

# A Heuristic Constraint Programming Method for the p-Median Problem With Distance Constraints

The European Journal on Artificial  
Intelligence  
1–28

© The Author(s) 2025

Article reuse guidelines:

sagepub.com/journals-permissions

DOI: 10.1177/30504554251344177

journals.sagepub.com/home/eai



Panteleimon Iosif<sup>1</sup> , Nikolaos Ploskas<sup>1</sup>   
and Kostas Stergiou<sup>1</sup> 

## Abstract

In a facility location problem, we seek to locate a set of facilities in an area, where demand points (clients) may be present, so that some criterion is optimized. A well-known facility location problem is the p-median problem, where we seek to minimize the sum of distances between demand points and their nearest facility. We consider a variant of the p-median problem where distance constraints exist between facilities and between facilities and demand points. This problem can be used to model the requirements that arise when locating semi-obnoxious facilities. We first introduce integer linear programming and constraint programming (CP) models and implement them in Gurobi and OR-Tools, respectively. Then, we describe a greedy heuristic that can be used to prune branches during a search within an incomplete CP solver. We also introduce a number of domain-specific value ordering heuristics that can be applied within such a solver. The developed method is evaluated under different problem generation models, and compared to Gurobi and OR-Tools. The results demonstrate that our method is more robust than the two complete solvers, significantly outperforming either of them in certain classes of problems, both in terms of run times and the quality of the solutions found.

## Keywords

constraint programming, facility location, distance constraints, value ordering, optimization

Received: January 28, 2025; accepted: May 6, 2025

## 1 Introduction

Facility location problems are widely studied in operations research (OR), artificial intelligence (AI), computational geometry, and other disciplines. In such problems, we seek to locate a set of facilities in an area, where demand points (clients) that will be serviced by the facilities may be present, so that some criterion is optimized. The optimization criterion largely depends on the types of facilities to be located. When the facilities have beneficial properties (e.g., pharmacies), we typically want to locate them close to clients. In contrast, when the facilities are (ob)noxious, that is, they have hazardous effects (e.g., dump sites), we seek to locate them far from clients and/or each other. In between, we have the class of semi-obnoxious facilities that have both desirable and undesirable properties.

Prime examples of the first two categories are the p-median and p-dispersion problems, respectively. In the former, we seek to locate p facilities in an area where demand points are present, so that the sum of distances between demand points and their nearest facility is minimized. Hence, we have a minsum objective. In the latter, we seek to locate p facilities so that the closest distance between any two facilities is maximized. Hence, in this case, we have a maxmin objective. Apart from the difference in the type of objective function, another crucial difference between the two problems is that in p-dispersion there are no clients to be serviced by the facilities.

<sup>1</sup>Department of Electrical and Computer Engineering, University of Western Macedonia, Kozani, Greece

### Corresponding Author:

Panteleimon Iosif, Department of Electrical and Computer Engineering, University of Western Macedonia, Kozani, Greece.

Email: p.iosif@uowm.gr

Here, we are concerned with a variant of the p-median problem where we again seek to minimize the sum of the distances between the clients and their closest facility, but in this case, we also have hard distance constraints between facilities and also between facilities and clients. These constraints impose lower bounds on the distance between the locations of any two facilities and also between any facility and any client. In the rest of the paper, we call this problem p-median with distance constraints (pMD). Although this problem was proposed as far back as 1984 (Moon & Chaudhry, 1984), and despite its usefulness in modeling the location of semi-obnoxious facilities (Krarup et al., 2002), it has received little attention. We consider the discrete uncapacitated case where the potential location points for the facilities are given, the facilities are heterogeneous (i.e., they have varying properties), and after the location has been completed, demand points are serviced by the facility that is located closest to them.

As an example of a pMD, consider the problem of locating a set of filling (gas) stations in an area. Filling stations are typical semi-obnoxious facilities. For their convenience, clients wish to have them placed at a close distance, but not too close, given the danger involved in case of an accident, as well as for other reasons (e.g., traffic and pollution). In addition, there are safety regulations that must be taken into account when opening filling stations. Among other constraints, such regulations impose minimum safety distances between the stations in an area and also between stations and clients. Another property that facilities such as filling stations have is that they are often heterogeneous. That is, they can have varying characteristics. In the case of filling stations such characteristics may be the overall size, tank capacity, types of fuel on offer (gasoline, diesel, natural gas, etc.), number of available fuel dispensers, etc. Such heterogeneous characteristics imply that the minimum allowed distances may not be uniform (e.g., large stations must be placed further away than smaller ones).

Location problems with distance constraints have received relatively little attention within the vast literature on location problems (Berman & Huang, 2008; Chaudhry et al., 1986; Comley, 1995; Moon & Chaudhry, 1984; Moon & Papayanopoulos, 1991; Tansel et al., 1982). Very recently, p-dispersion with distance constraints was studied, comparing integer linear programming (ILP) to exact and heuristic constraint programming (CP) approaches (Iosif et al., 2024b; Ploskas et al., 2023).

We start by giving an ILP formulation for the pMD, based on the ReVelle & Swain formulation for the p-median problem (ReVelle & Swain, 1970), extending it to cover the case of heterogeneous facilities and adding distance constraints. Then, we describe a CP model that can capture the pMD as a constraint satisfaction/optimization problem in a natural way. The CP (respectively, ILP) model has been implemented in the CP-SAT OR-Tools (respectively, Gurobi) solver. These are state-of-the-art solvers for CP and ILP, respectively. Experimental results reveal that these solvers struggle with large, hard instances of the pMD, often failing to find a feasible solution within 1 h of central processing unit (CPU) time. In addition, such large instances have extensive, and sometimes prohibitive, memory requirements due to the large sizes of the models. These problems highlight the need for heuristic and lightweight approaches that may sacrifice completeness to efficiently handle the challenges posed by large, hard problems.

Hence, we investigate the applicability of a CP-based heuristic method in pMDs, similar to the one proposed in Ploskas et al. (2023) for p-dispersion problems with distance constraints. This method employs a CP solver that incorporates a heuristic pruning technique, which tries to prune the search tree by reasoning, at any node, about the best cost that can be achieved. Specifically, after the first solution has been found, a greedy heuristic is run at each visited node to estimate the best cost that can be achieved if the subtree rooted at that node is explored. If this estimated cost is not better than the cost of the best solution found so far, then the current branch is cut off.

We also introduce four specialized value ordering heuristics that try to intelligently guide search by selecting appropriate sites to place the facilities using increasingly more complex information. The first two heuristics only exploit local information about the facility that we currently try to assign to a site. The third heuristic takes into account information about past assignments when making the decision, while the fourth heuristic, in addition, reasons about the possible future assignments.

We experimented with pMD problems of three types. In the first two, the clients and the potential facility sites are randomly placed in a grid, while in the third, we use the p-median benchmark instances (Beasley, 1985) as a basis to generate pMDs. Problems of the first two types, which typically have few solutions, can be very hard for Gurobi, as it often does not discover any solution within 1 h of CPU time, but mostly manageable by OR-Tools, except for the larger ones. In contrast, OR-Tools fares badly on problems of the third type, which typically has many solutions, while Gurobi finds most of these problems quite easy.

The heuristic CP solver's performance is more stable across problem classes with different characteristics compared to standard ILP and CP solvers, despite not being the best in all classes. Specifically, it is by far more efficient and effective than Gurobi in grid-structured problems, and it also often outperforms OR-Tools, both in terms of run times and solution quality, being able to handle large instances of these types that are out of reach for the complete solvers. On the other hand, it is outperformed by Gurobi on most p-median-based instances (though it is quite efficient in some cases), but it is much

more efficient than OR-Tools. In addition, experiments with the four specialized value ordering heuristics demonstrated that through the use of these heuristics, and especially the more informed ones, we can often obtain improvements not only in solution quality, but also in run times.

Finally, as our heuristic pruning method is often quite aggressive, meaning that it can cut off many branches and reduce the size of the search space significantly, it may also result in the omission of optimal and near-optimal solutions. To partially alleviate this problem, we investigate the application of the heuristic together with a sampling technique that calculates the best estimation over different orderings for the future variables (i.e., unassigned facilities). Experimental results show that through this technique, we find better solutions, albeit by paying a run-time penalty.

This paper is an extended version of a conference paper by Iosif et al. (2024a), where the p-median problem with distance constraints was studied. Here, we extend that paper in the following ways:

- We describe our approach in more detail.
- We provide more extensive experimental results, offering deeper insights into the details and the performance of the CP heuristic solver.
- We give additional experimental results to assess the impact of the sampling technique proposed in Iosif et al. (2024a).
- And most importantly, we introduce four specialized value-ordering heuristics for the pMD and evaluate their effectiveness within the CP heuristic solver framework.

The remainder of this paper is structured as follows: In Section 2, we discuss related work. In Section 3, we give the necessary background on CP and ILP, and we formally define the pMD problem. In Sections 4 and 5, we present the ILP and CP formulations for pMDs, respectively. Section 6 details the proposed heuristic methodology, while Section 7 presents experimental results. Finally, Section 8 concludes the paper with a discussion and potential directions for future work.

## 2 Related Work

In the past few decades, a vast literature encompassing different models, problem characteristics, applications, algorithms, and heuristics for the various facility location problems, has been accumulated. The p-median problem is a prime example of a problem with “pull” objectives, where clients wish to have the facilities (e.g., pharmacies and stores) located close to them, whereas the p-dispersion problem is an example of a problem with “push” objectives, where we seek to place the facilities away from each other. In between we have the class of semi-obnoxious location problems where the facilities have both desirable and undesirable properties (Krarup et al., 2002). In this case, clients wish to have facilities located close to them, but not too close. For example, the residents of a suburb wish to have bars situated near them, but not too close because of the noise and traffic associated with such facilities.

The semi-obnoxious case can be viewed as a biobjective optimization problem. For instance, we might wish to minimize the total service cost and at the same time maximize the minimum distance between any facility and a client. Several strategies to handle such problems have been studied (Krarup et al., 2002). The problem can be treated directly as biobjective and we may try to find the Pareto set, which is anything but easy for location problems with multiple facilities (Ortigosa et al., 2015; Tutunchi & Fathi, 2019; Yapicioglu et al., 2007). As an alternative, a model with a single criterion can be derived, where the objective function is a weighted combination of both the pull and the push objective. Another option is to place a bound on the obnoxious effects, as a set of constraints, and hence to view the problem as having a single objective (Carrizosa & Plastria, 1999). This is the approach that we follow in this paper.

The p-median problem, which is  $\mathcal{NP}$ -hard on general networks for an arbitrary p, was originally proposed by Hakimi (1964, 1965). In 1970, ReVelle and Swain (1970) presented the first integer programming formulation for the p-median problem, utilizing a structure proposed by Balinski (1965) in a plant location problem. Thereafter, various alternative formulations have been proposed (Church, 2003; Cornuejols et al., 1980; Densham & Rushton, 1992; Rosing et al., 1979). For a thorough literature review on formulations of the p-median problem, see Church (2003).

Apart from the standard ILP approach, other methods, such as Lagrangian relaxation (Beltran et al., 2006) and numerous other heuristics, have been proposed for the p-median problem. The *greedy* or *myopic* heuristic first solves the 1-median problem by locating one facility in such a way as to minimize the total cost for all demand nodes. Facilities are then added one by one in the same way, that is, trying to minimize the total cost at each step, until all p facilities have been located (Kuehn & Hamburger, 1963). A survey of heuristic and meta-heuristic methods for the p-median problem can be found in Mladenović et al. (2007).

## 2.1 Distance Constraints

There are several early works that considered maximum distance constraints between the demand nodes and the facility locations (Church & Meadows, 1977; Chvatal, 1979; Khumawala, 1973; Tansel et al., 1982). Tansel et al. (1982) were the first to include distance constraints in a facility location problem. Specifically, they studied the distance-constrained p-center problem. The distance constraints that they consider are between the demand points and the centers. They study the case where the network is a tree and they also provide the dual of this problem. They give polynomially bounded procedures for solving both problems.

Moon and Chaudhry (1984) were the first to systematically study location problems with distance constraints. They provided a thorough categorization of such problems according to several factors, such as the optimization criterion, the type of distance constraint ( $>$  or  $<$ ), and whether constraints exist between facilities, or between facilities and clients, or between both. In the terminology they introduced, the pMD is called Median/ $l/l$ . That is, we seek to minimize the sum of the distances between the clients and their closest facility (as in p-median) subject to constraints that impose lower bounds (hence the  $l$ ) in the distance between any two facilities and also between any facility and any client. But whereas in Median/ $l/l$  the minimum distances in the distance constraints are uniform, denoting that the facilities are homogeneous, in the pMD they may differ from constraint to constraint because we consider the more general case of heterogeneous facilities.

Chaudhry et al. (1986) proposed heuristics for selecting a maximum-weight set of locations such that no two are closer than a given distance from each other. Moon and Papayanopoulos (1991) considered the problem of locating two facilities so as to minimize the maximum of combined Euclidean distances to unweighted existing points when the facilities must be separated by at least a specified distance. An interactive graphical method that produces near-optimal solutions was proposed. Comley (1995) studied the problem of locating a small number (up to three) of heterogeneous semi-obnoxious facilities that interact with each other as well as with other existing facilities. The minimum Euclidean distance between any pair of such facilities was maximized using a quadratic 0–1 programming algorithm.

Berman and Huang (2008) studied the problem of locating homogeneous obnoxious facilities on a network so as to minimize the total demand covered, subject to the condition that no two facilities are allowed to be closer than a prespecified distance. Drezner et al. (2018) proposed the Weber obnoxious facility location problem where we seek to locate one facility so that the weighted sum of distances between the facility and demand points is minimized, with the additional requirement that the facility location is at least a given distance from demand points because it is obnoxious to them. Drezner et al. (2019) considered a continuous multiple obnoxious facility location problem where a given number of facilities must be located in a convex polygon with the objective of maximizing the minimum distance between facilities and a given set of communities subject to distance constraints between facilities.

Welch and Salhi (1997) studied the location of obnoxious facilities with interactions between them. Location problems with distance constraints that restrict the placement of facilities near certain demand points have also been studied (e.g., Brimberg & Juel, 1998; Maier & Hamacher, 2019; Orloff, 1977).

Although CP has been successfully applied to many combinatorial problems, there are very few works investigating CP-related methods for facility location problems (Cambazard et al., 2012; Fazel-Zarandi & Beck, 2009; Ploskas et al., 2023; Sorkhabi et al., 2018). But very recently, ILP and CP models, as well as a CP-based heuristic method, were proposed for the p-dispersion problem with distance constraints (Iosif et al., 2024b; Ploskas et al., 2023).

## 3 Background and Problem Definition

We first give some necessary background on CP and ILP and then we formally define the pMD problem.

### 3.1 Constraint Programming (CP)

CP is a paradigm for solving combinatorial problems that integrates a wide range of techniques, mainly from AI and OR. In CP, the user states the constraints on the feasible solutions for a set of decision variables, and a solver is then used to deliver a feasible solution, or the optimal one according to some criterion. In a constraint satisfaction problem (CSP)  $P = (X, \text{Dom}, C)$ , we have:

- A set of  $n$  variables  $X = \{x_1, \dots, x_n\}$ .
- A set  $\text{Dom} = \{\text{Dom}(x_1), \dots, \text{Dom}(x_n)\}$  of finite domains, one for each variable in  $X$ .
- A set  $C = \{c_1, \dots, c_e\}$  of  $e$  hard constraints that must be necessarily satisfied. Each constraint  $c_i \in C$  involves a subset of  $X$ , known as the *scope* of  $c_i$ , and specifies the combinations of values that the variables in this subset are allowed to take.

In a binary CSP, any constraint involves at most two variables. If in addition to the hard constraints, there is an objective function to be optimized then we refer to the problem as a constraint satisfaction and optimization problem (CSOP). In many cases, in addition to the decision variables, the model of a CSP, or a CSOP, may include extra (auxiliary) variables that are useful for modeling purposes. Similarly, a model may include redundant constraints, that is, constraints whose removal does not change the set of solutions.

A *solution* to a CSP is a *complete assignment*, that is, an assignment involving all variables, that satisfies all constraints in  $C$ . A *consistent partial assignment* is an assignment to a set  $S \subseteq \mathcal{X}$  of variables that satisfies all constraints among the variables in  $S$ . The search process can be visualized by a traversal of a search tree where the root corresponds to the empty assignment (no decision variable has been assigned yet) and the rest of the nodes correspond to partial assignments.

Complete algorithms for CSPs are based on backtracking depth-first search interleaved with constraint propagation. Backtracking search tries to assign the decision variables with values from their domains so that constraints are satisfied, while constraint propagation filters values from domains that are deemed inconsistent according to some local consistency property that is applied to the constraints.

It is well known that one of the cornerstones of CP and a major reason for its success in various domains is constraint propagation (Bessiere, 2006). CP solvers typically propagate binary constraints using the local consistency property called arc consistency. For binary problems, a value  $a_i \in \text{Dom}(x_i)$  is arc consistent (AC) iff for every constraint between  $x_i$  and another variable  $x_j$ , there exists a value  $a_j \in \text{Dom}(x_j)$  s.t. the pair of values  $(a_i, a_j)$  satisfies the constraint (Mackworth, 1977). In this case, we say that  $a_j$  is a *support* for  $a_i$  in  $\text{Dom}(x_j)$ , and vice versa. In case a value  $a_i \in \text{Dom}(x_i)$  has no support in some domain  $\text{Dom}(x_j)$ , the application of AC will filter it from  $\text{Dom}(x_i)$ . The backtracking search algorithm that applies AC after each branching decision during a search is known as maintaining arc consistency (Sabin & Freuder, 1994). Unary constraints that involve only one variable (e.g.,  $x > 0$ ) are typically handled by applying the property of node consistency. A value  $a_i \in \text{Dom}(x_i)$  is node consistent (NC) iff it satisfies all the unary constraints on  $x_i$ . Values that are not NC can be detected and removed from the corresponding domains by iterating over all the variable domains in a preprocessing phase.

