A CP/LS Heuristic Method for Maxmin and Minmax Location Problems with Distance Constraints

Panteleimon Iosif
University of Western Macedonia, Kozani, Greece

Nikolaos Ploskas
University of Western Macedonia, Kozani, Greece

Kostas Stergiou
University of Western Macedonia, Kozani, Greece

Dimosthenis C. Tsouros
KU Leuven, Belgium

Abstract

In facility location problems we seek to locate a set of facilities in an area, where clients may be present, so that some criterion is optimized. For instance, in the p-center problem we seek to minimize the maximum distance between any client and its closest facility, whereas in the p-dispersion problem we seek to maximize the minimum distance between any two facilities. Hence, in the former we have a minmax objective, whereas in the latter we have a maxmin objective. Recently, a variant of p-dispersion where distance constraints exist between facilities was studied from a CP and ILP perspective. An incomplete CP solver that uses a greedy heuristic to prune branches was shown to significantly outperform Gurobi and OR-Tools in terms of execution time, although it failed to discover optimal or near-optimal solutions in many instances. We enhance this work in two directions, regarding the effectiveness and the applicability of the approach. We first show how local search can be used to obtain better estimations of the bound at each node, resulting in more focused pruning, which allows for optimal or near-optimal solutions to be discovered in many more instances. Then, we demonstrate how the framework can be applied on the p-center problem with distance constraints, comparing it to ILP and CP models implemented in Gurobi and OR-Tools, respectively.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases Constraint Programming, Local Search, facility location, distance constraints, optimization

Digital Object Identifier 10.4230/LIPIcs.CP.2024.14

1 Introduction

Facility location problems are widely studied in OR, AI, computational geometry and other disciplines. In such problems we seek to locate a set of facilities in an area, where clients may be present, so that some criterion is optimized. The optimization criterion largely depends on the type of facilities to be located. When the facilities have beneficial properties (e.g. pharmacies), we want to locate them close to clients. In contrast, when the facilities are (ob)noxious, i.e. 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. For instance, we may wish to locate gas stations close to clients for their convenience, but not too close because of the pollution and potential danger they are associated with.

The p-center and the p-dispersion problems have been widely studied when modeling beneficial and obnoxious scenarios, respectively [29, 38, 34]. In the former we seek to locate p facilities in an area where clients are present, so that the maximum distance between any client and its closest facility is minimized. In the latter we seek to locate p (ob)noxious facilities, so that the minimum distance between any two facilities is maximized. Hence, in p-center we have a minmax objective, whereas in p-dispersion we have a maxmin objective. The relationship between these two problems has been identified as early as 1977 [45].

In practice, p-center problems occur when the close proximity of facilities to clients is desirable, and in addition, we seek a fair location, in the sense that no client is too far from the facilities. On the other hand, p-dispersion problems occur whenever a close proximity of facilities is dangerous or for other reasons undesirable [34, 20, 37].

Recently, a variant of p-dispersion that includes distance constraints between facilities was studied from a CP and ILP perspective [41]. Distance constraints in location problems were first studied by Moon and Chaudhry [38] and can stem from operational needs and regulations, such as clearance distances for safe chemical storage [1], separation distances between packages containing radioactive materials [49] or portable fire extinguishers [50]. Also, by placing distance constraints between facilities and/or between facilities and clients, we can model the requirements that arise when trying to locate semi-obnoxious facilities [33].

Ploskas et al. [41] described a search method that uses a heuristic to prune branches, based on a estimation of the cost at each node. If the estimated bound, provided by a greedy assignment of the unassigned variables, is not higher than the cost of the incumbent solution then a fail is forced and the sub-tree below the current node is pruned. A solver that uses this method is naturally incomplete, but the solver of [41] was shown to be much faster than standard CP and ILP solvers. However, it was able to discover the optimal solution in only 2 out of the 82 MDPLIB benchmark instances for which the optimal solution is known, and was quite far from the optimal in many cases. This is because the greedy heuristic may often underestimate the cost, resulting in exceedingly high branch pruning, and the omission of the optimal and close-to-optimal solutions.

We extend the work of [41] in two directions, regarding effectiveness and applicability. We first enhance the heuristic used to prune the search space through the use of local search (LS). Specifically, at each node we compute the greedy assignment and use it as the initial solution to the relaxed problem obtained by not considering the distance constraints. Then we try to improve this assignment through the application of a variant of the best pairwise interchange heuristic for p-dispersion (a LS method) [22]. This results in better estimations of the bound, allowing for the discovery of 40 optimal and many near-optimal solutions, and resulting in better solutions being discovered in almost all instances for which the optimal is not known. We also add a second local search component that tries to improve any new solution being discovered by searching in the neighborhood of feasible solutions.

