89	41.31	13.08
$\alpha \geq \lfloor 0.5n \rfloor$	7.12	5.29	28.08	9.24
$\alpha \geq \lfloor 0.3n \rfloor$	10.68	18.78	26.34	8.36

Table B.4: CPU-Time and integrality GAP for $n = 20, p = 5$.

	$\mathbf{DOMP_{OTG}}$		$\mathbf{DOMP_2}$	
	Time	GAP	Time	GAP
$\alpha \geq \lfloor 0.9n \rfloor$	2.72	0.88	16.27	12.57
$\alpha \geq \lfloor 0.7n \rfloor$	4.07	0.92	10.68	10.82
$\alpha \geq \lfloor 0.5n \rfloor$	6.55	2.25	6.61	7.48
$\alpha \geq \lfloor 0.3n \rfloor$	9.70	15.22	8.32	6.80

Table B.5: CPU-Time and integrality GAP for $n = 20, p = 6$.

	$\mathbf{DOMP_{OTG}}$		$\mathbf{DOMP_2}$	
	Time	GAP	Time	GAP
$\alpha \geq \lfloor 0.9n \rfloor$	0.64	0.07	4.45	10.86
$\alpha \geq \lfloor 0.7n \rfloor$	1.57	0.15	3.00	8.98
$\alpha \geq \lfloor 0.5n \rfloor$	4.96	0.80	2.37	6.21
$\alpha \geq \lfloor 0.3n \rfloor$	9.15	12.22	2.16	6.10

Table B.6: CPU-Time and integrality GAP for $n = 20, p = 10$.

	$\mathbf{DOMP_{OTG}}$		$\mathbf{DOMP_2}$	
	Time	GAP	Time	GAP
$\alpha \geq \lfloor 0.9n \rfloor$	26.76	4.99	415.28	14.68
$\alpha \geq \lfloor 0.7n \rfloor$	35.73	4.98	294.83	11.74
$\alpha \geq \lfloor 0.5n \rfloor$	42.88	5.77	190.36	7.74
$\alpha \geq \lfloor 0.3n \rfloor$	180.93	50.66	89.12	3.06

Table B.7: CPU-Time and integrality GAP for $n = 30, p = 7$.

	$\mathbf{DOMP_{OTG}}$		$\mathbf{DOMP_2}$	
	Time	GAP	Time	GAP
$\alpha \geq \lfloor 0.9n \rfloor$	24.49	1.17	167.10	12.13
$\alpha \geq \lfloor 0.7n \rfloor$	36.95	1.42	135.03	9.31
$\alpha \geq \lfloor 0.5n \rfloor$	43.35	2.63	79.34	5.78
$\alpha \geq \lfloor 0.3n \rfloor$	189.12	51.91	40.82	2.24

Table B.8: CPU-Time and integrality GAP for $n = 30, p = 10$.

	$\mathbf{DOMP_{OTG}}$		$\mathbf{DOMP_2}$	
	Time	GAP	Time	GAP
$\alpha \geq \lfloor 0.9n \rfloor$	7.79	0.14	60.05	10.85
$\alpha \geq \lfloor 0.7n \rfloor$	24.28	0.36	51.94	8.06
$\alpha \geq \lfloor 0.5n \rfloor$	44.40	1.40	28.41	4.86
$\alpha \geq \lfloor 0.3n \rfloor$	160.04	52.65	11.94	1.81

Table B.9: CPU-Time and integrality GAP for $n = 30, p = 15$.

B.2 GAP

In this section, we detail the result of Table 4.3 comparing the average integrality gap of the formulations in Chapter 4 for the different types of λ-vectors $T1, T2, T3, T7, T8$.

	Formulation				
λ-vector	**$\mathbf{DOMP_1}$**	**$\mathbf{DOMP_2}$**	**$\mathbf{DOMP_3}$**	**$\mathbf{DOMP_{BHP}}$**	**$\mathbf{DOMP_{OT}}$**
T1	0,14%	0,14%	0,14%	0,14%	0,14%
T2	36,04%	36,04%	17,92%	4,95%	4,95%
T3	16,18%	16,18%	3,83%	3,30%	3,30%
T7	11,66%	11,66%	2,14%	1,43%	1,43%
T8	2,73%	2,73%	0,65%	0,52%	0,52%

Table B.10: Average integrality GAP

B.3 Random instances: CPU times (plerminary study)

This section shows five tables reporting the detailed average CPU times of formulations $MDOMP_{BHP}(B\&B)$, $MDOMP_{OT}(B\&B)$, $DOMP_2(B\&B)$ and $DOMP_4(B\&C-3)$ for the random data instances and for the the different types of λ-vectors: $T1, T2, T3, T7, T8$. The reader may note that these tables complement the information provided by Table 4.4 in Chapter 4. The numbers in parenthesis report the number of instances that could not be solved to optimality within the time limit.