CP solvers also offer an array of *global constraints* that are very useful at the modeling level, as their use results in more compact models, but also speed up search as solvers typically provide fast specialized propagation algorithms for such constraints. A global constraint can capture a relation between an arbitrary number of variables. A well-known such constraint is the  $\text{AllDifferent}(x_1, \dots, x_n)$ , which specifies that all variables in the scope must be assigned different values (i.e., pairwise distinct). Another frequently used global constraint is the  $\text{Element}(\text{index}, \text{table}, \text{value})$  global constraint, which is used to link the discrete decision variable *index* and the variable *value* according to a given table of values *table*. Hence, *value* is equal to the  $\text{index}^{\text{th}}$  item of *table*, that is,  $\text{value} = \text{table}[\text{index}]$ .

Search in a CP solver is guided by variable and value ordering heuristics. After a branching decision (e.g., a variable assignment or a value removal from a domain) propagation kicks off. If this results in an empty domain, in which case we have a domain wipe-out (DWO), the search algorithm rejects the most recent branching decision and moves on to the next one. A standard general-purpose variable ordering heuristic is dom/weighted degree (wdeg) (Boussemart et al., 2004). This heuristic assigns a weight to each constraint, initially set to one. Each time a constraint causes a DWO, its weight is incremented by one. Each variable is associated with a wdeg, which is the sum of the weights over all constraints involving the variable and at least another (unassigned) variable. The dom/wdeg heuristic chooses the variable with a minimum ratio of current domain size to a wdeg.

Although value ordering heuristics have received less attention than variable ordering ones, as their effect on search effort is usually less profound in CSPs, they can be quite effective in CSOPs. The task of a value ordering heuristic is to choose a value for the current variable under consideration during a search. The guiding principle when designing value-ordering heuristics is to select the value most likely to succeed, that is, be part of a solution. There is a wide variety of such heuristics and they can be divided into static and dynamic (Rossi et al., 2006). Static heuristics produce a fixed ordering of values before the search begins, while dynamic ones determine the order as the search progresses. Static heuristics are generally computationally inexpensive, whereas dynamic heuristics guide the search using a more informed strategy, but are typically quite expensive to compute. Hence, the static lexicographic ordering of values is very commonly used. Like other types of heuristics, value ordering heuristics can be general-purpose or domain-specific. General-purpose heuristics use generic properties that CSPs share in common, while domain-specific ones incorporate expertise and deep knowledge of the problem at hand.

When faced with a CSOP, CP solvers typically add a bounding constraint when locating a solution, forcing any subsequent solution to have a better value according to the optimization function compared to the solution just found. This constraint is propagated, potentially resulting in domain pruning. If the filtering achieved is weak then the search in a CSOP degenerates to almost an exhaustive enumeration of the solutions. In contrast, if the filtering is strong then a CP solver may exhibit very good performance on a CSOP.

### 3.2 Integer Linear Programming (ILP)

Another paradigm for solving combinatorial problems is ILP. ILP is widely used in various fields, such as OR, to find optimal solutions where all or some (mixed-integer) of the decision variables are integers. This requirement often arises in practical applications where decisions involve discrete choices, such as assigning tasks to machines, scheduling shifts, or location problems, as in the case of this paper. ILP is a type of optimization paradigm where the objective function and the constraints are linear. ILP formulations are mathematical models, which can then be solved using specialized algorithms, such as branch and bound, cutting planes, and branch and cut.

An ILP problem is generally formulated as follows:

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \mathbf{Ax} \leq \mathbf{b} \\ & \mathbf{A}_{\text{eq}} \mathbf{x} = \mathbf{b}_{\text{eq}} \\ & \mathbf{x} \in \mathbb{Z}^n \end{aligned}$$

where  $m$  is the number of inequality constraints,  $k$  is the number of equality constraints,  $n$  is the number of decision variables,  $\mathbf{x} \in \mathbb{Z}^n$  is an  $n$ -dimensional vector of decision variables,  $\mathbf{c}$  is an  $n$ -dimensional vector of coefficients,  $\mathbf{A}$  is an  $m \times n$  matrix of coefficients for the inequality constraints,  $\mathbf{b}$  is an  $m$ -dimensional vector of constants for the inequality constraints, while  $\mathbf{A}_{\text{eq}}$  is a  $k \times n$  matrix of coefficients for the equality constraints and  $\mathbf{b}_{\text{eq}}$  is a  $k$ -dimensional vector of constants for the equality constraints. The decision variables are also subject to lower and upper bounds:  $\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$ , where  $\mathbf{l}$  is an  $n$ -dimensional vector of lower bounds and  $\mathbf{u}$  is an  $n$ -dimensional vector of upper bounds.

Binary variables are a special case of integer variables in ILP, where the variables are restricted to take values in the set  $\{0, 1\}$ . These variables are commonly used in ILP formulations to model decisions that have a yes/no nature.

Solving ILP problems involves specialized algorithms and software tools known as solvers. These solvers are designed to handle the complexities associated with the discrete nature of ILP problems and provide optimal or near-optimal solutions efficiently.

### 3.3 The $p$ -Median Problem With Distance Constraints

In a  $p$ -median problem with distance constraints (pMD),  $p$  facilities in a set of facilities  $F$  (i.e.,  $p = |F|$ ) are to be placed on  $p$  nodes of a weighted network  $G = \{V, E\}$  embedded on a plane. A set  $CL$  of demand points (clients) to be serviced by the facilities is already located in the network. In the following, we will use the terms demand points and clients interchangeably. We assume that the set  $P$  of nodes in  $G$  where facilities can potentially be located is known, and so are the nodes where demand points are located. Hence, we deal with a discrete/network location problem. The weight  $w_{ij}$  on an edge  $(i, j) \in E$  denotes the symmetric service cost (typically the distance) between nodes  $i$  and  $j$ . That is, the cost of serving a client located at node  $i$  assuming a facility is located at node  $j$ , and vice versa.

Between each pair of facilities  $f_i$  and  $f_j$  there is a distance constraint  $\text{dis}(f_i, f_j) > d1_{ij}$  specifying that the (Euclidean) distance  $\text{dis}(f_i, f_j)$  between the nodes where the facilities  $f_i$  and  $f_j$  are located must be greater than  $d1_{ij}$ , where  $d1_{ij}$  is a real constant. Also, between each facility  $f_i$  and any client  $c_k$  there is a distance constraint  $\text{dis}(f_i, c_k) > d2_i$  specifying that the (Euclidean) distance  $\text{dis}(f_i, c_k)$  between the node where facility  $f_i$  is located and node  $k$  where client  $c_k$  is located must be greater than  $d2_i$ , where  $d2_i$  is a real constant.

The Euclidean distance between two nodes  $i$  and  $j$  should not be confused with the minimum distance between the two nodes in the network, which is given by the sum of weights of the edges that belong to the shortest path between  $i$  and  $j$ . It is natural to model the costs of service between  $i$  and  $j$  through the cost of the paths between the two nodes, but we believe that any distance constraint established due to, say, safety reasons, should consider a metric such as the Euclidean distance instead.

A common assumption in the literature on location problems with distance constraints is that the lower distance bound  $d1_{ij}$  is fixed to a specific value for all the constraints between facilities, and so is  $d2_i$  for constraints between facilities and demand points. This is a reasonable assumption in the case where the facilities are homogeneous, and therefore in essence indistinguishable, but it is not always realistic, especially when the facilities have different properties. In this paper, we deal with the heterogeneous case where the distance bound for the distance constraints between facilities may vary from constraint to constraint. However, we assume that for any specific facility  $f_i$ , the distance constraints between  $f_i$  and all demand nodes have the same bound.

In the following, as is common in the literature, we assume that the pairwise distances between all nodes in the network  $G$  have been precomputed and are stored in two 2-d matrices,  $\mathbf{D}$  and  $\mathbf{SP}$  that are given as input to the algorithms. The first stores the Euclidean distances and the second stores the lengths of the shortest paths. Hence,  $D[i, j]$  is the Euclidean distance between points  $i$  and  $j$  while  $SP[i, j]$  is the length of the shortest path between the same points. These assumptions are not restrictive as other metrics can be used to capture the distances between points.

To summarize our notation, we have:

- $CL$ : set of demand nodes.
- $P$ : set of candidate facility sites.
- $F$ : set of facilities.
- $p$ : the number of facilities to be located.
- $D[i, j]$ : the Euclidean distance between any two nodes.
- $SP[i, j]$ : the shortest path distance between any two nodes.
- $d1_{ij}$ : the lower bound in the allowed distance between each pair of facilities  $(i, j)$ , where  $i \in F$  and  $j \in F$ .
- $d2_i$ : the lower bound in the allowed distance between each facility  $i \in F$  and all demand nodes.

The goal is to minimize the sum of distances between each demand point and its nearest located facility, subject to the satisfaction of all the distance constraints.

## 4 ILP Formulation

In this section, we present a binary ILP model for pMD. The model is based on the formulation of ReVelle and Swain (1970) for the p-median problem, extended to the case of heterogeneous facilities with distance constraints. Although alternative formulations for the p-median problem exist, it is accepted that ReVelle and Swain's (1970) formulation exhibits integer-friendly properties, and a recent study by Ploskas and Stergiou (2022) demonstrated that it is the most efficient option when modeling the p-median problem with homogeneous distance constraints.

Dealing with heterogeneous facilities introduces the need for additional decision variables, to know which specific facility is located in which candidate location, whereas in ReVelle and Swain (1970), and any other p-median formulation, we only need to know if a site hosts a facility or not. The introduction of new decision variables necessitates the generalization of the constraints that forbid the location of two facilities in the same place and the location of a single facility in multiple places. Also, compared to ReVelle and Swain's (1970) formulation, in the pMD model there exist two additional sets of constraints to capture the distance constraints existing between facilities and between facilities and clients. In the following, we make use of the following additional notation:

- $C = \{(i, j, f_1, f_2) | i, j \in P, f_1, f_2 \in F, D[i, j] \leq d1_{f_1 f_2}\}, \forall i \in P, \forall j \in P$ , and for each pair of facilities  $(f_1, f_2)$ : the set of quadruples  $(i, j, f_1, f_2)$  s.t. facilities  $f_1$  and  $f_2$  cannot be placed in facility sites  $i$  and  $j$ , respectively, because  $i$  and  $j$  are not in a safe distance between each other with respect to the allowed distance between  $f_1$  and  $f_2$ .
- $N = \{(i, j) | i \in P, j \in F, \exists k \in CL, D[i, k] \leq d2_j\}, \forall i \in P, \forall j \in F$ : the set of pairs  $(i, j)$  s.t. facility  $j$  cannot be placed in facility site  $i$  because there exists a demand node  $k$  that is not in safe distance from  $i$  with respect to the allowed distance between  $j$  and the demand nodes.
- $x_{ij} = 1$  if a facility  $j \in F$  is located at a facility site  $i \in P$  and 0 otherwise.
- $y_{ij} = 1$  if a demand node  $i \in CL$  is assigned to a facility site  $j \in P$  and 0 otherwise.

As we deal with heterogeneous facilities, we need  $|P| \times |F|$  variables, that is, one variable  $x_{ij}, \forall (i, j), i \in P, j \in F$ , in order to know whether or not a specific facility  $j \in F$  is located at a facility site  $i \in P$ . In addition, we need  $|CL| \times |P|$  variables, that is,  $y_{ij}, \forall (i, j), i \in CL, j \in P$ , in order to know whether or not a demand node  $i \in CL$  is assigned to a facility site  $j \in P$ . Variables  $y_{ij}, \forall (i, j), i \in CL, j \in P$ , are required in order to: (i) calculate the distance between each demand node and the facility that serves it (in the objective function of the model), and (ii) place restrictions on which facilities can serve each demand node based on the distance constraints.

The model for pMDs can be expressed as:

$$\min \sum_{i \in CL} \sum_{j \in P} SP[i, j] \times y_{ij} \quad (1)$$

$$\text{s.t.} \quad \sum_{j \in F} x_{ij} \leq 1, \forall i \in P \quad (2)$$

$$\sum_{i \in P} x_{ij} = 1, \forall j \in F \quad (3)$$

$$\sum_{i \in P} \sum_{j \in F} x_{ij} = p \quad (4)$$

$$\sum_{j \in P} y_{ij} = 1 \quad \forall i \in CL \quad (5)$$

$$y_{ij} \leq \sum_{k \in F} x_{jk} \quad \forall i \in CL, \forall j \in P \quad (6)$$

$$x_{if_1} + x_{jf_2} \leq 1, \forall (i, j, f_1, f_2) \in C \quad (7)$$

$$x_{ij} = 0, \forall (i, j) \in N \quad (8)$$

$$x_{ij} \in \{0, 1\}, \forall i \in P, \forall j \in F \quad (9)$$

$$y_{ij} \in \{0, 1\}, \forall i \in CL, \forall j \in P \quad (10)$$

The objective function 1 aims at minimizing the sum of the distances between the clients and their nearest located facility. Constraint 2 ensures that each facility site can host at most one facility, while Constraint 3 guarantees that each facility should be hosted at exactly one facility site. Constraint 4 specifies that  $p$  facilities are to be located. Constraint 5 ensures that each demand node will be served by one facility site, while Constraint 6 guarantees that each demand node will be served by a facility site that indeed hosts a facility. It is a generalization of the Balinski constraint (Balinski, 1965) to the heterogeneous case.

Constraint 7 models the distance constraints between facilities. It ensures that each facility is at a safe distance from all other facilities by not allowing two facilities  $f_1$  and  $f_2$  to be established at sites that are at a distance closer than the allowed distance between  $f_1$  and  $f_2$ . This constraint generalizes to the heterogeneous case the simplest and most efficient formulation of distance constraints given by Berman and Huang (2008) for the minimum weighted covering location problem, where distance constraints are homogeneous.

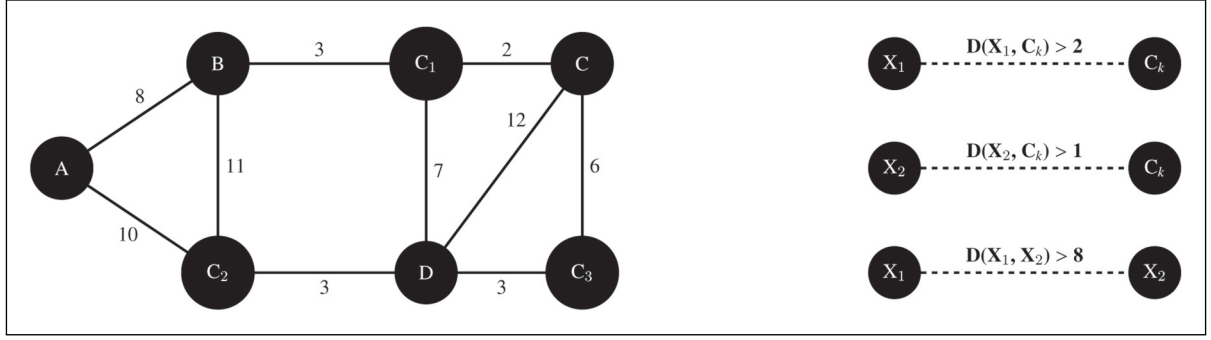
Finally, Constraint 8 is the distance constraint between facilities and demand nodes. This constraint ensures that any facility site that is opened is at a safe distance from every demand node. Instead of setting variables  $x_{ij}$ , where  $(i, j) \in N$ , equal to 0, these variables can be removed from the model.

## 5 CP Formulation

The pMD is modeled as a CSOP  $(X, \text{Dom}, C, O)$ , where  $X$  is the set of decision variables,  $\text{Dom}$  is the set of finite domains,  $C$  is the set of hard constraints, and  $O$  is the optimization function. The model is as follows:

- (1) For each facility  $i \in F$  there is a finite domain variable  $x_i$ . These  $p$  variables are the decision variables in the problem, meaning that  $|X| = |F| = p$ . For each variable  $x_i \in X$ ,  $\text{Dom}(x_i)$  includes as values all the points where a facility can be located, that is,  $\forall x_i \in X : \text{Dom}(x_i) = P$ .
- (2)  $Y1$  is a set of auxiliary variables, s.t. for each pair of variables  $(x_i, x_j) \in X \times X \mid i < j$ , there is a variable  $y1_{ij} \in Y1$  and a constraint  $y1_{ij} = D[x_i, x_j]$  (slightly abusing the notation, we write  $D[x_i, x_j]$  to refer to the distance between the values that  $x_i$  and  $x_j$  are assigned to). Hence, each  $y1_{ij} \in Y1$  models the Euclidean distance between  $x_i$  and  $x_j$ . In CP solvers such as OR-Tools, the constraint  $y1_{ij} = D[x_i, x_j]$  can be implemented using the element global constraint, that is,  $y1_{ij} = \text{Element}(D, [x_i, x_j])$ .
- (3)  $Y2$  is a set of auxiliary variables, s.t. for each pair of facilities and clients  $(x_i, c_k) \in X \times CL$ , there is a variable  $y2_{ik} \in Y2$  and a constraint  $y2_{ik} = D[x_i, c_k]$ . Hence, each  $y2_{ik} \in Y2$  models the Euclidean distance between  $x_i$  and  $c_k$ . This can also be implemented using the Element constraint, that is,  $y2_{ik} = \text{Element}(D, [x_i, c_k])$ .
- (4)  $S$  is a set of auxiliary variables, s.t. for each pair of variables and clients  $(x_i, c_k) \in X \times CL$ , there is a variable  $s_{ik} \in S$  and a constraint  $s_{ik} = SP[x_i, c_k]$ . Hence, each  $s_{ik} \in S$  models the service cost (i.e., shortest path distance) between  $x_i$  and  $c_k$ . Again, this can be implemented using the Element constraint, that is,  $s_{ik} = \text{Element}(SP, [x_i, c_k])$ .