Secondly, we demonstrate that the entire concept of building an incomplete CP solver around a heuristic that estimates the cost at each node and accordingly prunes the search space, is also applicable in other location problems with distance constraints, using the p-center problem as a demonstration. As baseline methods, we introduce ILP and CP models for this problem and implement them in Gurobi and CP-SAT OR-Tools, respectively. We use similar reasoning to the p-dispersion problem to build a CP solver that uses a simple model of the problem and applies a local search method to estimate the bound at each node, and thereby prune branches, making the solver incomplete.

We experimented with p-center problems with distance constraints of two types. In the first one the locations of the clients and the potential facility sites are randomly placed in a grid, while in the second we use p-median benchmark instances [3] as basis. Problems of the

first type, that typically have few solutions, can be very hard for Gurobi, as it often does not discover any solution within 1 hour of cpu time, but mostly manageable by OR-Tools, except for the larger ones. In contrast, OR-Tools fares badly on problems of the second type, that typically have many solutions, while Gurobi finds most of these problems quite easy. But both face difficulties in handling very large problems because of the size of their models.

The incomplete CP solver's performance is much more robust. It is outperformed by OR-Tools, in terms of solution quality, in small/medium size grid-structured problems, but is much more efficient than Gurobi. It also discovers solutions in all instances, even the larger ones that are out of reach for the complete solvers. On the other hand, the solver outperforms OR-Tools in p-median based problems, finding many optimal or near-optimal solutions much faster. It can also be quite faster than Gurobi in some cases, especially when the number of facilities is small, but cannot compete in instances where we try to locate a large number of facilities in relatively few facility sites.

In the following we first review related work and define the two problems, focusing on the p-center variant which has not been considered before, in terms of solution methods. Hence, we propose ILP and CP models for this problem. Then we describe the use of local search to boost the effectiveness of a CP solver. Finally, we present experimental results.

2 Related Work

The p-center problem, which is $\mathcal{NP}$-hard in the general case, was originally proposed by Hakimi, along with the related p-median problem [29, 30]. Since then, various ILP formulations for the p-center problem have been proposed [13, 18, 6]. A review of exact and heuristic methods for the p-center problem is given in [6]. The p-dispersion problem, which is also $\mathcal{NP}$-hard on general networks for an arbitrary p [25], was originally mentioned by Shier [45]. However, the term p-dispersion first appeared in the analysis of location problems with distance constraints by Moon and Chaudhry [38]. The first ILP solution was proposed by Kuby [34] while a specialized algorithm and alternative ILP models followed [19, 44, 43].

Regarding distance constraints, Moon and Chaudhry were the first to systematically study them [38]. Both p-dispersion and p-center with distance constraints were mentioned as problems that can arise in real-life scenarios, but no approaches towards solving them were proposed. Recently, Dai et al. revisited the former as part of a study on circle dispersion in non-convex polygons [12]. Distance constraints have also been considered in the context of other location problems [9, 10, 32, 47, 8, 39, 11, 48]. For instance, Tansel et al. studied the distance constrained *p-center* problem for the case where the network is a tree [47], whereas Comley studied the problem of locating a small number of semi-obnoxious facilities that interact with each other as well as with other existing facilities [11]. More recently, Berman and Huang studied the problem of locating obnoxious facilities so as to minimize the total demand covered, so that no two facilities are closer than a pre-specified distance [4]. Drezner et al. studied obnoxious facility location problems with restrictions on the distance between facilities and demand points [16, 17].

There are very few CP-related methods for facility location problems [23, 7, 46, 41]. As mentioned above, we build on the work of [41] which is concerned with the p-dispersion problem. There are CP works outside facility location that are relevant to ours, as they too sacrifice the completeness of a CP solver to solve optimization problems faster [31, 36, 26]. However, these works typically do this through different ways, e.g. by adding extra constraints that may disallow solutions but cut off large portions of the search tree.

The *greedy* heuristic for p-dispersion (resp. p-center) solves the 2-dispersion problem first (resp. the 1-center problem) and then places the remaining facilities one by one, choosing the location point for the currently considered facility f that maximizes the minimum distance in the set of already placed facilities plus f (resp. minimizes the maximum distance between clients and their closest facility among the already located ones plus f) [35, 21]. The *best pairwise interchange* heuristic for p-dispersion starts with a random location of the facilities, then it finds the pair of facilities f_1,f_2 that are closest to each other, and then it finds the free location point v that by allocating either f_1 or f_2 to v, the value of the objective function is maximally improved. This is repeated until no further improvement is possible [22].

3 Problem Definition

We assume that p facilities in a set of facilities F are to be placed on p nodes of a weighted network G. Hence, we deal with discrete/network location problems. Each facility site can host at most one facility. In a *p-center with distance constraints problem* (pCD), a set CL of demand points (clients) to be serviced by the facilities is located at certain (known) nodes of G. In a *p-dispersion with distance constraints problem* (pDD), we are only interested in the dispersion of the facilities. In the following, we will use the terms demand points and clients interchangeably, and we will assume that the set of nodes in G where the clients are located is known, and so is the set of nodes P where the facilities can potentially be located. The weight w_{ij} of an edge denotes the symmetric service cost (typically the distance) between nodes i and j. We assume that once the facilities have been located in a pCD, each client will be serviced by its closest facility, and there are no capacity restrictions on the facilities.