Solution approach	**Time**	$\mathbf{T_{max}}$
$MDOMP_{BHP}(B\&B)$	0.04	0.17
$MDOMP_{OT}(B\&B)$	0.01	0.08
$DOMP_2(B\&B)$	5.12	54.05
$DOMP_4(B\&C-3)$	5.46	31.90

Table B.11: CPU-Time of the different formulations for $n = 10, 20, 30$: T1.

Solution approach	**Time**	$\mathbf{T_{max}}$
$MDOMP_{BHP}(B\&B)$	0.05	0.28
$MDOMP_{OT}(B\&B)$	0.04	0.28
$DOMP_2(B\&B)$	30.78	285.49
$DOMP_4(B\&C-3)$	128.05	2088.47

Table B.12: CPU-Time of the different formulations for $n = 10, 20, 30$: T2.

Solution approach	**Time**	$\mathbf{T_{max}}$
$MDOMP_{BHP}(B\&B)$	0.12	0.50
$MDOMP_{OT}(B\&B)$	0.05	0.20
$DOMP_2(B\&B)$	53.59	349.66
$DOMP_4(B\&C-3)$	460.64(2)	7200.00

Table B.13: CPU-Time of the different formulations for $n = 10, 20, 30$: T3.

Solution approach	**Time**	$\mathbf{T_{max}}$
$MDOMP_{BHP}(B\&B)$	0.19	0.93
$MDOMP_{OT}(B\&B)$	0.16	0.96
$DOMP_2(B\&B)$	80.34	914.35
$DOMP_4(B\&C-3)$	487.61(9)	7200.00

Table B.14: CPU-Time of the different formulations for $n = 10, 20, 30$: T7.

Solution approach	**Time**	$\mathbf{T_{max}}$
$MDOMP_{BHP}(B\&B)$	0.05	0.30
$MDOMP_{OT}(B\&B)$	0.02	0.12
$DOMP_2(B\&B)$	19.58	300.86
$DOMP_4(B\&C-3)$	18.53	377.78

Table B.15: CPU-Time of the different formulations for $n = 10, 20, 30$: T8.

B.4 Random instances: CPU times and number of nodes

This section shows five tables reporting the detailed average CPU times and number of nodes of formulations $MDOMP_{BHP}(B\&B)$ and $MDOMP_{OT}(B\&B)$ for the random data instances and for the the different types of λ-vectors: $T1, T2, T3, T7, T8$. The reader may note that these tables complement the information provided by Tables 4.5, 4.6 and 4.7 in Chapter 4. The numbers in parenthesis report the number of instances that could not be solved to optimality within the time limit.

	MDOMP$_{\mathbf{OT}}$		**MDOMP$_{\mathbf{BHP}}$**	
n	**Time**	**Nodes**	**Time**	**Nodes**
100	0.30	1	1.81	1
120	0.41	1	3.44	1
140	0.61	1	6.94	1
160	0.73	1	8.90	1
180	0.92	1	13.34	1

Table B.16: Average CPU-Time and number of nodes by number of clients: T1.

	MDOMP$_{\mathbf{OT}}$		**MDOMP$_{\mathbf{BHP}}$**	
n	**Time**	**Nodes**	**Time**	**Nodes**
100	0.68	16	0.66	16
120	0.89	31	0.89	31
140	1.01	1	1.04	1
160	1.71	15	1.73	15
180	2.00	4	1.96	4

Table B.17: Average CPU-Time and number of nodes by number of clients: T2.

	$\mathbf{MDOMP_{OT}}$		$\mathbf{MDOMP_{BHP}}$	
n	**Time**	**Nodes**	**Time**	**Nodes**
100	3.25	2416	14.45	3116
120	6.21	3331	35.97	5249
140	32.34	13325	227.57	20196
160	497.58	114727	1971.31(1)	127949
180	482.85	104484	2482.56(1)	125634

Table B.18: Average CPU-Time and number of nodes by number of clients: T3.