**Figure 1.** Illustration of the p-median with Distance Constraints (pMD) Problem from Example 1. On the Left, an Undirected Graph of the Problem is Given. The Larger Nodes Represent Demand Nodes (e.g., Node  $C_1$ ), Whereas Smaller Nodes Denote the Candidate Location Points Where the Two Facilities Could be Placed (e.g., Node A). Weights on the Edges of the Graph (Solid Lines) Denote the Distance Between Pairs of Nodes. On the right, Euclidean Distance Constraints Between the two Facilities ( $D[X_1, X_2] > 8$ ) and Between Each Facility and all Clients (e.g.,  $D[X_1, C_k] > 2, \forall k \in |CL|$ ) are Demonstrated.

- (5)  $Z$  is a set of auxiliary variables, s.t. for each client  $c_k \in CL$ , there is a variable  $z_k \in Z$  and a constraint  $z_k = \min(s_{1k}, s_{2k}, \dots, s_{pk})$ . Hence, each  $z_k \in Z$  models the shortest path distance between each client and its nearest facility.
- (6) For each variable  $y_{1ij} \in Y_1$ , there is a distance constraint  $y_{1ij} > d_{1ij}$ .
- (7) For each variable  $y_{2ik} \in Y_2$ , there is a distance constraint  $y_{2ik} > d_{2ik}$ .
- (8) There is a variable  $z$ , s.t.  $z = \text{sum}(Z)$ , capturing the sum of the shortest path distances between clients and their closest facility.
- (9) The objective function is  $O = \text{minimize}(z)$ .

The  $s_{ik} = \text{SP}[x_i, c_k]$  constraints link the auxiliary variables  $s_{ik}$ , and therefore also the  $z$  variable and the objective function, with the decision variables. This is because the value of  $s_{ik}$  is computed by accessing the matrix **SP** using as indices the location of  $c_k$ , which is fixed, and the value (i.e., location) of  $x_i$ . The bounding constraint that will be added by a CP solver once a solution with cost `cur_cost`, that is better than all previous ones is found, will be of the form  $z < \text{cur\_cost}$ . The propagation of this constraint may result in the domain pruning of the  $z_k$  variables, which in turn may force the pruning of the  $s_{ik}$  variables' domains, and in the end, through the  $s_{ik} = \text{SP}[x_i, c_k]$  constraints, this may result in the pruning of the decision variables' domains.

We also considered adding to the model an `AllDifferent` constraint over all variables in  $X$  to speed up propagation. Such a constraint is redundant, as the facility distance constraints already force the facilities to be placed at different locations. Experiments with and without the `AllDifferent` showed no noticeable difference in run times, and therefore, we do not consider this option any further.

Let us now give an example of a small pMD to illustrate the CP model.

**Example 1.** There are two facilities to be located (i.e.,  $p = |X| = 2$ ), four potential facility sites ( $|P| = 4$ ) and three clients ( $|CL| = 3$ ). The potential facility sites (nodes A, B, C, and D) and the clients are located on a graph, as shown in Figure 1. There is a binary distance constraint  $D[x_1, x_2] > d_{12}$  between the two variables  $x_1$  and  $x_2$  that model the facilities, where  $d_{12} = 8$ . There are also six unary distance constraints  $D[x_i, c_k] > d_{2i}, \forall i \in |X|, \forall k \in |CL|$  that impose bounds on the minimum allowed distances between facilities and clients, as shown in Figure 1. The CP model is formulated as follows:

- **Variables (19 in total)**

- *Decision variables:*

$x_1, x_2$ , with  $\text{Dom}(x_1) = \text{Dom}(x_2) = P = \{A, B, C, D\}$ .

- *Auxiliary variables:*

$y_{112}$ , to model the Euclidean distance between  $x_1$  and  $x_2$ .

$y_{2ik}$  ( $i \in \{1, 2\}, k \in \{1, \dots, 3\}$ ), to model the Euclidean distance between the two facilities and the three clients.

$s_{ik}$  ( $i \in \{1, 2\}, k \in \{1, \dots, 3\}$ ), to model the shortest path distances between facilities and clients.

$z_1, z_2, z_3$ , to facilitate the modeling of the shortest path distance between each client and its nearest facility.

$z$ , to model the objective function.

- **Constraints (24 in total)**
  - $\text{Element}(D, [x_1, x_2], y1_{12})$
  - $\text{Element}(D, [x_i, c_k], y2_{ik}), \forall i \in |X|, \forall k \in |CL|$
  - $\text{Element}(\text{SP}, [x_i, c_k], s_{ik}), \forall i \in |X|, \forall k \in |CL|$
  - $y1_{12} > 8$
  - $y2_{1k} > 2, \forall k \in |CL|$
  - $y2_{2k} > 1, \forall k \in |CL|$
  - $z_k = \min(s_{1k}, s_{2k}), \forall k \in |CL|$
  - $z = \sum(z_k), \forall k \in |CL|$
- **Objective function**
  - $\text{minimize } z$

## 6 A Heuristic CP Approach to the pMD

Ploskas et al. (2023) proposed a heuristic technique for the p-dispersion problem with distance constraints that tries to prune early the parts of the search tree for which it seems unlikely that their exploration will improve the value of the optimization function. Specifically, the cost of the first feasible solution found is used as the initial lower bound denoting the cost of the best solution found so far. Thereafter, at each node of the search tree, after the currently tried assignment,  $x_i = a$  has been propagated, and assuming no failure occurs, an upper bound for the best possible solution under the current assignment is computed, giving an estimation of the best possible cost that can be achieved if the subtree rooted at the specific node is explored. If this is not higher than the current lower bound then the current branch of the search tree is abandoned and the search moves on. Each time a solution with a higher cost than the current lower bound is found, the lower bound is updated.

We continue this work by adapting this method to pMDs. In our case, the cost of the first feasible solution found is used as the initial upper bound since we have a minimization problem. Besides the bounding technique, our heuristic method uses a simpler model of the problem. As our experiments demonstrate, a reason for the failure of OR-Tools to solve large instances of the pMD is the size of the model it constructs, mainly because of the very large domains, along with a large number of auxiliary variables. However, in practice, it is not uncommon for location problems to include a very large number of potential facility sites (hence very large domains in the CP model).

To alleviate this problem, we propose to use a much simpler model (albeit losing propagation power), dropping all the auxiliary variables and relevant constraints, resulting in a model with only the p-decision variables and the distance constraints among pairs of variables, and between variables and clients. The optimization function can now be handled procedurally within the solver by simply computing the cost of any new solution found, so as to determine if this cost is better than the cost of the best solution discovered so far. If so, then the bound is tightened. Considering the small pMD demonstrated in Example 1, the simpler model is as follows:

- **Decision variables (2 in total)**
  - $x_1, x_2$ , with  $\text{Dom}(x_1) = \text{Dom}(x_2) = P = \{A, B, C, D\}$ .
- **Constraints (7 in total)**
  - $D[x_1, x_2] > 8$
  - $D[x_1, c_k] > 2, \forall k \in |CL|$
  - $D[x_2, c_k] > 1, \forall k \in |CL|$
- **Objective function**
  - $\min \sum_{k \in |CL|} \min_{i \in |X|} \text{SP}[x_i, c_k]$

This simple model has 17 fewer variables and 17 fewer constraints compared to the model of Example 1, as the auxiliary variables and the relevant Element constraints are not present. As a downside, propagation is weakened because the objective function is no longer linked to the decision variables. Therefore, as we also discuss below, any tightening of the bound cannot result in the pruning of the decision variables' domains.

In the following, we first describe the operation of a CP solver that can use the branch pruning heuristic, and then we detail the inner workings of the heuristic.

## 6.1 Search Framework

Algorithm 1 gives a high-level description of the solver in which the pruning heuristic has been implemented. Given a pMD  $(X, \text{Dom}, C, O)$ , the algorithm starts by propagating the hard constraints in  $C$ , that is, the distance constraints, as a typical CP solver does. Function *Propagate* enforces node consistency on the unary constraints between facilities and clients, and arc consistency on the binary constraints between facilities. As explained, these distance constraints are the only constraints present in the simple pMD model that we use. If no failure (empty domain) is detected then the algorithm initializes the depth to 1, it sets the best cost found (*best\_found*) to a sufficiently large value, and commences the search by selecting a variable using a variable ordering heuristic. While the depth of search is  $>0$ , denoting that the search space has not been exhaustively searched, a branching decision is made, that is, a value is selected, using a value ordering heuristic, and assigned to the currently considered variable.

If all variables have been assigned ( $\text{depth} = n$ ), which means that a feasible solution has been found, the cost of this solution is calculated and if this cost is better than the best cost found up to that point, then the value of *best\_found* is updated accordingly. At this point, a standard CP solver would add a constraint of the form  $z < \text{best\_found}$  to the model, to ensure that any subsequently found solution will be better than the one just discovered. However, this cannot be done in our model, as there is no  $z$  variable. But in this way, we avoid the costly propagation involved. As a downside, while the search unravels, it is not guaranteed that any newly discovered solution will be better than the best discovered up to that point.

If not all variables have been assigned yet, function *Propagate* is called to propagate the value assignment that has just been made. If no failure occurs, the heuristic bounding mechanism is triggered by calling function *Bound\_Estimation* (Algorithm 2), provided that at least one feasible solution has already been found (i.e.,  $\text{sol\_found} = \text{TRUE}$ ). If this function succeeds, meaning that the estimated cost is better than the best bound found up to that point then the algorithm moves forward by increasing the depth of search and selecting a new unassigned variable. On the other hand, if propagation fails or the estimated bound is not better than the value of *best\_found* then the current branch is abandoned and a new value for the current variable is selected, as indicated by the value ordering heuristic. If all values for this variable have been tried at this depth of search, then the algorithm backtracks to the previously selected variable and tries a new value for it.

---

### Algorithm 1. CP\_Solver( $X, \text{Dom}, C, O$ )

---

```

if Propagate( $X, \text{Dom}, C$ ) = FALSE
    return NULL;
depth  $\leftarrow$  1;
best_found  $\leftarrow$   $+\infty$ ;
sol_found  $\leftarrow$  FALSE;
select an unassigned variable  $x_i$ ;
while depth  $\geq$  1
    if all values in  $\text{Dom}(x_i)$  have been tried
        depth  $\leftarrow$  depth-1;
    else
        select a value  $a \in \text{Dom}(x_i)$  that is indicated by the value ordering heuristic;
        if depth =  $n$ 
            sol_found  $\leftarrow$  TRUE;
            cur_cost  $\leftarrow$  Compute_Solution_Cost( $X, \text{Dom}, C$ )
            if cur_cost < best_found
                best_found  $\leftarrow$  cur_cost;
        else if Propagate( $X, \text{Dom}, C, x_i \leftarrow a$ ) = TRUE
            if sol_found = TRUE
                if Bound_Estimation( $X, \text{Dom}, x_i \leftarrow a, \text{best\_found}$ ) = TRUE
                    depth  $\leftarrow$  depth + 1;
                    select an unassigned variable  $x_i$ ;
            else
                depth  $\leftarrow$  depth+1;
                select an unassigned variable  $x_i$ ;
if sol_found = FALSE
    return NULL;
else
    return best_found;

```

---

If the value `best_found` remains unchanged upon termination (i.e., `sol_found = FALSE`), then the algorithm has proved that the problem is infeasible and the solver returns `NULL`. Otherwise, the best cost found is returned. In the former case, the heuristic part of the algorithm (i.e., the bounding mechanism) will never be triggered, as no feasible solution will have been found. Hence, the search space will be systematically explored in a typical CP solver fashion until a backtrack to depth 0 occurs, proving that the problem is infeasible.

## 6.2 Branch Pruning Heuristic

Let us now describe the function `Bound_Estimation` (Algorithm 2) that implements the branch pruning heuristic. As in Ploskas et al. (2023), at each node, we relax the problem by considering only the objective function (i.e., the distance constraints are not taken into account) and we heuristically estimate the cost that can be achieved if the subtree rooted at that node is explored. But in our case, a lower bound estimation is computed, as we deal with a minimization problem.

The estimation of this lower bound is performed using the reasoning of the greedy heuristic for the  $p$ -median problem. Specifically, assuming that  $x_i$  is the current variable,  $(x_1 \leftarrow v_1), \dots, (x_{i-1} \leftarrow v_{i-1})$  is the assignment to past variables (i.e., the already assigned ones) and  $v_i$  is the value under consideration for  $x_i$ , we greedily compute the cost of the “best” assignment for the future variables (i.e., the unassigned ones)  $x_{i+1}, \dots, x_p$ . That is, we visit these variables one by one, starting with  $x_{i+1}$ , and for each such variable  $x_j$ ,  $i + 1 \leq j \leq p$ , and each value  $v_j \in \text{Dom}(x_j)$ , we find the sum of the shortest path distances between all demand points and their nearest facility/variable, taking into account the previously assigned variables  $x_1, \dots, x_{j-1}$  and the assignment  $x_j \leftarrow v_j$ . After all values in  $\text{Dom}(x_j)$  have been processed, the value that minimizes this sum is then (temporarily) assigned to  $x_j$ . This is repeated until all variables have been assigned. If the cost of the derived complete assignment is equal to or higher than the current upper bound (i.e., the value of `best_found`) then  $v_i$  is not assigned to  $x_i$ . That is, we speculate that the exploration of the subtree rooted at node  $x_i \leftarrow v_i$  will not yield an improvement to the value of `best_found`.

---

**Algorithm 2.** `Bound_Estimation(X, Dom,  $x_i \leftarrow v_i$ , best_found)`

---

```

for each  $x_j, i + 1 \leq j \leq p$ 
   $\text{dis} \leftarrow \infty$ ;
   $\text{val} \leftarrow -1$ ;
  for each  $v_j \in \text{Dom}(x_j)$ 
     $x_j \leftarrow v_j$ ;
     $\text{temp-cost} \leftarrow$  sum of SP distances of all clients and their nearest facility among  $x_1, \dots, x_j$ ;
    if  $\text{temp-cost} < \text{dis}$ 
       $\text{dis} \leftarrow \text{temp-cost}$ ;
       $\text{val} \leftarrow v_j$ ;
   $x_j \leftarrow \text{val}$ ;
if  $\text{dis} < \text{best\_found}$ 
  return true;
else
  return false;

```

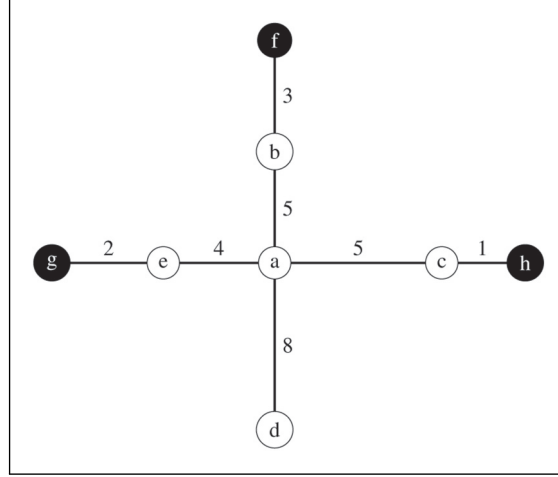
---

## 6.3 An Example of Erroneous Pruning and a Sampling Technique

The following example demonstrates that the greedy bound estimation that is performed at every node can overestimate the cost, which may result in the omission of the optimal solution.

**Example 2.** Consider a network  $G = \{V, E\}$  with  $V = \{a, b, c, d, e, f, g\}$  (Figure 2). Assume that three demand points are situated at nodes  $f, g, h$  and we wish to locate three facilities in the network, modeled by three variables  $x_1, x_2, x_3$ . Assume that there exist distance constraints between facilities and demand nodes and the domains of the variables after these constraints have been processed are:  $\text{Dom}(x_1) = \{a, b, c, d, e\}$ ,  $\text{Dom}(x_2) = \{a, b, d\}$  and  $\text{Dom}(x_3) = \{a, d\}$ . The distance constraints between facilities are:  $D[x_1, x_2] > 1, D[x_1, x_3] > 1, D[x_2, x_3] > 1$ . The labels on the edges denote the distance between the corresponding nodes, which for simplicity we assume is equal to the Euclidean distance between the two nodes.

The optimal solution is  $\langle x_1 = c, x_2 = b, x_3 = a \rangle$  with cost 10. Assume that the first solution found, with cost = 15, is  $\langle x_2 = a, x_3 = d, x_1 = c \rangle$ . Now, assume that as the search for better solutions unfolds, at some point the algorithm backtracks to the first level of the search tree undoes the decision  $x_2 = a$ , and then tries the assignment  $x_2 = b$ . A lower bound under this assignment will now be computed using the greedy heuristic. Assume that the heuristic



**Figure 2.** An Example p-Median with Distance Constraints (pMD) Problem where the Branch Pruning Heuristic Misses the Optimal Solution.

visits the remaining variables in lexicographic order (i.e., first  $x_1$  and then  $x_3$ ). Given the assignment  $x_2 = b$ , all three values  $a, c, e$  for  $x_1$  give the same maximum decrease in the value of the objective function (from 25 to 15). If it chooses according to the lexicographic order then it will select  $a$  for  $x_1$  and it will then select value  $d$  for  $x_3$ , as it is the only remaining option. The lower bound computed will have a value equal to 15, which is not lower than that of the best solution discovered so far. Hence, the branch rooted at the assignment  $x_2 = b$  will be abandoned and the search will move on, resulting in the omission of the optimal solution.

As the example demonstrates, an important drawback of the heuristic bound estimation procedure is that the bound computed at each node is based on a single run of the greedy heuristic with a fixed ordering of the unassigned variables. In the above example, if the order in which the greedy heuristic visits the future variables is reversed (i.e., first  $x_3$  and then  $x_1$ ) then the optimal solution will not be pruned and will be later discovered.

To address this, we enhanced our method through a technique that randomly samples over the possible orderings of the unassigned variables, computing a bound using the greedy heuristic for every sampled ordering. To minimize unnecessary computations, we set the number of samples equal to the number of all possible permutations (i.e., different variable orderings) if the latter is smaller than the former. If the estimated lower bound of a sample is lower than the cost of the best solution found so far, then the current branch is accepted. This results in weaker pruning, and therefore higher CPU times, but a better approximation of the optimal solution, as our experiments demonstrate. This technique (depicted by Algorithm 3) modifies the function `Bound_Estimation` by repeatedly randomly reordering the future variables and running the greedy heuristic on each obtained ordering.

---

**Algorithm 3.** `Bound_Estimation_with_Samplings( $X, \text{Dom}, x_i \leftarrow v_i, \text{best\_found}, \text{samples}$ )`

---

```

best_sample_cost  $\leftarrow \infty$ ;
while samples > 0
  shuffle future vars  $x_{i+1}, \dots, x_p$  in a random way;
  for each  $x_j, i+1 \leq j \leq p$ 
    dis  $\leftarrow \infty$ ;
    val  $\leftarrow -1$ ;
    for each  $v_j \in \text{Dom}(x_j)$ 
       $x_j \leftarrow v_j$ ;
      temp-cost  $\leftarrow$  sum of SP distances of all clients and their nearest facility among  $x_1, \dots, x_j$ ;
      if temp-cost < dis
        dis  $\leftarrow$  temp-cost;
        val  $\leftarrow v_j$ ;
       $x_j \leftarrow \text{val}$ ;
    if best_sample_cost > dis
      best_sample_cost  $\leftarrow$  dis;
    if best_sample_cost < best_found
      return true;
  samples  $\leftarrow$  samples - 1;
return false;

```

---

## 6.4 Value Ordering Heuristics

We now introduce four domain-specific value ordering heuristics for pMDs. These heuristics use information about the current state of search to select a value assignment for the current variable that may maximally contribute to the improvement of the objective value. The first two heuristics only consider the current variable and its available values, meaning that they only have a local view of the problem, but they are also cheap to compute. The third heuristic takes into account the already assigned variables too, whereas the fourth one, which is computationally the most expensive, additionally considers the possible assignments to future variables.

**6.4.1 Greedy-Minmax.** We call the first heuristic *greedy-minmax*. This value ordering heuristic selects the value that would induce the smallest cost in the objective function if we were to solve the 1-center problem. The p-center problem is similar to the p-median and differs only in the objective function. The goal is to determine the locations of  $p$  facilities in a way that minimizes the maximum distance between any client and their closest facility. The 1-center problem is a special case of the p-center, where the goal is to locate a single facility (i.e.,  $p = 1$ ).

Therefore, the site that will be selected for the location of the facility under consideration will be the one whose maximum distance from a client is the smallest among all potential sites. Specifically, suppose that we try to pick a value for the current variable  $x_i$ . For each value  $v_i \in \text{Dom}(x_i)$ , we calculate the maximum SP distance between  $v_i$  and any client. The value to be assigned to  $x_i$  is the one with the minimum such distance. Algorithm 4 depicts this process.

---

**Algorithm 4.** Greedy\_minmax( $X, \text{Dom}, x_i$ )

---

```

min-dist  $\leftarrow \infty$ ;
val  $\leftarrow -1$ ;
for each  $v_i \in \text{Dom}(x_i)$ 
  dist  $\leftarrow$  maximum SP distance between  $v_i$  and any client;
  if dist < min-dist
    min-dist  $\leftarrow$  dist;
    val  $\leftarrow v_i$ ;
return val;

```

---

**6.4.2 Greedy-Minsum.** The *greedy-minsum* value ordering heuristic also holds a local view of the problem. However, instead of solving the 1-center problem, it considers the 1-median. In other words, for each value  $v_i \in \text{Dom}(x_i)$ , we compute the sum of SP distances between  $v_i$  and all clients. The value to be assigned to  $x_i$  is the one with the minimum sum. Algorithm 5 depicts this process.

---

**Algorithm 5.** Greedy\_minsum( $X, \text{Dom}, x_i$ )

---

```

min-dist  $\leftarrow \infty$ ;
val  $\leftarrow -1$ ;
for each  $v_i \in \text{Dom}(x_i)$ 
  dist  $\leftarrow$  sum of SP distances between  $v_i$  and all clients;
  if dist < min-dist
    min-dist  $\leftarrow$  dist;
    val  $\leftarrow v_i$ ;
return val;

```

---

**6.4.3 Greedy Look-Back.** In contrast to the two previous value ordering heuristics, the third one, which we call *greedy look-back*, exploits information about past assignments. That is, while trying to find the best location for the current facility, it also considers the locations of the facilities that have already been placed. This leads to more precise estimations and therefore may result in more informed branching, as the location of the current facility may not alter the service status of a client (i.e., there is a facility already located closer to them).

Specifically, for each value  $v_i \in \text{Dom}(x_i)$ , we calculate the total cost of the assignment  $x_i \leftarrow v_i$ , taking into account the past assignments to variables  $x_1, \dots, x_{i-1}$  and the currently evaluated assignment  $x_i \leftarrow v_i$ . That is, we calculate the sum of distances between clients and their nearest facility, assuming that each client is assigned to the closest facility among the past ones and the value under consideration ( $v_i$ ) for the current variable ( $x_i$ ). The value selected is the one giving the minimum sum. Algorithm 6 depicts this process.

---

**Algorithm 6.** Greedy\_LookBack( $X, \text{Dom}, x_i$ )

---

```

min-cost  $\leftarrow \infty$ ;
val  $\leftarrow -1$ ;
for each  $v_i \in \text{Dom}(x_i)$ 
   $x_i \leftarrow v_i$ 
  temp-cost  $\leftarrow$  sum of SP distances of all clients and their nearest facility among  $x_1, \dots, x_i$ ;
  if dist < min-dist
    min-dist  $\leftarrow$  dist;
    val  $\leftarrow v_i$ ;
return val;

```

---

**6.4.4 Greedy Look-Ahead.** The final heuristic is called *greedy look-ahead*. This heuristic considers the information derived from previous assignments, as greedy look-back does, while also trying to estimate the best cost achievable under the current branching decision. This is done by greedily assigning values to future variables (i.e., unassigned ones), considering the relaxed problem (i.e., without distance constraints).

In more detail, suppose that there are  $p$  decision variables and  $x_i$  is the one under consideration. We visit the future variables one by one and for each variable  $x_j, i + 1 \leq j \leq p$  and each value  $v_j \in \text{Dom}(x_j)$ , we find the sum of SP distances between all demand points and their nearest facility, taking into account the assignments for variables  $x_1, \dots, x_{j-1}$ , excluding the current one ( $x_i$ ). After all values in  $\text{Dom}(x_j)$  have been processed, the value that minimizes this sum is then temporarily assigned to  $x_j$  and this is repeated until all variables (except  $x_i$ ) have been assigned. After the “best” partial assignment including all variables except  $x_i$  has been computed, we calculate the best complete assignment by trying the addition of each value  $v_i \in \text{Dom}(x_i)$  to the partial assignment, so as to make it complete. The value that minimizes the cost of the total assignment is selected.

Algorithm 7 depicts this process.

---

**Algorithm 7.** Greedy\_LookAhead( $X, \text{Dom}, x_i$ )

---

```

min-dist  $\leftarrow \infty$ ;
val  $\leftarrow -1$ ;
for each  $x_j, i + 1 \leq j \leq p$ 
  dist  $\leftarrow \infty$ ;
  tmp_val  $\leftarrow -1$ ;
  for each  $v_j \in \text{Dom}(x_j)$ 
     $x_j \leftarrow v_j$ ;
    temp-cost  $\leftarrow$  sum of SP distances of all clients and their nearest facility among  $x_1, \dots, x_j$  except  $x_i$ ;
    if temp-cost < dist
      dist  $\leftarrow$  temp-cost;
      tmp_val  $\leftarrow v_j$ ;
   $x_j \leftarrow$  tmp_val;
for each  $v_i \in \text{Dom}(x_i)$ 
   $x_i \leftarrow v_i$ ;
  temp-cost  $\leftarrow$  sum of SP distances of all clients and their nearest facility among  $x_1, \dots, x_p$ ;
  if temp-cost < min-dist
    min-dist  $\leftarrow$  temp-cost;
    val  $\leftarrow v_i$ ;
return val;

```

---

We also experimented with a similar heuristic, using a slightly different approach for the estimation of the best value to branch on. This heuristic is computed as follows: for each  $v_i \in \text{Dom}(x_i)$ , we visit the future variables one by one and for each variable  $x_j, i + 1 \leq j \leq p$  and each value  $v_j \in \text{Dom}(x_j)$ , we find the sum of SP distances between all demand points and their nearest facility, considering the assignments to variables  $x_1, \dots, x_{j-1}$ , including  $x_i \leftarrow v_i$ . After all values in  $\text{Dom}(x_j)$  have been processed, the value that minimizes this sum is then temporarily assigned to  $x_j$  and this is repeated until all variables have been assigned. The value for current variable  $x_i$  that is involved in the complete assignment with the minimum sum of distances is assigned to  $x_i$ . This variant is considerably less efficient than the greedy look-ahead described above, as it is significantly more expensive to compute. Although results are similar on relatively small instances, experiments with instances that include variables with large domains, demonstrated that this method is computationally prohibitive. Therefore, we do not consider this heuristic any further.

## 7 Experiments

Computations were performed on an Intel i7 CPU 8700 with 16 GB of main memory, a clock of 3.2 GHz, an L1 cache of 348 kB, an L2 cache of 2 MB, and an L3 cache of 12 MB, running under CentOS 8.4. We set a time limit of 3,600 s for all the experiments reported below. The ILP model was solved using Gurobi 9.0.3 (Gurobi Optimization, LLC, 2023) and was stored in compressed sparse column format, as the constraint matrix can sometimes be too large to be stored as a full array. The CP model was written in the CPMpy modeling tool (Guns, 2019) and compiled into CP-SAT OR-Tools (O.-T. Development Team, 2024). CPMpy is a CP modeling library in Python, based on Numpy, that provides direct solver access. While a native OR-Tools implementation via the Python API was feasible, CPMpy allows us to maintain solver flexibility (for further experimentation) without sacrificing efficiency or model integrity. In fact, we have actively verified that the model generated by CPMpy is identical to the native OR-Tools model.

CP-SAT OR-Tools is a state-of-the-art CP solver that leverages SAT (satisfiability) methods. Since this solver does not support real-valued domains, which are present in our problems, we scale all values by a sufficiently large factor to convert them into integers. The results are then transformed back to their original values. In addition, OR-Tools includes several built-in variable and value selection heuristics. In preliminary experiments, we tested various settings, such as prioritizing decision variables  $x \in X$  during branching and applying different value orderings (e.g., min-value). However, results did not show any significant improvements in runtime or objective value across these settings compared to the default configuration of OR-Tools, while often giving worse results. Consequently, we used the default OR-Tools settings for our experiments.

The heuristic CP approach was implemented in a custom (heuristic) solver written in C, following the structure of Algorithm 1. This solver will be denoted as  $\text{CP}_h$  hereafter.  $\text{CP}_h$  uses the dom/wdeg heuristic for variable ordering, arc consistency for the propagation of distance constraints between facilities, and node consistency for the constraints between facilities and clients. Standard lexicographic value ordering was used when comparing  $\text{CP}_h$  to Gurobi and OR-Tools in Section 7.2.1. An evaluation of the value ordering heuristics described in Section 6.4 was also carried out (Section 7.2.2).

### 7.1 Problem Generation Models

Given the lack of benchmarks for the pMD, we experimented with pMD instances generated in three different ways. The first two place the candidate facility sites and the clients on the nodes of a grid, while the third uses the p-median benchmark library as a basis to create pMDs.

**7.1.1 Grid-Based Generation Model.** The *grid generation model* creates problems embedded in an  $n \times n$  grid. It takes the following parameters:  $n, p, |CL|, |P|$ . We first randomly select  $|CL| + |P|$  among the  $n \times n$  nodes, for the clients and the potential facility locations.  $|CL|$  of these nodes are randomly selected to place the clients and the remaining nodes are the potential facility locations.

We assume that the weight of each edge in the grid is equal to 1. Therefore, given that we have a grid, the length of the shortest path between any client and any potential facility site can be, at best, equal to the Manhattan distance between them. For each distance constraint  $D[x_i, x_j] > d_{1ij}$  between facilities  $x_i$  and  $x_j$ ,  $d_{1ij}$  is randomly set to an integer number in the interval  $[0, \text{max\_euc}/2]$ , where  $\text{max\_euc}$  is the maximum Euclidean distance between two points on the grid. Accordingly, for the constraints specifying the distances between facilities and clients, a random integer is set in the (experimentally selected) interval  $[0, 3]$ , in order to minimize infeasibilities.

**7.1.2 Minimum Weighted Covering Generation Model.** The second random generator is a variant of the first one and is based on the generator used in Berman and Huang (2008) for minimum weighted covering location problems with distance



constraints. It again creates pMDs embedded in an  $n \times n$  grid, taking the parameters  $n, p, |CL|, |P|$ , plus the extra parameters  $t$  and  $k$ . Parameter  $t$  takes a value in the interval  $(0, 1]$  and determines the tightness of the generated network. That is, given that the total number of edges in an  $n \times n$  grid is  $2 \times n \times (n - 1)$ , the network will contain  $t \times 2 \times n \times (n - 1)$  edges. Parameter  $k$  is used for the generation of the constraints, as described below.

The generator first randomly creates a tree with  $|CL| + |P|$  nodes in the grid. This is done by first randomly generating the Cartesian coordinates of the  $|CL| + |P|$  nodes in the grid uniformly. Then nodes are connected randomly until a tree is formed. At this point, the only edges in the grid that are active (i.e., belong to the network under generation) are the  $|CL| + |P| - 1$  edges that belong to the tree. Then, the remaining  $(t \times 2 \times n \times (n - 1)) - |CL| + |P| - 1$  required edges are added to the network by randomly selecting pairs of adjacent nodes<sup>1</sup> in the tree and adding the edge that connects them to the network, if it does not already belong to it. Hence, cycles can be formed in the network through this process. The next step in the process randomly selects  $|CL|$  among the  $|CL| + |P|$  nodes to place the clients. The  $|P|$  nodes remaining are the potential locations of the facilities.

We have tried two different ways for the generation of the distance constraints:

- (1) For any constraint between variables  $x_i$  and  $x_j$ ,  $d1_{ij}$  is randomly set to an integer number in the interval  $[0, \lfloor n/k \rfloor]$ , where  $k$  is a parameter. Accordingly, for the unary constraints specifying the distances between facilities and clients.
- (2) The variables are split into two sets, with the first one corresponding to facilities that need to be placed further apart, and the second to facilities that can be closer to clients and to one another. The distances for the constraints are drawn from three intervals, the *low*, the *medium*, and the *high* one. For a constraint where both involved variables  $x_i$  and  $x_j$  belong to the first set,  $d1_{ij}$  is set to a random number in the high interval. Accordingly, for a constraint between variables of the second set, the distance is set to a random number in the low interval, while for a constraint between variables from different sets, the distance is set to a value in the medium interval. In our experiments, the three intervals were generated by splitting the interval  $[0, \lfloor n/k \rfloor]$  in three (almost) equal parts.

**7.1.3 *p*-Median Benchmark Library Based Generation.** The *p*-median based generator takes instances from the *p*-median benchmark dataset (Beasley, 1985), consisting of problems with 100–900 nodes and 5–200 facilities. We randomly select  $|P|$  nodes to be candidate facilities, while the remaining nodes are clients. We have considered two cases: **1)** 80% of the nodes are candidate facility sites and the remaining 20% are clients. If the resulting number of candidate sites is less than or equal to  $p$ , then we progressively increase the number of candidate sites until  $|P| > p$  and the generated instances are feasible. **2)** Twenty percent (20%) of the nodes are candidate facility sites and the remaining 80% are clients. Similarly to the previous case, we progressively increase the number of candidate sites until  $|P| > p$  and the generated instances are feasible. To set the parameter  $d1_{ij}$ , we find the minimum and maximum distance between all pairs of candidate sites and we set  $d1_{ij}$  equal to a random number in the range  $[\min, \min + (\max - \min)/10]$ . Similarly for parameter  $d2_i$ .

From now on, we will refer to the grid model of Subsection 7.1.1 (respectively, 7.1.2) as  $\text{grid}_1$  (respectively,  $\text{grid}_2$ ). For the  $\text{grid}_1$  and *p*-median generation models and each setting of the parameters, 10 instances were generated, whereas for the  $\text{grid}_2$  generation model, we generated 20 instances.

Table 1 details the classes generated for pMDs using the three-generation models. For the *p*-median-based ones, we give the name of the *p*-median benchmark used as basis. Each such class is defined by the parameters  $\langle |V|, p, |P|, |CL| \rangle$ . For example, class  $\langle 500, 5, 100, 400 \rangle$  includes problems with 500 points, 5 facilities, 100 potential locations, and 400 clients. The second column gives classes where the number of clients is larger or equal to the number of candidate sites, while the third gives classes where there are more candidate sites than clients. Each  $\text{grid}_1$  class is defined by the parameters  $\langle n, p, |P|, |CL| \rangle$ . For example, in class  $\langle 10, 20, 80, 20 \rangle$  we have a  $10 \times 10$  grid, 20 facilities, 80 potential locations, and 20 clients. In the case of  $\text{grid}_2$ , each class is defined by the parameters  $\langle n, p, |P|, |CL|, t, m \rangle$ . That is, the additional parameter  $m$  is used, denoting the method used for the generation of the distance constraints, taking value 1 or 2, as detailed in Section 7.1.2. Parameter  $k$  was set to 3. All generated pMD instances can be found in the following GitHub repository (<https://github.com/pkiosif/pMD-problems>).

## 7.2 Experimental Results

We first evaluate the performance of the custom solver, which implements the branch pruning heuristic, by comparing it against Gurobi and OR-Tools on instances generated using the three models described above. Then, we evaluate the domain-specific value ordering heuristics, and finally, we give results from the sampling method of Section 6.3.