	$\mathbf{MDOMP_{OT}}$		$\mathbf{MDOMP_{BHP}}$	
n	**Time**	**Nodes**	**Time**	**Nodes**
100	20.25	167	18.34	181
120	49.76	406	53.11	474
140	123.72	718	137.71	739
160	372.18	1752	722.01	2437
180	836.78	3157	1121.26(1)	2748

Table B.19: Average CPU-Time and number of nodes by number of clients: T7.

	$\mathbf{MDOMP_{OT}}$		$\mathbf{MDOMP_{BHP}}$	
n	**Time**	**Nodes**	**Time**	**Nodes**
100	0.33	1	1.93	1
120	0.42	1	3.52	1
140	0.67	2	7.59	2
160	0.81	1	8.60	1
180	0.96	1	12.62	1

Table B.20: Average CPU-Time and number of nodes by number of clients: T8.

Bibliography

T. Achterberg. Scip: Solving constraints integer programs. *Matemathical Programming Computation*, 1(1):1–41, 2009.

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

D. L. Appelgren. A column generation algorithm for a ship scheduling problem. *Transportation Science*, 3:53–68, 1969.

D. Applegate, R.E. Bixby, V. Chvátal, and W. Cook. Finding cuts in TSP, technical report. *Operations Research Letters*, 95-05, 1995.

C. Arbib, M. Labbé, and M. Servilio. Scheduling two chains of unit jobs on one machine: A polyhedral study. *Networks*, 58-2:103–113, 2011.

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

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

C. Barnhart, C. A. Hane, and P. H. Vance. Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research*, 48(2):318–326, 2000.

J.E. Beasley. OR-Library, 2012. http://people.brunel.ac.uk/ mastjjb/jeb/info.html.

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

V. Blanco, S. El Haj Ben Ali, and J. Puerto. Revisiting several problems and algorithms in continuous location wih $l_\tau-$norm. *Computational Optimization and Applications*, 58(3):563–595, 2014.

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. ISSN 0305-0548.

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

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

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

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

G. Cornuéjols and J.M. Thizy. Some facets of the simple plant location polytope. *Mathematical Programming*, (23):50–74, 1982.

G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

G. B. Dantzig, A. Orden, and P. Wolfe. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, 5(2):183–195, 1955.

M. S. Daskin. *Network and Discrete Location: Models, Algorithms, and Applications.* J. Wiley & sons, New York (N.Y.), Chichester, Weinheim, 1995.

M. S. Daskin. *Network and Discrete Location: Models, Algorithms, and Applications, Second Edition.* J. Wiley & sons, Hoboken, New Jersey, 2013.

I. R. de Farias Jr. A family of facets for the uncapacitated p-median polytope. *Operations Research Letters*, 28(4):161–167, 2001.

J. Desrosiers and M. Lübecke. A primer in column generation. In G. Desaulniers, J. Desrosiers, and M. M. Salomon, editors, *Column Generation*. Kluwer, 2005.

P. Domínguez-Marín. *The Discrete Ordered Median Problem: Models and Solution Methods*. Kluwer, 2003.

Z. Drezner. *Facility Location: A Survey of Applications and Methods.* Springer, New York, 1995.

Z. Drezner and H. Hamacher. *Facility Location: A Survey of Applications and Therory.* Springer, New York, 2002.

O. du Merle and J.P. Vial. Proximal ACCPM, acutting plane method for column generation and Lagrangean relaxation: application to the p-median problem. Technical report, HEC Genève, Université de Genève, 2002.

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

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

T. A. Feo and M. G. C. Resende. A probabilistic huristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

T. A. Feo and M. G. C. Resende. Greedy ramdomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

E. Fernández, J. Puerto, and A. M. Rodríguez-Chía. On discrete optimization with ordering. *Annals of Operations Research*, 207(1):83–96, 2013.

T.L.R. Ford and D.R. Fulkerson. A suggested computation for maximal multicommodity networks flows. *Management Science*, 5:97–101, 1958.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9:849–859, 1961.

P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem–part ii. *Operations Research*, 11:863–888, 1963.

M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*. Algorithms and combinatorics. Springer-Verlag, Berlin, New York, 1988.

M. Guignard. Fractional vertices cuts and facets of the simple plant location problem. *Mathematical Programming Study*, (12):152–160, 1980.