**7.2.1 Comparing the Solvers.** Table 2 compares the three solvers on *p*-median-based problems. We report the total CPU times taken by Gurobi, OR-Tools, and  $\text{CP}_h$  ( $\sum \text{cpu}$  columns). The number of instances where any solver reached the cutoff

**Table 1.** Problem Classes and Their Characteristics.

| p-Median based      | $ CL  \geq  P $   | $ P  >  CL $     | Grid based | grid <sub>1</sub> | grid <sub>2</sub>      |
|---------------------|-------------------|------------------|------------|-------------------|------------------------|
| pmed05              | <100,33,40,60>    | <100,33,80,20>   | g1         | <10,10,80,20>     | <20,15,70,30,0.3,1>    |
| pmed10              | <200,67,100,100>  | <200,67,160,40>  | g2         | <10,20,80,20>     | <20,20,115,35,0.3,1>   |
| pmed15              | <300,100,150,150> | <300,100,240,60> | g3         | <20,10,350,50>    | <20,20,80,20,0.2,1>    |
| pmed16 <sup>1</sup> | <400,5,80,320>    | <400,5,320,80>   | g4         | <20,20,350,50>    | <20,35,150,50,0.5,1>   |
| pmed16 <sup>2</sup> | <400,10,80,320>   | <400,10,320,80>  | g5         | <20,25,350,50>    | <30,25,250,50,0.3,1>   |
| pmed16 <sup>3</sup> | <400,20,80,320>   | <400,20,320,80>  | g6         | <30,25,250,50>    | <30,30,300,100,0.35,1> |
| pmed21              | <500,5,100,400>   | <500,5,400,100>  | g7         | <30,20,300,50>    | <20,12,70,30,0.6,2>    |
| pmed26              | <600,5,120,480>   | <600,5,480,120>  | g8         | <30,10,500,100>   | <20,30,245,75,0.5,2>   |
| pmed31              | <700,5,140,560>   | <700,5,560,140>  | g9         | <30,20,500,100>   | <30,25,230,70,0.25,2>  |
| pmed33              | <700,70,350,350>  | <700,70,560,140> | g10        | <30,20,700,200>   | <20,35,170,30,0.5,2>   |
| pmed36              | <800,10,400,400>  | <800,10,640,160> | g11        | <50,100,1300,200> |                        |
| pmed38              | <900,5,450,450>   | <900,5,720,180>  |            |                   |                        |

limit of 1 h without terminating is given in brackets in these columns. In this case, we count 3,600 s toward its CPU time sum and we record the best solution it was able to find. We also report the mean optimality gap for  $CP_h$  (%gap). If the cost of the optimal solution found by a complete solver is  $x$  and the cost of the best solution found by  $CP_h$  is  $y$  then the gap is computed as  $(y - x)/x$ . As Gurobi solved all instances of all p-median classes, the optimal cost is known in these cases. Columns  $t_b$  give the mean CPU time taken by the solvers to find their best solution. Columns  $t_m$  give the mean CPU time taken by Gurobi and OR-Tools to find the first solution that matches (or improves) the cost of the best solution found by  $CP_h$ . If one is unable to match the best solution found by  $CP_h$  in some instances,  $t_m$  is left blank (-). The last column gives the number of instances where  $CP_h$  found the optimal solution (#opt). In brackets, we give the number of instances where the optimal is known<sup>2</sup>. Finally, OR-Tools suffered memory exhaustion and crashed on all instances of some classes. This is denoted by MEM in the corresponding  $\sum_{cpu}$  column.

Evidently, Gurobi outperforms OR-Tools in all classes of the p-median-based problems in terms of total CPU time and mean cost. In fact, Gurobi finds these problems quite easy, terminating within the time limit in all instances. Hence, it is clearly the best solver for these types of pMDs. Looking at the performance of  $CP_h$ , it is able to find many near-optimal solutions, explaining the low optimality gaps in many classes, while managing to locate more than half of the optimal solutions in some classes (e.g., pmed16, pmed21, pmed26 and pmed31 of  $|CL| \geq |P|$ ). On the other hand,  $CP_h$ 's performance is much worse than Gurobi's in classes with a large number of variables (i.e., facilities to be assigned), such as pmed15 and pmed33. In such problems, not only do run times increase considerably, but also higher optimality gaps are obtained, meaning that  $CP_h$  found it hard to locate solutions of good quality.

However, if we compare  $CP_h$  to OR-Tools, as columns  $t_b$  and  $t_m$  indicate, it takes a significant amount of time (often orders of magnitude longer runs) for OR-Tools to match the solutions found by  $CP_h$ . OR-Tools finds the p-median-based pMDs quite difficult to solve, reaching the time limit of 1 h in most classes, and not being able to prove optimality. Furthermore, in classes with a large number of variables and domain sizes, the model created exceeds the memory capacity of our machine, resulting in a system crash. On the other hand,  $CP_h$ , being a lightweight solver that uses a simpler pMD model, did not face such problems and was quite competitive to Gurobi in classes with a few number of facilities (especially in category  $|CL| \geq |P|$ ).

Table 3 compares the three solvers on problems generated using the grid models. The columns are the same as in Table 2, except that the mean values of the cost are presented rather than the optimality gaps. This is because the optimal solutions are unknown in many instances of many classes, as Gurobi and OR-Tools did not manage to terminate within the time limit. The lowest mean objective value is highlighted in bold. For Gurobi and OR-Tools, we give in brackets (in the cost column) the number of instances in which they managed to find at least one solution. For  $CP_h$ , we give in brackets (in the cost column) the number of instances where it managed to find an equal or better solution than both of the other solvers. If a solver was unable to find any solution within the time limit in some instances of a class then columns  $t_b$  and cost are left blank (-) for this class.

Interestingly, the results differ significantly from those in Table 2. OR-Tools performs better than Gurobi in general, solving problems in classes that are unreachable for the latter. Gurobi seems to struggle to even find any solution to most problems of various classes. In fact, in only 7 out of 21 classes, it is able to locate at least one solution in all instances of the class. On the other hand, OR-Tools is able to locate at least one solution in all instances in 16 out of 21 classes. Gurobi suffered memory exhaustion in g10 and g11 classes of grid<sub>1</sub> problems (as did OR-Tools in the latter). Regarding g1<sup>(1)</sup> of grid<sub>2</sub> problems, all solvers managed to prove infeasibility for the one infeasible instance of this class, almost instantly.

**Table 2.** Comparing Solvers on p-Median-Based pMD Problems.

| class               | Gur $\sum$ cpu | $t_b$ | $t_m$ | ORt $\sum$ cpu | $t_b$ | $t_m$ | CP <sub>h</sub> $\sum$ cpu | $t_b$ | %gap  | #opt   |
|---------------------|----------------|-------|-------|----------------|-------|-------|----------------------------|-------|-------|--------|
| $ CL  \geq  P $     |                |       |       |                |       |       |                            |       |       |        |
| pmed05              | 2 (0)          | 0     | 0     | 8,038 (0)      | 289   | 177   | >22,460 (6)                | —     | —     | 5 (10) |
| pmed10              | 27 (0)         | 2     | 2     | >36,000 (10)   | 2,092 | —     | >27,141 (6)                | 829   | 2.83  | 0 (10) |
| pmed15              | 137 (0)        | 12    | 8     | >36,000 (10)   | 3,489 | —     | >36,000 (10)               | 983   | 3.18  | 0 (10) |
| pmed16 <sup>1</sup> | 101 (0)        | 5     | 5     | 4,078 (0)      | 36    | 35    | 3 (0)                      | 0.2   | 0.15  | 6 (10) |
| pmed16 <sup>2</sup> | 90 (0)         | 7     | 5     | >20,834 (5)    | 786   | —     | 32 (0)                     | 3     | 0.41  | 3 (10) |
| pmed16 <sup>3</sup> | 34 (0)         | 2     | 1     | >36,000 (10)   | 1,782 | —     | 421 (0)                    | 41    | 0.83  | 0 (10) |
| pmed21              | 205 (0)        | 8     | 8     | >13,204 (3)    | 127   | 92    | 5 (0)                      | 0.5   | 0.16  | 7 (10) |
| pmed26              | 154 (0)        | 14    | 13    | >13,759 (3)    | 114   | 110   | 9 (0)                      | 0.8   | 0.25  | 6 (10) |
| pmed31              | 1,093 (0)      | 41    | 41    | >24,050 (6)    | 157   | 155   | 19 (0)                     | 2     | 0.23  | 7 (10) |
| pmed33              | 206 (0)        | 18    | 16    | MEM            | —     | —     | >36,000 (10)               | 3,527 | 14.07 | 0 (10) |
| pmed36              | 2,764 (0)      | 136   | 86    | >36,000 (10)   | 2,463 | —     | 2,210 (0)                  | 209   | 0.74  | 0 (10) |
| pmed38              | 861 (0)        | 56    | 22    | >24,717 (6)    | 470   | 448   | 117 (0)                    | 10    | 0.4   | 1 (10) |
| $ P  >  CL $        |                |       |       |                |       |       |                            |       |       |        |
| pmed05              | 4 (0)          | 0     | 0     | 337 (0)        | 31    | 31    | >7,396 (2)                 | 209   | 1.51  | 3 (10) |
| pmed10              | 46 (0)         | 4     | 4     | >29,185 (7)    | 1,535 | 1,354 | >28,440 (7)                | 1,566 | 6.5   | 0 (10) |
| pmed15              | 203 (0)        | 19    | 16    | MEM            | —     | —     | >36,000 (10)               | 2,899 | 10.6  | 0 (10) |
| pmed16 <sup>1</sup> | 32 (0)         | 2     | 2     | >36,000 (10)   | 192   | —     | 52 (0)                     | 5     | 0.75  | 1 (10) |
| pmed16 <sup>2</sup> | 28 (0)         | 2     | 1     | >36,000 (10)   | 2,386 | —     | 492 (0)                    | 46    | 2.18  | 0 (10) |
| pmed16 <sup>3</sup> | 45 (0)         | 3     | 2     | >36,000 (10)   | 3,090 | —     | 4,527 (0)                  | 425   | 5.03  | 0 (10) |
| pmed21              | 90 (0)         | 7     | 5     | >36,000 (10)   | 773   | —     | 137 (0)                    | 12    | 0.35  | 3 (10) |
| pmed26              | 91 (0)         | 7     | 6     | >36,000 (10)   | 993   | —     | 149 (0)                    | 13    | 0.62  | 4 (10) |
| pmed31              | 687 (0)        | 48    | 32    | >36,000 (10)   | 1,557 | —     | 382 (0)                    | 35    | 0.35  | 4 (10) |
| pmed33              | 441 (0)        | 42    | 39    | MEM            | —     | —     | >36,000 (10)               | 3,508 | 35.97 | 0 (10) |
| pmed36              | 1,969 (0)      | 141   | 80    | MEM            | —     | —     | 3,997 (0)                  | 384   | 0.79  | 0 (10) |
| pmed38              | 561 (0)        | 37    | 27    | >36,000 (10)   | 2,683 | —     | 274 (0)                    | 25    | 0.8   | 3 (10) |

Note. pMD = p-median with distance constraints.

CP<sub>h</sub> is usually much more efficient, finding solutions in all instances of all classes within the time limit, apart from three instances in the g10 grid<sub>2</sub> class, but including all instances of the very hard g10 and g11 grid<sub>1</sub> classes. In the vast majority of instances, CP<sub>h</sub> is able to terminate much faster (sometimes by orders of magnitude) than the two complete solvers. Regarding solution quality, in the easier classes such as g1 and g2 of grid<sub>1</sub>, where both complete solvers found solutions in all instances, the solutions discovered by CP<sub>h</sub> are typically worse (though not by large margins). However, in the harder classes (e.g., g9 of grid<sub>1</sub> and grid<sub>2</sub>) that are out of reach for Gurobi and very difficult for OR-Tools, CP<sub>h</sub> managed to locate solutions of better quality. Overall, it beat or equaled the best of the other solvers in terms of solution quality in 73 out of 110 grid<sub>1</sub> instances, and 88 out of 200 grid<sub>2</sub> instances.

Taking a deeper look at the performance of the solvers, an important factor that seems to affect it is the number of solutions existing in a problem. Gurobi benefits from the presence of many solutions in an instance, while this does not hold for CP<sub>h</sub> and OR-Tools. In contrast, Gurobi finds it hard to deal with problems that only have a few solutions, whereas the CP solvers handle such problems more efficiently. Let us note that p-median-generated instances, where Gurobi excels, typically have a very large number of solutions. For example, a simple enumeration of the feasible solutions, using a complete CP solver without the objective function, counted 13,343,409 solutions in 10 min of CPU time on average for the 10 instances of pmed38 with  $|CL| \geq |P|$ . The enumeration was stopped after 10 min, meaning that the actual number of solutions could be much higher.

On the other hand, grid-based problems, where Gurobi falters, typically have few solutions. For instance, we counted only 11,872 solutions until termination in class g3 of grid<sub>2</sub>. Further to this, focusing on two contrasting instances of class g2 of grid<sub>1</sub>; one having only 58 feasible solutions and another with 87,950,294 solutions, Gurobi was not able to find a solution in the former, while it managed to match the solution of CP<sub>h</sub> in 163 secs in the latter. When there are few solutions, Gurobi is either unable to find any solution within the time limit and/or finds it hard to obtain a good initial bound, meaning that its progress towards the optimal is very slow. In contrast, in the presence of many solutions, such as in the p-median-based instances, it starts with a good initial bound and is able to quickly improve it.

**Table 3.** Comparing Solvers on Grid-Based pMD Problems.

| class                   | Gur $\sum$ cpu | $t_b$ | $t_m$ | cost              | ORt $\sum$ cpu | $t_b$ | $t_m$ | cost              | CP <sub>n</sub> $\sum$ cpu | $t_b$ | cost              | #opt    |
|-------------------------|----------------|-------|-------|-------------------|----------------|-------|-------|-------------------|----------------------------|-------|-------------------|---------|
| <b>GRID<sub>1</sub></b> |                |       |       |                   |                |       |       |                   |                            |       |                   |         |
| g1                      | 8 (0)          | 0     | 0     | <b>36.1</b> (10)  | 62 (0)         | 3     | 2     | 36.1 (10)         | 1 (0)                      | 0     | 37.1 (2)          | 2 (10)  |
| g2                      | >20,139 (5)    | —     | —     | — (9)             | 861 (0)        | 33    | 33    | <b>28.6</b> (10)  | 27 (0)                     | 1     | 28.9 (7)          | 7 (10)  |
| g3                      | >5,003 (1)     | 91    | 36    | <b>138.6</b> (10) | >36,000 (10)   | 1,244 | 775   | 141.2 (10)        | 199 (0)                    | 18    | 144.7 (0)         | 0 (9)   |
| g4                      | >36,000 (10)   | —     | —     | — (1)             | >36,000 (10)   | 1,429 | —     | 127.7 (10)        | >32,827 (9)                | 1,489 | <b>122.1</b> (9)  | 0 (0)   |
| g5                      | >36,000 (10)   | —     | —     | — (0)             | >33,331 (9)    | —     | —     | — (7)             | >30,020 (7)                | 1,810 | <b>126.8</b> (8)  | 1 (1)   |
| g6                      | >36,000 (10)   | —     | —     | — (0)             | >36,000 (10)   | —     | —     | — (3)             | >23,473 (5)                | 933   | <b>189.6</b> (10) | 0 (0)   |
| g7                      | >36,000 (10)   | —     | —     | — (0)             | >36,000 (10)   | —     | —     | — (9)             | >36,000 (10)               | 1,727 | <b>185.8</b> (8)  | 0 (0)   |
| g8                      | >23,536 (6)    | 632   | 553   | <b>425.6</b> (10) | >36,000 (10)   | 1,666 | —     | 441.5 (10)        | 1,607 (0)                  | 146   | 439.8 (1)         | 0 (4)   |
| g9                      | >36,000 (10)   | —     | —     | — (0)             | >36,000 (10)   | 2,017 | —     | 393.3 (10)        | >31,546 (8)                | 1,657 | <b>378.5</b> (8)  | 0 (0)   |
| g10                     | MEM            | —     | —     | —                 | >36,000 (10)   | —     | —     | — (9)             | >27,476 (6)                | 1,427 | <b>778.6</b> (10) | 0 (0)   |
| g11                     | MEM            | —     | —     | —                 | MEM            | —     | —     | —                 | >36,000 (10)               | 3,490 | <b>946.6</b> (10) | 0 (0)   |
| <b>GRID<sub>2</sub></b> |                |       |       |                   |                |       |       |                   |                            |       |                   |         |
| g1 <sup>(1)</sup>       | 988 (0)        | 21    | 10    | 56 (19)           | 447 (0)        | 11    | 9     | <b>56</b> (19)    | 4 (0)                      | 0.1   | 57.05 (7)         | 7 (19)  |
| g2 <sup>(1)</sup>       | 5,798 (0)      | 165   | 71    | <b>70.95</b> (20) | >18,070 (2)    | 156   | 39    | 70.95 (20)        | 83 (0)                     | 2.3   | 72.8 (7)          | 7 (20)  |
| g3 <sup>(1)</sup>       | >53,659 (13)   | —     | —     | — (16)            | 2,600 (0)      | 61    | 61    | <b>40.7</b> (20)  | 136 (0)                    | 3     | 40.9 (18)         | 18 (20) |
| g4 <sup>(1)</sup>       | >69,997 (19)   | —     | —     | — (2)             | >44,736 (10)   | 1,191 | —     | 99.15 (20)        | >29,560 (4)                | 804   | <b>98.95</b> (14) | 8 (10)  |
| g5 <sup>(1)</sup>       | >72,000 (20)   | —     | —     | — (0)             | >70,092 (19)   | 1,872 | —     | <b>133.6</b> (20) | >63,462 (16)               | 1,209 | 136 (9)           | 0 (1)   |
| g6 <sup>(1)</sup>       | >68,827 (19)   | —     | —     | — (6)             | >60,558 (15)   | 1,606 | —     | 305.35 (20)       | >37,210 (7)                | 1,063 | <b>298.7</b> (14) | 3 (5)   |
| g7 <sup>(2)</sup>       | 29 (0)         | 0     | 0     | <b>54.85</b> (20) | 720 (0)        | 8     | 5     | 54.85 (20)        | 3 (0)                      | 0     | 56.05 (7)         | 7 (20)  |
| g8 <sup>(2)</sup>       | 326 (0)        | 15    | 14    | <b>177.2</b> (20) | >72,000 (20)   | 2,270 | 2,435 | 183.7 (20)        | >10,159 (1)                | 392   | 183.75 (1)        | 1 (20)  |
| g9 <sup>(2)</sup>       | >60,879 (16)   | —     | —     | — (11)            | >70,285 (19)   | 1,641 | —     | 179.35 (20)       | >26,960 (5)                | 747   | <b>176.55</b> (5) | 0 (5)   |
| g10 <sup>(2)</sup>      | >29,667 (5)    | —     | —     | — (17)            | >68,529 (19)   | 941   | —     | <b>46.1</b> (20)  | >39,189 (9)                | —     | — (6)             | 5 (15)  |

Note. pMD = p-median with distance constraints.

The lowest mean objective value for each problem class is highlighted in bold. In case of ties, the value of the best-performing solver is highlighted.

**Table 4.** Comparing Value Ordering Heuristics on p-Median Based pMD Problems ( $|CL| \geq |P|$ ).

| $ CL  \geq  P $     |                   |       |                          |                   |       |                           |                   |       |                           |
|---------------------|-------------------|-------|--------------------------|-------------------|-------|---------------------------|-------------------|-------|---------------------------|
| Class               | Lexico            |       |                          | Greedy-minmax     |       |                           | Greedy-minsum     |       |                           |
|                     | $\sum \text{cpu}$ | $t_b$ | cost                     | $\sum \text{cpu}$ | $t_b$ | cost                      | $\sum \text{cpu}$ | $t_b$ | cost                      |
| pmed05              | >22,460 (6)       | 453   | 2,126 <sub>8</sub> (5)   | >20,453 (5)       | 272   | 2,190.89 <sub>9</sub> (3) | >14,008 (3)       | 341   | 2,177.22 <sub>9</sub> (6) |
| pmed10              | >27,141 (6)       | 829   | 1,638.7 (0)              | >28,036 (7)       | 681   | 1,633.8 (0)               | >20,710 (5)       | 513   | 1,630.6 (1)               |
| pmed15              | >36,000 (10)      | 983   | 2,119.1 (0)              | >36,000 (10)      | 1,599 | 2,125.7 (0)               | >36,000 (10)      | 785   | 2,121.2 (0)               |
| pmed16 <sup>1</sup> | 3 (0)             | 0.2   | 8,680.2 (6)              | 1.4 (0)           | 0     | 8,682.5 (5)               | 1 (0)             | 0     | 8,677.3 (5)               |
| pmed16 <sup>2</sup> | 32 (0)            | 3     | 7,723.6 (3)              | 21 (0)            | 2     | 7,710.4 (2)               | 17 (0)            | 1     | <b>7,707.5</b> (4)        |
| pmed16 <sup>3</sup> | 421 (0)           | 41    | 6,284.1 (0)              | 364 (0)           | 35    | <b>6,271.1</b> (0)        | 229 (0)           | 21    | 6,285.1 (0)               |
| pmed21              | 5 (0)             | 0.5   | 9,626 (7)                | 4 (0)             | 0.3   | 9,624.4 (7)               | 3 (0)             | 0.1   | 9,623.9 (8)               |
| pmed26              | 9 (0)             | 0.8   | 10,466.3 (6)             | 6 (0)             | 0.4   | 10,462.2 (7)              | 5 (0)             | 0.3   | 10,469.9 (6)              |
| pmed31              | 19 (0)            | 1.7   | 10,464.5 (7)             | 13 (0)            | 1     | 10,453.2 (6)              | 8 (0)             | 0.4   | 10,456.4 (6)              |
| pmed33              | >36,000 (10)      | 3,527 | 3,755.4 (0)              | >36,000 (10)      | 3,478 | 3,761.4 (0)               | >36,000 (10)      | 3,525 | 3,717.2 (0)               |
| pmed36              | 2,210 (0)         | 209   | 6,388.2 (0)              | 1,411 (0)         | 122   | 6,391.3 (1)               | 624 (0)           | 43    | 6,390.3 (0)               |
| pmed38              | 117 (0)           | 10    | 7,159.8 (1)              | 93 (0)            | 6     | 7,150.9 (4)               | 77 (0)            | 5     | 7,156.7 (2)               |
| Class               | Greedy-lookback   |       |                          | Greedy-lookahead  |       |                           |                   |       |                           |
|                     | $\sum \text{cpu}$ | $t_b$ | cost                     | $\sum \text{cpu}$ | $t_b$ | cost                      | $\sum \text{cpu}$ | $t_b$ | cost                      |
| pmed05              | >21,936 (6)       | 55    | 2,152.5 <sub>8</sub> (4) | >18,387 (5)       | 41    | 2,133.75 <sub>8</sub> (4) |                   |       |                           |
| pmed10              | >28,969 (8)       | 321   | <b>1,628.3</b> (0)       | >25,365 (6)       | 718   | 1,646.2 (1)               |                   |       |                           |
| pmed15              | > 36,000 (10)     | 586   | 2,114.7 (0)              | >35,290 (9)       | 1,428 | <b>2,113.1</b> (0)        |                   |       |                           |
| pmed16 <sup>1</sup> | 1 (0)             | 0     | 8,677.3 (5)              | 1 (0)             | 0     | <b>8,672.1</b> (8)        |                   |       |                           |
| pmed16 <sup>2</sup> | 10 (0)            | 0.6   | 7,712.6 (3)              | 11 (0)            | 0.8   | 7,711.8 (2)               |                   |       |                           |
| pmed16 <sup>3</sup> | 141 (0)           | 12    | 6,282.7 (0)              | 339 (0)           | 32    | 6,281.3 (0)               |                   |       |                           |
| pmed21              | 2 (0)             | 0     | 9,623.9 (8)              | 3 (0)             | 0.1   | <b>9,622.6</b> (8)        |                   |       |                           |
| pmed26              | 4 (0)             | 0.2   | <b>10,458</b> (8)        | 4 (0)             | 0.2   | 10,461.7 (7)              |                   |       |                           |
| pmed31              | 9 (0)             | 0.4   | 10,456.4 (6)             | 7 (0)             | 0.4   | <b>10,448.1</b> (9)       |                   |       |                           |
| pmed33              | >36,000 (10)      | 2,894 | 3,478.2 (0)              | >35,171 (8)       | 2,146 | <b>3,419.4</b> (0)        |                   |       |                           |
| pmed36              | 549 (0)           | 35    | 6,386.5 (0)              | 335 (0)           | 14    | <b>6,373.8</b> (1)        |                   |       |                           |
| pmed38              | 66 (0)            | 4     | 7,156.7 (2)              | 60 (0)            | 3     | <b>7,138.1</b> (6)        |                   |       |                           |

Note. pMD = p-median with distance constraints.

The lowest mean objective value for each problem class is highlighted in bold. In case of ties, the value of the best-performing setting is highlighted.

**7.2.2 Value Ordering Heuristics.** Tables 4 to 7 evaluate the performance of  $CP_h$  under five value ordering heuristics, namely *lexico* and the ones presented in section 6.4 (greedy-minmax, greedy-minsum, greedy-lookback and greedy-lookahead). There are four tables in total, one for each category of problems (pmed with  $|CL| \geq |P|$ , pmed with  $|P| > |CL|$ , grid<sub>1</sub>, grid<sub>2</sub>). We report the total CPU time taken by the solver to terminate ( $\sum \text{cpu}$  columns) and, in brackets, the number of instances where the solver timed out. Columns  $t_b$  give the mean CPU time taken to find the best solution, using each heuristic, and the cost columns give the mean objective value (in brackets we give the number of instances where the optimal solution was located). The lowest mean objective value is highlighted in bold. Settings that failed to find a solution in any instance are excluded from the comparison. Finally, in case the solver did not manage to find any solution in some instances of a class, we calculate the mean cost and  $t_b$  over the instances of the class where at least one solution was discovered. We report the number of these instances with a subscript in the cost column.

When evaluating the heuristics, we need to consider their effect on the quality of the solutions that the solver discovers, as well as on the CPU time taken. Table 8 summarizes the results, focusing on these two criteria. Specifically, we give the number of instances over each of the four categories, where each domain-specific value ordering heuristic achieved better, worse, or similar results compared to *lexico*. Let us clarify how the comparison is made by explaining what the numbers in the different columns of the table indicate.

*#imp* The number of instances where a value ordering heuristic either improves both the objective value and the total CPU time, compared to *lexico*, or achieves the same results in one metric and better in the other.

*#wor* The number of instances where a value ordering heuristic either worsens both the objective value and the total CPU time, compared to *lexico*, or achieves the same results in one metric and worse in the other.

*#time* The number of instances where a value ordering heuristic improves the total CPU time, but worsens the value of the objective function.

**Table 5.** Comparing Value Ordering Heuristics on p-Median Based pMD Problems ( $|P| > |CL|$ ).

| $ P  >  CL $        |                 |       |                    |                  |       |                    |               |       |                  |
|---------------------|-----------------|-------|--------------------|------------------|-------|--------------------|---------------|-------|------------------|
| Class               | Lexico          |       |                    | Greedy-minmax    |       |                    | Greedy-minsum |       |                  |
|                     | $\sum_{cpu}$    | $t_b$ | cost               | $\sum_{cpu}$     | $t_b$ | cost               | $\sum_{cpu}$  | $t_b$ | cost             |
| pmed05              | >7,396 (2)      | 209   | 426.4 (3)          | >14,553 (4)      | 57    | 430.3 (3)          | >8,736 (2)    | 387   | 424.5 (4)        |
| pmed10              | >28,440 (7)     | 1,566 | 377.6 (0)          | >30,122 (8)      | 1,577 | 376.2 (0)          | >29,262 (7)   | 1,029 | <b>366.5</b> (0) |
| pmed15              | >36,000 (10)    | 2,899 | 545.7 (0)          | >36,000 (10)     | 1,625 | 526.7 (0)          | >36,000 (10)  | 2,215 | 524.1 (0)        |
| pmed16 <sup>1</sup> | 52 (0)          | 5     | 1,545.2 (1)        | 30 (0)           | 2     | 1,545.6 (2)        | 20 (0)        | 1     | 1,541.3 (4)      |
| pmed16 <sup>2</sup> | 492 (0)         | 46    | 1,296.7 (0)        | 419 (0)          | 37    | 1,297.4 (0)        | 187 (0)       | 13    | 1,297.7 (0)      |
| pmed16 <sup>3</sup> | 4,527 (0)       | 425   | 990 (0)            | 3,382 (0)        | 293   | 986.7 (0)          | 1,930 (0)     | 153   | 986.7 (0)        |
| pmed21              | 137 (0)         | 12    | 1,794.9 (3)        | 95 (0)           | 7     | 1,795 (3)          | 42 (0)        | 2     | 1,794.6 (3)      |
| pmed26              | 149 (0)         | 13    | 1,936.2 (4)        | 88 (0)           | 6     | 1,939 (1)          | 50 (0)        | 2     | 1,936.3 (2)      |
| pmed31              | 382 (0)         | 35    | 1,994.8 (4)        | 214 (0)          | 16    | 1,999 (3)          | 116 (0)       | 6     | 1,991.1 (6)      |
| pmed33              | >36,000 (10)    | 3,508 | 1,211.3 (0)        | >36,000 (10)     | 3,502 | 1,106 (0)          | >36,000 (10)  | 3,440 | 1,064.9 (0)      |
| pmed36              | 3,997 (0)       | 384   | 2,065.2 (0)        | 2,546 (0)        | 229   | 2,067.8 (1)        | 1,328 (0)     | 100   | 2,064.6 (0)      |
| pmed38              | 274 (0)         | 25    | 2,497.4 (3)        | 215 (0)          | 18    | 2,494.1 (3)        | 130 (0)       | 8     | 2,488.3 (5)      |
| Class               | Greedy-lookback |       |                    | Greedy-lookahead |       |                    |               |       |                  |
|                     | $\sum_{cpu}$    | $t_b$ | cost               | $\sum_{cpu}$     | $t_b$ | cost               | $\sum_{cpu}$  | $t_b$ | cost             |
| pmed05              | >7,322 (2)      | 56    | <b>424</b> (6)     | >3,782 (1)       | 83    | 424.4 (4)          |               |       |                  |
| pmed10              | >23,542 (5)     | 1,131 | 368.5 (0)          | >25,108 (6)      | 1,194 | 367.5 (0)          |               |       |                  |
| pmed15              | >35,145 (7)     | 1,301 | <b>510.4</b> (2)   | >36,000 (10)     | 2,090 | 524.6 (0)          |               |       |                  |
| pmed16 <sup>1</sup> | 18 (0)          | 1     | 1,541.3 (4)        | 18 (0)           | 1     | <b>1,540.6</b> (4) |               |       |                  |
| pmed16 <sup>2</sup> | 166 (0)         | 10    | 1,296.2 (0)        | 131 (0)          | 8     | <b>1,286.5</b> (1) |               |       |                  |
| pmed16 <sup>3</sup> | 1,384 (0)       | 97    | 986.5 (0)          | 1,090 (0)        | 73    | <b>986.5</b> (0)   |               |       |                  |
| pmed21              | 40 (0)          | 1     | 1,794.6 (3)        | 42 (0)           | 2     | <b>1,792.3</b> (3) |               |       |                  |
| pmed26              | 61 (0)          | 3     | 1,937.7 (2)        | 43 (0)           | 2     | <b>1,928.5</b> (7) |               |       |                  |
| pmed31              | 114 (0)         | 6     | <b>1,991.1</b> (6) | 78 (0)           | 2     | 1,992.8 (4)        |               |       |                  |
| pmed33              | >36,000 (10)    | 2,252 | 970.3 (0)          | >36,000 (10)     | 2,912 | <b>962.2</b> (0)   |               |       |                  |
| pmed36              | 1,143 (0)       | 81    | 2,067.6 (0)        | 994 (0)          | 70    | <b>2,061.8</b> (2) |               |       |                  |
| pmed38              | 113 (0)         | 6     | 2,488.3 (5)        | 90 (0)           | 4     | <b>2,483</b> (6)   |               |       |                  |

Note. pMD = p-median with distance constraints.

The lowest mean objective value for each problem class is highlighted in bold. In case of ties, the value of the best-performing setting is highlighted.

*#cost* The number of instances where a value ordering heuristic improves the value of the objective function, but worsens the total CPU time.

*#eq* The number of instances where a value ordering heuristic achieves similar results compared to *lexico* in both metrics.

Regarding run times, we consider that a heuristic improves the performance if it achieves at least a 30% speed-up compared to *lexico*. In any pairwise comparison between one of the four heuristics and *lexico* we consider the run time performance as equal on instances where both heuristics terminated in  $< 1$  s.

Tables 4 and 5, as well as Table 8, demonstrate that in the p-median-based classes, greedy-minmax, greedy-minsum, and *greedy-lookback* provide slight improvements compared to *lexico*. However, *greedy-lookahead* exhibits the best overall performance, as it is able to reduce the total run-times in most cases (often, quite considerably), while also notably improving the quality of the solutions discovered. For example, in classes pmed21 to pmed31 ( $|P| > |CL|$ ), it is able to reduce the total run times by an order of magnitude, while discovering better solutions, as well as more optimal ones. Also, in pmed33 where the solver timed out in all instances (Table 5), while all four heuristics were able to provide better results in terms of solution quality compared to *lexico*, greedy-lookahead was the best by considerable margins. Notably, in the pairwise comparison between greedy-lookahead and *lexico* given in Table 8, 168 instances are classified in the *#imp* column, whereas only 19 are classified in the *#wor* column. The respective numbers for greedy-lookback are 149 and 20, meaning that this heuristic is also very efficient compared to *lexico*.