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

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

E. L. Johnson. Modeling and strong linear programs for mixed integer programming. In SteinW. Wallace, editor, *Algorithms and Model Formulations in Mathematical Programming*, volume 51 of *NATO ASI Series*, pages 1–43. Springer Berlin Heidelberg, 1989. ISBN 978-3-642-83726-5.

J. Kalcsics, S. Nickel, J. Puerto, and A. Tamir. Algorithmic results for ordered median problems. *Operations Research Letters*, 30:149–158, 2002.

J. Kalcsics, S. Nickel, J. Puerto, and A. M. Rodríguez-Chía. Distribution systems design with role dependent objectives. *European Journal of Operational Research*, 202:491–501, 2010.

N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

L.G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979.

L.G. Khachiyan. A polynomial algorithm in linear programming. *Zhurnal Vychisditel'noi Matematiki i Matematicheskoi Fiziki*, 20:51–68, 1980.

A. Klose and S. Görtz. A branch-and-price algorithm for the capacitated facility location problem. *European Journal of Operational Research*, 179: 1109–1125, 2007.

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*, 2016. ISSN 0305-0548. doi: http://dx.doi.org/10.1016/j.cor.2016.06.004. URL `http://www.sciencedirect.com/science/article/pii/S0305054816301368`.

G. Laporte, S. Nickel, and F. Saldanha. *Location Science.* Springer International Publishing Switzerland, 2002.

R. Larson and A. Odoni. *Urban Operations Research.* Prentice-Hall, New York, 1981.

L.A.N. Lorena and E.L.F. Senne. A column generation approach to capacitated p-median problems. *Computers & Operations Research*, 31(6):863 – 876, 2004.

A. Marín, S. Nickel, J. Puerto, and S. Velten. A flexible model and efficient solution strategies for discrete location problems. *Discrete Applied Mathematics*, 157(5):1128–1145, 2009. ISSN 0166-218X.

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

P. Mirchandani and R. Francis. *Discrete Location Theory.* J. Wiley & sons, New York, 1990.

G.L. Nemhauser. Column generation for linear and integer programming. *Documenta Mathematica-Extra Volume ISMP*, pages 65–73, 2012.

G.L. Nemhauser and L.E. Trotter. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.

S. Nickel. Discrete ordered weber problems. In *Operations Research Proceedings 2000*, pages 71–76. Springer Verlag, 2001.

S. Nickel and J. Puerto. *Location Theory: A Unified Approach.* Springer Verlag, 2005.

S. Nickel, J. Puerto, and A. M. Rodríguez-Chía. MCDM location problems. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 761–787. Springer New York, New York, NY, 2005.

S. Nickel, J. Puerto, and A. M. Rodríguez-Chía. *Location Science*, chapter Location problems with multiple criteria, pages 205–247. Springer International Publishing Switzerland, 2015.

W. Ogryczack and A. Tamir. Minimizing the sum of the $k-$largest functions in linear time. *Information Processing Letters*, 85:117–122, 2003.

M. W. Padberg. On the Facial Structure of Set Packing Polyhedra. *Mathematical Programming*, 5:199–215, 1973.

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

J. Puerto. Lecturas en teoría de localización. Technical report, Universidad de Sevilla. Secretariado de Publicaciones, 1996.

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

J. Puerto, A. B. Ramos, and A. M. Rodríguez-Chía. Single-allocation ordered median hub location problems. *Computers & Operations Research.*

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*, 234:61–76, 2014.

J. Reese. Solution methods for the capacitated p-median problem: an annotated bibliography. *Networks*, 48(3):125–142, 2006.

D. M. Ryan and A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling*, pages 269–280. North-Holland, Amsterdan, 1981.

A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, Inc., New York, NY, USA, 1986.

E.L.F Senne and L.A.N Lorena. Stabilizing column generation using Lagrangean/surrogate relaxation: an application to p-median location problems. In *Proceedings of the EURO2001 Conference*, Rotterdam, July 2001. Erasmus University.

E.L.F. Senne, L.A.N. Lorena, and Marcos. A. Pereira. A branch-and-price approach to p-median location problems. *Computers & Operations Research*, 32:1655 – 1664, 2005.

P. H. Vance, C. Barnhart, E. L. Johnson, and G. L. Nemhauser. Solving binary cutting stock problems by column generation and branch-and-bound. *Computational Optimization and Applications*, 3:111–130, 1994.

I. Vasilyev, X. Klimentova, and M. Boccia. Polyhedral study of simple plant location problem with order. *Operations Research Letters*, (2):153–158, 2013.

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