Accordingly, Tables 6 and 7, as well as Table 8, demonstrate that the value ordering heuristics achieve better results than *lexico* in most classes of the grid categories, especially in *grid*<sub>1</sub> problems. In this category, they reduce the average cost in at least half of the classes (e.g., g4 and g6), and in the most challenging class, g11, the heuristics obtain a significant cost reduction, with greedy-lookback reducing the cost by more than half. The results in *grid*<sub>2</sub> do not differ significantly,

**Table 6.** Comparing Value Ordering Heuristics on Grid Based pMD Problems (GRID<sub>1</sub>).

| GRID <sub>1</sub> |                     |       |                        |                     |           |                         |                     |       |                      |
|-------------------|---------------------|-------|------------------------|---------------------|-----------|-------------------------|---------------------|-------|----------------------|
| Class             | Lexico              |       |                        | Greedy-minmax       |           |                         | Greedy-minsum       |       |                      |
|                   | $\sum_{\text{cpu}}$ | $t_b$ | cost                   | $\sum_{\text{cpu}}$ | $t_b$     | cost                    | $\sum_{\text{cpu}}$ | $t_b$ | cost                 |
| g1                | 1 (0)               | 0     | 37.1 (2)               | 1 (0)               | 0         | 36.8 (4)                | 1 (0)               | 0     | 36.9 (3)             |
| g2                | 27 (0)              | 1     | 28.9 (7)               | 30 (0)              | 0.4       | 28.9 (7)                | 28 (0)              | 0.6   | 28.9 (7)             |
| g3                | 199 (0)             | 18    | <b>144.7</b> (0)       | 118 (0)             | 10        | 145.1 (0)               | 100 (0)             | 8     | 145.6 (0)            |
| g4                | >32,827 (9)         | 1,489 | 122.1 (0)              | >32,099 (7)         | 1,823     | <b>120.9</b> (0)        | >32,233 (8)         | 1,592 | 121.1 (0)            |
| g5                | >30,020 (7)         | 1,810 | <b>126.8</b> (1)       | >27,204 (6)         | 724       | 126.5 <sub>8</sub> (1)  | >27,895 (6)         | 1,117 | 127 <sub>8</sub> (1) |
| g6                | >23,473 (5)         | 933   | <b>189.6</b> (0)       | >22,322 (5)         | 502       | 190.22 <sub>9</sub> (0) | >22,290 (5)         | 511   | 187 <sub>8</sub> (0) |
| g7                | >36,000 (10)        | 1,727 | 185.8 (0)              | >36,000 (10)        | 1,630     | 181.6 (0)               | >36,000 (10)        | 1,340 | 183.3 (0)            |
| g8                | 1,607 (0)           | 146   | <b>439.8</b> (0)       | 1,522 (0)           | 91        | 440.1 (0)               | 786 (0)             | 58    | 441.7 (0)            |
| g9                | >31,546 (8)         | 1,657 | 378.5 (0)              | >30,816 (8)         | 1,440     | 374.7 (0)               | >31,040 (8)         | 1,691 | 372.9 (0)            |
| g10               | >27,476 (6)         | 1,427 | 778.6 (0)              | >25,822 (6)         | 969       | <b>773.1</b> (0)        | >26,102 (6)         | 1,013 | 775.1 (0)            |
| g11               | >36,000 (10)        | 3,490 | 946.6 (0)              | >36,000 (10)        | 3,498     | 829.1 (0)               | >36,000 (10)        | 3,519 | 790.8 (0)            |
| Class             | Greedy-lookback     |       |                        | Greedy-lookahead    |           |                         |                     |       |                      |
|                   | $\sum_{\text{cpu}}$ | $t_b$ | cost                   | $\sum_{\text{cpu}}$ | $t_b$     | cost                    | $\sum_{\text{cpu}}$ | $t_b$ | cost                 |
| g1                | 1 (0)               | 0     | 36.9 (4)               | 1 (0)               | 0         | <b>36.8</b> (5)         |                     |       |                      |
| g2                | 21 (0)              | 0.7   | 28.8 (8)               | 31 (0)              | 1         | <b>28.7</b> (9)         |                     |       |                      |
| g3                | 94 (0)              | 7     | 145 (0)                | 126 (0)             | 9         | 146.3 (0)               |                     |       |                      |
| g4                | >31,813 (6)         | 1,419 | 121 (0)                | >32,846 (9)         | 1,778     | 121.3 (0)               |                     |       |                      |
| g5                | >27,808 (6)         | 920   | 126.5 <sub>8</sub> (1) | >30,233 (7)         | 1,743     | 127.1 (1)               |                     |       |                      |
| g6                | >22,165 (5)         | 553   | 186.5 <sub>8</sub> (0) | >23,521 (5)         | 508       | 189.88 <sub>8</sub> (0) |                     |       |                      |
| g7                | >36,000 (10)        | 1,468 | 183.4 (0)              | >36,000 (10)        | 1,549 (0) | <b>179.3</b> (0)        |                     |       |                      |
| g8                | 1,014 (0)           | 80    | 443.1 (0)              | 930 (0)             | 78        | 441.4 (0)               |                     |       |                      |
| g9                | >30,913 (8)         | 1,551 | 372.4 (0)              | >31,403 (8)         | 1,513     | <b>372.1</b> (0)        |                     |       |                      |
| g10               | >26,193 (6)         | 902   | 775.3 (0)              | >28,358 (6)         | 770       | 773.8 (0)               |                     |       |                      |
| g11               | >36,000 (10)        | 2,093 | <b>419.9</b> (0)       | >36,000 (10)        | 2,522     | 422.7 (0)               |                     |       |                      |

Note. pMD = p-median with distance constraints.

The lowest mean objective value for each problem class is highlighted in bold. In case of ties, the value of the best-performing setting is highlighted.

with the heuristics providing better performance in most classes, with the notable exception of greedy-lookahead. Overall, the greedy-lookback heuristic seems to be the best option in grid problems. As seen in Table 8, the pairwise comparison between greedy-lookback and *lexico* classifies 132 instances in the *#imp* column, and 75 in the *#wor* column. In contrast, 86 instances fall in the *#imp* column for greedy-lookahead and 103 in the *#wor* column. We conjecture that this discrepancy is most likely due to the small number of solutions present in such problems, resulting in greedy-lookahead being often misled, but this requires further investigation.

In summary, greedy-lookback and greedy-lookahead demonstrated the best performance among the proposed value ordering heuristics, with respect to both solution quality and run-time. Our results from p-median-based classes indicate that when faced with problems that have many feasible solutions, and therefore a higher probability of many promising paths, a heuristic such as greedy-lookahead can be quite useful. However, using such a heuristic can sometimes be both expensive and misleading when dealing with highly constrained problems (e.g., *grid*<sub>2</sub> classes), as it may make false decisions that direct the solver toward unproductive exploration of paths that do not lead to solutions. On the other hand, greedy-lookback is quite robust, as it dominates the standard *lexico* option in both types of problems, it is the best heuristic in grid-based classes and is not far from greedy-lookahead in p-median-based ones.

**7.2.3 Sampling.** Table 9 demonstrates the effect of the sampling technique on CPU times and solution quality. Four different settings, with varying numbers of samples, are compared. For each number of samples, we give the total CPU times ( $\sum_{\text{cpu}}$ ), mean cost values (cost), and number of optimal solutions discovered (*#opt*). Time-outs are presented as in Tables 2 and 3. In these experiments, we used some classes of problems where *CP<sub>h</sub>* demonstrated better performance than Gurobi and OR-Tools, such as g2 and g3. Additionally, we have selected classes where our solver exhibits worse results (e.g., pmed33), in order to evaluate the ability of the sampling technique to improve solution quality on such hard problems. Regarding the *CP<sub>h</sub>* solver's settings, we use the greedy-lookback value ordering heuristic, as it demonstrated the most robust performance among all the tested heuristics. Due to technical reasons, the experiments reported in Table 9, including those with sample size 1, were conducted on a different machine than those of Tables 2 to 8. This is not a

**Table 7.** Comparing Value Ordering Heuristics on Grid Based pMD Problems (GRID<sub>2</sub>).

| GRID <sub>2</sub>  |                   |       |                         |                   |       |                                |                   |       |                   |
|--------------------|-------------------|-------|-------------------------|-------------------|-------|--------------------------------|-------------------|-------|-------------------|
| Class              | Lexico            |       |                         | Greedy-minmax     |       |                                | Greedy-minsum     |       |                   |
|                    | $\sum \text{cpu}$ | $t_b$ | cost                    | $\sum \text{cpu}$ | $t_b$ | cost                           | $\sum \text{cpu}$ | $t_b$ | cost              |
| g1 <sup>(1)</sup>  | 4 (0)             | 0     | 57.05 (7)               | 4 (0)             | 0     | 56.95 (4)                      | 4 (0)             | 0     | 56.79 (9)         |
| g2 <sup>(1)</sup>  | 83 (0)            | 2     | 72.8 (7)                | 64 (0)            | 2     | 73.25 (5)                      | 88 (0)            | 3     | 73.2 (3)          |
| g3 <sup>(1)</sup>  | 136 (0)           | 3     | 40.9 (18)               | 127 (0)           | 2     | 40.9 (18)                      | 138 (0)           | 3     | <b>40.7</b> (20)  |
| g4 <sup>(1)</sup>  | >29,560 (4)       | 804   | 98.95 (8)               | >29,171 (4)       | 785   | 98.5 (7)                       | >29,609 (4)       | 600   | 98.45 (7)         |
| g5 <sup>(1)</sup>  | >63,462 (16)      | 1,209 | 136 (0)                 | >62,686 (16)      | 1,440 | 134.05 (0)                     | >62,864 (16)      | 1,464 | <b>133.4</b> (0)  |
| g6 <sup>(1)</sup>  | >37,210 (7)       | 1,063 | 298.7 (3)               | >37,865 (7)       | 652   | 298.2 (1)                      | >37,927 (7)       | 632   | <b>297.75</b> (3) |
| g7 <sup>(2)</sup>  | 3 (0)             | 0     | 56.05 (7)               | 2 (0)             | 0     | 55.7 (9)                       | 2 (0)             | 0     | <b>55.4</b> (12)  |
| g8 <sup>(2)</sup>  | >10,159 (1)       | 392   | 183.75 (1)              | >12,436 (1)       | 315   | 184.1 (0)                      | >11,984 (1)       | 397   | 183.65 (0)        |
| g9 <sup>(2)</sup>  | >26,960 (5)       | 747   | 176.55 (0)              | >28,698 (4)       | 565   | 176.25 (0)                     | >33,531 (4)       | 879   | 176.7 (0)         |
| g10 <sup>(2)</sup> | >39,189 (9)       | 857   | 46.41 <sub>17</sub> (5) | >34,884 (9)       | 784   | 47.55 (5)                      | >36,641 (8)       | 880   | <b>46.95</b> (6)  |
|                    |                   |       |                         |                   |       |                                |                   |       |                   |
| Class              | Greedy-lookback   |       |                         | Greedy-lookahead  |       |                                |                   |       |                   |
|                    | $\sum \text{cpu}$ | $t_b$ | cost                    | $\sum \text{cpu}$ | $t_b$ | cost                           | $\sum \text{cpu}$ | $t_b$ | cost              |
| g1 <sup>(1)</sup>  | 4 (0)             | 0     | <b>56.68</b> (9)        | 6 (0)             | 0.2   | 56.89 (7)                      |                   |       |                   |
| g2 <sup>(1)</sup>  | 85 (0)            | 3     | 73.35 (3)               | 89 (0)            | 1     | <b>72.75</b> (8)               |                   |       |                   |
| g3 <sup>(1)</sup>  | 128 (0)           | 2     | 40.75 (19)              | 186 (0)           | 4     | 40.7 (20)                      |                   |       |                   |
| g4 <sup>(1)</sup>  | >29,409 (4)       | 460   | <b>98.35</b> (8)        | >36,430 (8)       | 968   | 99.1 (7)                       |                   |       |                   |
| g5 <sup>(1)</sup>  | >62,857 (16)      | 1,383 | <b>133.6</b> (1)        | >66,223 (16)      | 1,546 | 135.6 (0)                      |                   |       |                   |
| g6 <sup>(1)</sup>  | >37,404 (7)       | 447   | 298.35 (2)              | >46,409 (9)       | 1,114 | 300.45 (3)                     |                   |       |                   |
| g7 <sup>(2)</sup>  | 2 (0)             | 0     | 55.8 (10)               | 3 (0)             | 0     | 55.85 (9)                      |                   |       |                   |
| g8 <sup>(2)</sup>  | >10,951 (2)       | 223   | <b>183.15</b> (2)       | >18,896 (4)       | 347   | 181.5 <sub>18</sub> (0)        |                   |       |                   |
| g9 <sup>(2)</sup>  | >30,860 (5)       | 712   | <b>175.9</b> (0)        | >30,068 (6)       | 788   | 176.35 (1)                     |                   |       |                   |
| g10 <sup>(2)</sup> | >36,528 (9)       | 588   | 47.45 (5)               | >39,132 (9)       | 740   | <b>46.75</b> <sub>16</sub> (5) |                   |       |                   |

Note. pMD = p-median with distance constraints.

The lowest mean objective value for each problem class is highlighted in bold. In case of ties, the value of the best-performing setting is highlighted.

**Table 8.** Improvements Overall all Instances Between Lexico and Every Greedy Heuristic in all pMD Categories.

| Class             | Greedy-minmax   |          |           |           |           | Greedy-minsum    |           |           |           |            |
|-------------------|-----------------|----------|-----------|-----------|-----------|------------------|-----------|-----------|-----------|------------|
|                   | #imp            | #cost    | #time     | #eq       | #wor      | #imp             | #cost     | #time     | #eq       | #wor       |
| $ CL  \geq  P $   | 54              | 4        | 4         | 29        | 29        | 54               | 1         | 14        | 29        | 22         |
| $ P  >  CL $      | 65              | 3        | 23        | 7         | 22        | 86               | 1         | 19        | 3         | 11         |
| Total (pmed)      | <b>119</b>      | <b>7</b> | <b>27</b> | <b>36</b> | <b>51</b> | <b>140</b>       | <b>2</b>  | <b>33</b> | <b>32</b> | <b>33</b>  |
| GRID <sub>1</sub> | 56              | 1        | 6         | 18        | 29        | 56               | 1         | 8         | 16        | 29         |
| GRID <sub>2</sub> | 67              | 1        | 9         | 69        | 54        | 74               | 3         | 7         | 68        | 48         |
| Total (Grid)      | <b>123</b>      | <b>2</b> | <b>15</b> | <b>87</b> | <b>83</b> | <b>130</b>       | <b>4</b>  | <b>15</b> | <b>84</b> | <b>77</b>  |
| Class             | Greedy-lookback |          |           |           |           | Greedy-lookahead |           |           |           |            |
|                   | #imp            | #cost    | #time     | #eq       | #wor      | #imp             | #cost     | #time     | #eq       | #wor       |
| $ CL  \geq  P $   | 61              | 1        | 16        | 28        | 14        | 66               | 1         | 6         | 34        | 13         |
| $ P  >  CL $      | 88              | 0        | 23        | 3         | 6         | 102              | 1         | 9         | 2         | 6          |
| Total (pmed)      | <b>149</b>      | <b>1</b> | <b>39</b> | <b>31</b> | <b>20</b> | <b>168</b>       | <b>2</b>  | <b>15</b> | <b>36</b> | <b>19</b>  |
| GRID <sub>1</sub> | 54              | 1        | 8         | 21        | 26        | 49               | 1         | 9         | 33        | 18         |
| GRID <sub>2</sub> | 78              | 4        | 4         | 65        | 49        | 37               | 15        | 2         | 61        | 85         |
| Total (Grid)      | <b>132</b>      | <b>5</b> | <b>12</b> | <b>86</b> | <b>75</b> | <b>86</b>        | <b>16</b> | <b>11</b> | <b>94</b> | <b>103</b> |

Note. pMD = p-median with distance constraints.

The total values for each metric in the pmed and Grid categories are highlighted in bold.



**Table 9.** The Effect of Sampling on Central Processing Unit (CPU) Times and Solution Quality.

| Class              | s=1                 |                     |      | s=5                 |                     |      | s=10                |                     |      | s=20                |                     |      |
|--------------------|---------------------|---------------------|------|---------------------|---------------------|------|---------------------|---------------------|------|---------------------|---------------------|------|
|                    | $\sum_{\text{cpu}}$ | cost                | #opt | $\sum_{\text{cpu}}$ | cost                | #opt | $\sum_{\text{cpu}}$ | cost                | #opt | $\sum_{\text{cpu}}$ | cost                | #opt |
| $ CL  \geq  P $    |                     |                     |      |                     |                     |      |                     |                     |      |                     |                     |      |
| pmed10             | >29,017 (8)         | 1,628.3             | 0    | >30,906 (8)         | 1,622.7             | 1    | >32,857 (9)         | 1,626.4             | 0    | >33,219 (9)         | 1,633.3             | 0    |
| pmed15             | >36,000 (10)        | 2,114.7             | 0    | >36,000 (10)        | 2,115               | 0    | >36,000 (10)        | 2,114.3             | 0    | >36,000 (10)        | 2,116.8             | 0    |
| pmed21             | 3 (0)               | 9,623.9             | 8    | 7 (0)               | 9,614.9             | 9    | 8 (0)               | 9,614.9             | 9    | 9 (0)               | 9,614.9             | 9    |
| pmed31             | 10 (0)              | 10,456.4            | 6    | 23 (0)              | 10,442.1            | 9    | 27 (0)              | 10,441              | 10   | 29 (0)              | 10,441              | 10   |
| pmed33             | >36,000 (10)        | 3,484.1             | 0    | >36,000 (10)        | 3,501.9             | 0    | >36,000 (10)        | 3,507.5             | 0    | >36,000 (10)        | 3,510.5             | 0    |
| pmed38             | 85 (0)              | 7,156.7             | 2    | 330 (0)             | 7,138.5             | 6    | 341 (0)             | 7,134.3             | 8    | 356 (0)             | 7,135.5             | 8    |
| $ P  >  CL $       |                     |                     |      |                     |                     |      |                     |                     |      |                     |                     |      |
| pmed10             | >25,030 (5)         | 369.2               | 0    | >35,425 (9)         | 373.7               | 0    | >34,525 (9)         | 374.4               | 0    | >36,000 (10)        | 375.2               | 0    |
| pmed15             | >36,000 (10)        | 511.3               | 2    | >36,000 (10)        | 518.2               | 0    | >36,000 (10)        | 518.6               | 0    | >36,000 (10)        | 519.4               | 0    |
| pmed21             | 48 (0)              | 1,794.6             | 3    | 160 (0)             | 1,788.9             | 9    | 202 (0)             | 1,789.6             | 7    | 265 (0)             | 1,789.4             | 7    |
| pmed31             | 138 (0)             | 1,991.1             | 6    | 402 (0)             | 1,988               | 8    | 510 (0)             | 1,988.5             | 8    | 686 (0)             | 1,987.9             | 9    |
| pmed33             | >36,000 (10)        | 970.9               | 0    | >36,000 (10)        | 977                 | 0    | >36,000 (10)        | 978.7               | 0    | >36,000 (10)        | 979.3               | 0    |
| pmed38             | 143 (0)             | 2,488.3             | 5    | 507 (0)             | 2,481.1             | 6    | 637 (0)             | 2,480               | 7    | 766 (0)             | 2,478.8             | 9    |
| GRID <sub>1</sub>  |                     |                     |      |                     |                     |      |                     |                     |      |                     |                     |      |
| g2                 | 27 (0)              | 28.8                | 8    | 55 (0)              | 28.6                | 10   | 81 (0)              | 28.7                | 9    | 127 (0)             | 28.6                | 10   |
| g3                 | 121 (0)             | 145                 | 0    | 991 (0)             | 143.2               | 0    | 2,987 (0)           | 142.1               | 0    | 8,554 (0)           | 141.8               | 1    |
| g4                 | >32,830 (9)         | 124                 | 0    | >32,895 (9)         | 119.56 <sub>9</sub> | 0    | >32,934 (9)         | 119.89 <sub>9</sub> | 0    | >32,920 (9)         | 120.22 <sub>9</sub> | 0    |
| g5                 | >29,557 (6)         | 126.5 <sub>8</sub>  | 1    | >29,745 (7)         | 126.25 <sub>8</sub> | 1    | >29,802 (7)         | 126.25 <sub>8</sub> | 1    | >29,803 (7)         | 126.13 <sub>8</sub> | 1    |
| g6                 | >23,210 (5)         | 187.25 <sub>8</sub> | 0    | >23,662 (5)         | 187 <sub>8</sub>    | 0    | >23,766 (5)         | 187 <sub>8</sub>    | 0    | >23,523 (5)         | 186.88 <sub>8</sub> | 0    |
| g8                 | 1,299 (0)           | 443.1               | 0    | >9,402 (1)          | 436.3               | 0    | >15,740 (3)         | 440.2               | 0    | >20,523 (5)         | 439.5               | 0    |
| GRID <sub>2</sub>  |                     |                     |      |                     |                     |      |                     |                     |      |                     |                     |      |
| g1 <sup>(1)</sup>  | 5 (0)               | 56.68               | 9    | 21 (0)              | 56.21               | 15   | 45 (0)              | 56.16               | 17   | 107 (0)             | 56                  | 19   |
| g2 <sup>(1)</sup>  | 111 (0)             | 73.35               | 3    | 409 (0)             | 72                  | 9    | 837 (0)             | 71.6                | 12   | 1,746 (0)           | 71.3                | 14   |
| g7 <sup>(2)</sup>  | 3 (0)               | 55.8                | 10   | 12 (0)              | 55.15               | 15   | 33 (0)              | 55.05               | 16   | 83 (0)              | 54.95               | 18   |
| g8 <sup>(2)</sup>  | >12,107 (2)         | 183.15              | 2    | >17,128 (4)         | 182.75              | 1    | >16,889 (4)         | 181.6               | 1    | >26,912 (3)         | 181.6               | 1    |
| g9 <sup>(2)</sup>  | >34,660 (6)         | 176.1               | 0    | >45,122 (11)        | 176.3               | 0    | >48,410 (11)        | 176.95              | 1    | >54,378 (12)        | 176.9               | 0    |
| g10 <sup>(2)</sup> | >37,977 (9)         | 47.55               | 5    | >48,800 (10)        | 47.2                | 8    | >56,993 (13)        | 47.65               | 6    | >60,473 (14)        | 47.80               | 5    |

problem, as our goal regarding the sampling technique is to investigate how it affects the CP solver's performance in terms of the quality of solutions and the run time. The machine has 20 Intel(R) Xeon(R) Gold 6230 CPU cores at 2.10 GHz and 32 GB of main memory.

As expected, as the number of samples grows, so does the CPU time. This is due to the weaker pruning that is entailed by the lower bounds being computed, which means that larger portions of the search space are explored, and also to the overhead caused by the repeated runs of the greedy heuristic while computing the bounds. This overhead can have a significant impact on problems that  $CP_h$  found difficult to solve and timed out, and can result in worse solutions. This is particularly indicated by the results of the pmed15 and pmed33 classes in p-median-based problems. On the other hand, there is a significant improvement in classes that were relatively easy for the solver. In classes such as pmed21, pmed38 and g2, g8 from both the grid<sub>1</sub> and grid<sub>2</sub> categories, the solver is able to obtain better solutions and, especially in p-median classes, more optimal ones. Even with a small number of samples (e.g.,  $s = 5$ )  $CP_h$  is able to quite often discover many more optimal solutions, albeit by increasing the run times.


To conclude, in small problems, or when the cost of the solution is of prime importance, compared to the run time, the sampling method can be useful, as it helps to obtain solutions of better quality. However, in large problems, or when the run time is crucial, the sampling method seems to add an unnecessary CPU time overhead.


## 8 Conclusion


We have investigated the pMD, a variant of the p-median problem where distance constraints exist between facilities and between facilities and clients. This problem can be used to model the location of semi-obnoxious facilities. We proposed

ILP and CP approaches toward modeling and solving pMDs. We also demonstrated how a recent heuristic CP approach for the p-dispersion problem with distance constraints can be adapted and applied to the pMD. Results showed that incorporating a greedy branch pruning heuristic into a CP solver along with employing a simpler model of the problem, provides a solution tool that displays more stable performance across problem classes with different characteristics compared to standard ILP and CP solvers, despite not being the best option in all the classes tried. Additionally, we proposed four specialized value ordering heuristics designed to guide the search toward promising directions by exploiting information of the current search state. Results showed that in many cases these heuristics in combination with the branch pruning method were able to provide solutions of better quality compared to lexicographic value ordering, as well as better run times. Finally, we showed that the estimation of the greedy bounding heuristic can be more precise (although sacrificing CPU time performance) by trying different orderings for the variables it considers during the computation of the bound. In this way, more optimal solutions can be obtained and the objective cost can be reduced. As for future work, it would be very interesting to consider real-world case studies for the pMD and to investigate the applicability of the heuristic CP approach to other types of constraint optimization problems from domains such as scheduling and resource allocation.

### ORCID iDs

Panteleimon Iosif  <https://orcid.org/0009-0001-4589-3346>

Nikolaos Ploskas  <https://orcid.org/0000-0001-5876-9945>

Kostas Stergiou  <https://orcid.org/0000-0002-5702-9096>

### Funding

The authors received no financial support for the research, authorship, and/or publication of this article.

### Declaration of Conflicting Interests

The authors declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

### Notes

1. That is, nodes that are adjacent in the original grid.
2. We include this number for reasons of presentation uniformity, as the optimal is not known for all instances of the other generation models.

### References

- Balinski, M. (1965). Integer programming: Methods, uses, computations. *Management Science*, 12(3), 253–313. <https://doi.org/10.1287/mnsc.12.3.253>
- Beasley, J. E. (1985). A note on solving large p-median problems. *European Journal of Operational Research*, 21(2), 270–273. [https://doi.org/10.1016/0377-2217\(85\)90040-2](https://doi.org/10.1016/0377-2217(85)90040-2)
- Beltran, C., Tadonki, C., & Vial, J. P. (2006). Solving the p-median problem with a semi-lagrangian relaxation. *Computational Optimization and Applications*, 35(2), 239–260. <https://doi.org/10.1007/s10589-006-6513-6>
- Berman, O., & Huang, R. (2008). The minimum weighted covering location problem with distance constraints. *Computers and Operations Research*, 35(12), 356–372. <https://doi.org/10.1016/j.cor.2006.03.003>
- Bessiere, C. (2006). Constraint propagation. In *Foundations of artificial intelligence* (Vol. 2, pp. 29–83). Elsevier. [https://doi.org/10.1016/S1574-6526\(06\)80007-6](https://doi.org/10.1016/S1574-6526(06)80007-6)
- Boussemart, F., Hemery, F., Lecoutre, C., & Sais, L. (2004). Boosting systematic search by weighting constraints. In *Proceedings of the 16th European conference on artificial intelligence*, ECAI'04 (pp. 146–150). IOS Press.
- Brimberg, J., & Juel, H. (1998). A minisum model with forbidden regions for locating a semi-desirable facility in the plane. *Location Science*, 6(1), 109–120. [https://doi.org/10.1016/S0966-8349\(98\)00050-3](https://doi.org/10.1016/S0966-8349(98)00050-3)
- Cambazard, H., Mehta, D., O'Sullivan, B., & Quesada, L. (2012). A computational geometry-based local search algorithm for planar location problems. In N. Beldiceanu, N. Jussien, & É. Pinson (Eds.), *Integration of AI and OR techniques in constraint programming for combinatorial optimization problems* (pp. 97–112). Springer. [https://doi.org/10.1007/978-3-642-29828-8\\_7](https://doi.org/10.1007/978-3-642-29828-8_7)
- Carriozosa, E., & Plastria, F. (1999). Location of semi-obnoxious facilities. *Studies in Locational Analysis*, 12(1999), 1–27. [https://www.researchgate.net/profile/Frank-Plastria-2/publication/246714839\\_Location\\_of\\_semi-obnoxious\\_facilities/links/543ec2810cf21c84f23cb2e3/Location-of-semi-obnoxious-facilities.pdf](https://www.researchgate.net/profile/Frank-Plastria-2/publication/246714839_Location_of_semi-obnoxious_facilities/links/543ec2810cf21c84f23cb2e3/Location-of-semi-obnoxious-facilities.pdf)
- Chaudhry, S. S., McCormick, S. T., & Moon, I. D. (1986). Locating independent facilities with maximum weight: Greedy heuristics. *International Journal of Management Science*, 14(5), 383–389. [https://doi.org/10.1016/0305-0483\(86\)90079-4](https://doi.org/10.1016/0305-0483(86)90079-4)
- Church, R. L. (2003). Cobra: A new formulation of the classic p-median location problem. *Annals of Operations Research*, 122, 103–120. <https://doi.org/10.1023/A:1026142406234>

- Church, R. L., & Meadows, M. E. (1977). Results of a new approach to solving the p-median problem with maximum distance constraints. *Geographical Analysis*, 9(4), 364–378. <https://doi.org/10.1111/j.1538-4632.1977.tb00589.x>
- Chvatal, V. (1979). A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3), 233–235. <https://doi.org/10.1287/moor.4.3.233>
- Comley, W. J. (1995). The location of ambivalent facilities: Use of a quadratic zero-one programming algorithm. *Applied Mathematical Modeling*, 19(1), 26–29. [https://doi.org/10.1016/0307-904X\(94\)00004-P](https://doi.org/10.1016/0307-904X(94)00004-P)
- Cornuejols, G., Nemhauser, G. L., & Wolsey, L. A. (1980). A canonical representation of simple plant location problems and its applications. *SIAM Journal on Algebraic Discrete Methods*, 1(3), 261–272. <https://doi.org/10.1137/0601030>
- Densham, P. J., & Rushton, G. (1992). Strategies for solving large location-allocation problems by heuristic methods. *Environment and Planning A*, 24(2), 289–304. <https://doi.org/10.1068/a240289>
- Drezner, T., Drezner, Z., & Schöbel, A. (2018). The Weber obnoxious facility location model: A big arc small arc approach. *Computers and Operations Research*, 98, 240–250. <https://doi.org/10.1016/j.cor.2018.06.006>
- Drezner, Z., Kalczyński, P., & Salhi, S. (2019). The planar multiple obnoxious facilities location problem: A Voronoi based heuristic. *Omega*, 87, 105–116. <https://doi.org/10.1016/j.omega.2018.08.013>
- Fazel-Zarandi, M. M., & Beck, J. C. (2009). Solving a location-allocation problem with logic-based benders' decomposition. In I. P. Gent (Ed.), *Principles and practice of constraint programming—CP 2009* (pp. 344–351). Springer. [https://doi.org/10.1007/978-3-642-04244-7\\_28](https://doi.org/10.1007/978-3-642-04244-7_28)
- Guns, T. (2019). Increasing modeling language convenience with a universal n-dimensional array, CPython as python-embedded example. In *Proceedings of the 18th workshop on constraint modelling and reformulation at CP (ModRef 2019)* (Vol. 19). [https://modref.github.io/papers/ModRef2019\\_Increasing%20modeling%20language%20convenience%20with%20a%20universal%20ndimensional%20array.pdf](https://modref.github.io/papers/ModRef2019_Increasing%20modeling%20language%20convenience%20with%20a%20universal%20ndimensional%20array.pdf)
- Gurobi Optimization, LLC (2023). Gurobi optimizer reference manual. <https://www.gurobi.com>.
- Hakimi, L. (1964). Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12(3), 450–459. <https://doi.org/10.1287/opre.12.3.450>
- Hakimi, L. (1965). Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations Research*, 13(3), 462–475. <https://doi.org/10.1287/opre.13.3.462>
- Iosif, P., Ploskas, N., & Stergiou, K. (2024a). A heuristic constraint programming approach to the p-median problem with distance constraints. In *Proceedings of the 13th Hellenic conference on artificial intelligence, SETN '24* (Article 7, pp. 1–10). Association for Computing Machinery.
- Iosif, P., Ploskas, N., Stergiou, K., & Tsouros, D. C. (2024b). A CP/LS heuristic method for maxmin and minmax location problems with distance constraints. In P. Shaw (Ed.), *30th international conference on principles and practice of constraint programming, CP 2024*, September 2–6, 2024, Girona, Spain, LIPIcs (Vol. 307, pp. 14:1–14:21). Schloss Dagstuhl—Leibniz-Zentrum für Informatik. <https://doi.org/10.4230/LIPIcs.CP.2024.14>
- Khumawala, B. M. (1973). An efficient algorithm for the p-median problem with maximum distance constraints. *Geographical Analysis*, 5(4), 309–321. <https://doi.org/10.1111/j.1538-4632.1973.tb00493.x>
- Krarup, J., Pisinger, D., & Plastria, F. (2002). Discrete location problems with push-pull objectives. *Discrete Applied Mathematics*, 123(1–3), 363–378. [https://doi.org/10.1016/S0166-218X\(01\)00346-8](https://doi.org/10.1016/S0166-218X(01)00346-8)
- Kuehn, A. A., & Hamburger, M. J. (1963). A heuristic program for locating warehouses. *Management Science*, 9, 643–666. <https://doi.org/10.1287/mnsc.9.4.643>
- Mackworth, A. K. (1977). Consistency in networks of relations. *Artificial Intelligence*, 8(1), 99–118. [https://doi.org/10.1016/0004-3702\(77\)90007-8](https://doi.org/10.1016/0004-3702(77)90007-8)
- Maier, A., & Hamacher, H. W. (2019). Complexity results on planar multifacility location problems with forbidden regions. *Mathematical Methods of Operations Research*, 89, 433–484. <https://doi.org/10.1007/s00186-019-00670-0>
- Mladenović, N., Brimberg, J., Hansen, P., & Moreno-Pérez, J. A. (2007). The p-median problem: A survey of metaheuristic approaches. *European Journal of Operational Research*, 179(3), 927–939. <https://doi.org/10.1016/j.ejor.2005.05.034>
- Moon, D. I., & Chaudhry, S. S. (1984). An analysis of network location problems with distance constraints. *Management Science*, 30(3), 290–307. <https://doi.org/10.1287/mnsc.30.3.290>
- Moon, I. D., & Papayanopoulos, L. (1991). Minimax location of two facilities with minimum separation: Interactive graphical solutions. *Journal of the Operations Research Society*, 42, 685–694. <https://doi.org/10.1057/jors.1991.134>
- Orloff, C. S. (1977). A theoretical model of net accessibility in public facility location. *Geographical Analysis*, 9, 244–256. <https://doi.org/10.1111/j.1538-4632.1977.tb00577.x>
- Ortigosa, P. M., Hendrix, E. M., & Redondo, J. L. (2015). On heuristic bi-criterion methods for semi-obnoxious facility location. *Computational Optimization and Applications*, 61, 205–217. <https://doi.org/10.1007/s10589-014-9709-1>
- O.-T. Development Team (2024)OR-Tools, CP-SAT solver. <https://developers.google.com/optimization/cp/cp-solver>.

- Ploskas, N., & Stergiou, K. (2022). Integer programming models for the semi-obnoxious p-median problem. <https://arxiv.org/abs/2207.05590>.
- Ploskas, N., Stergiou, K., & Tsouros, D. C. (2023). The p-dispersion problem with distance constraints. In R. H. C. Yap (Ed.), *29th International conference on principles and practice of constraint programming (CP 2023)*, Leibniz International Proceedings in Informatics (LIPIcs) (Vol. 280, pp. 30:1–30:18). Schloss Dagstuhl—Leibniz-Zentrum für Informatik. <https://doi.org/10.4230/LIPIcs.CP.2023.30>
- ReVelle, C. S., & Swain, R. W. (1970). Central facilities location. *Geographical Analysis*, 2(1), 30–42. <https://doi.org/10.1111/j.1538-4632.1970.tb00142.x>
- Rosing, K. E., ReVelle, C., & Rosing-Vogelaar, H. (1979). The p-median and its linear programming relaxation: An approach to large problems. *Journal of the Operational Research Society*, 30(9), 815–823. <https://doi.org/10.1057/jors.1979.192>
- Rossi, F., Van Beek, P., & Walsh, T. (2006). *Handbook of constraint programming*. Elsevier.
- Sabin, D., & Freuder, E. C. (1994). Contradicting conventional wisdom in constraint satisfaction. In A. Borning (Ed.), *Principles and practice of constraint programming* (pp. 10–20). Springer. [https://doi.org/10.1007/3-540-58601-6\\_86](https://doi.org/10.1007/3-540-58601-6_86)
- Sorkhabi, S. Y. D., Romero, D. A., Beck, J. C., & Amon, C. H. (2018). Constrained multi-objective wind farm layout optimization: Novel constraint handling approach based on constraint programming. *Renewable Energy*, 126(C), 341–353. <https://doi.org/10.1016/j.renene.2018.03.053>
- Tansel, B. C., Francis, R. L., Lowe, T. J., & Chen, M. (1982). Duality and distance constraints for the nonlinear p-center problem and covering problem on a tree network. *Operations Research*, 30(4), 725–744. <https://doi.org/10.1287/opre.30.4.725>
- Tutunchi, G. K., & Fathi, Y. (2019). Effective methods for solving the bi-criteria p-center and p-dispersion problem. *Computers & Operations Research*, 101, 43–54. <https://doi.org/10.1016/j.cor.2018.08.009>
- Welch, S., & Salhi, S. (1997). The obnoxious p facility network location problem with facility interaction. *European Journal of Operations Research*, 102, 302–319. [https://doi.org/10.1016/S0377-2217\(97\)00111-2](https://doi.org/10.1016/S0377-2217(97)00111-2)
- Yapicioglu, H., Smith, A. E., & Dozier, G. (2007). Solving the semi-desirable facility location problem using bi-objective particle swarm. *European Journal of Operational Research*, 177(2), 733–749. <https://doi.org/10.1016/j.ejor.2005.11.020>