Between each pair of facilities f_i and f_j there is a distance constraint $dis(f_i, f_j) > d_{ij}$ specifying that the distance $dis(f_i, f_j)$ between the points where the facilities f_i and f_j are located must be greater than d_{ij}, where d_{ij} is a constant. In a pCD there is also a distance constraint $dis(f_i, c_k) > dk_i$ between each facility f_i and any client c_k, specifying that the distance $dis(f_i, c_k)$ between the node where facility f_i is located and node k where client c_k is located must be greater than dk_i, where dk_i is a constant.

A common assumption in the relevant literature is that the distance bound d_{ij} is the same for all the constraints between facilities. This is reasonable when the facilities are homogeneous, and therefore in essence indistinguishable, but it is not always realistic, especially when the facilities have different properties, as for example in [1, 49]. In this paper we deal with the heterogeneous case where the distance bound may vary from constraint to constraint. However, we follow the homogeneity assumption with respect to clients, meaning that for a specific facility f_i, we assume that the minimum distance bound between its location and the locations of the clients is the same for all clients.

The distance between two points i and j can be given by the Euclidean distance, e.g. for the location of hazardous facilities, or by the shortest path in a street network, e.g. for the location of franchises, or by any other suitable metric. The methods we propose do not depend on any particular distance measure because, as is common in the literature, we assume that the pairwise distances between all possible client and facility location points are given in a 2-d distance matrix D. In the case of the pCD, we assume that the service cost between a client and the location of a facility, which we try to minimize, is as natural given by the length of the shortest path between the two nodes in the network. We assume that the shortest paths between all pairs of nodes have been precomputed and their lengths are stored in a 2-d matrix SP. However, if necessary, instead of precomputing the distances and storing them in a distance matrix, they could be computed "on the fly", under the condition

that this operation takes constant time. This holds for Euclidean or Manhattan distances given the coordinates of the points, but it does not hold for the shortest path in a network. To summarize, in a pDD we have:

- P: the set of candidate facility locations.
- F: the set of facilities to be located.
- p: the number of facilities to be located.
- $D[i, j]$: the distance between any two points.
- d_{ij}: the lower bound in the distance between each pair of facilities (i, j).

Additionally, in a pCD we have:

- CL: the set of demand nodes.
- $SP[i, j]$: the shortest path (s.p.) distance between any two points i and j.
- dk_i: the lower bound in the distance between each facility $i \in F$ and all demand nodes.

The goal in a pCD (resp. pDD) is to locate each facility so that the maximum s.p. distance between any client and its closest facility is minimized (resp. the minimum distance between any two facilities is maximized), subject to the satisfaction of all the distance constraints.

4 ILP Model for the pCD

In this section, we present an ILP formulation for the pCD, based on a standard formulation for p-center [18]. The model is extended to deal with heterogeneous facilities and to include distance constraints. We make use of the following additional notation:

- $C = \{(i, j, f_1, f_2) | i, j \in P, f_1, f_2 \in F, D[i, j] \leq d_{f_1 f_2}\}$: 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.
- $N = \{(i, j) | i \in P, j \in F, \exists k \in CL, D[i, k] \leq dk_j\}$: 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.
- $y_{ki} = 1$ if a demand node $k \in CL$ is assigned to a facility site $i \in P$ and 0 otherwise.

As we deal with heterogeneous facilities, we need $|P| \times |F|$ variables, i.e. 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 $i \in P$, because facilities are not indistinguishable as in the standard case of p-center. In addition, we need $|CL| \times |P|$ variables, i.e. $y_{ki}, \forall (k, i), k \in CL, i \in P$, in order to know whether or not a demand node $k \in CL$ is assigned to a facility site $i \in P$. Variables $y_{ki}, \forall (k, i), k \in CL, i \in P$, 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. Variable z is a continuous variable capturing the maximum s.p. distance between clients and located facilities.

The mixed-integer linear programming model for the pCD problem can be expressed as:

$$\min \quad z \tag{1}$$

$$\text{s.t.} \quad \sum_{i \in P} x_{ij} = 1 \qquad \forall j \in F \tag{2}$$

$$\sum_{i \in P} \sum_{j \in F} x_{ij} = p \tag{3}$$

$$\sum_{i \in P} y_{ki} = 1 \qquad \forall k \in CL \tag{4}$$

$$y_{ki} \leq \sum_{j \in F} x_{ij} \qquad \forall k \in CL, \forall i \in P \tag{5}$$

$$\sum_{i \in P} y_{ki} \times SP[k,i] \leq z \qquad \forall k \in CL \tag{6}$$

$$\sum_{j \in F} x_{ij} \leq 1 \qquad \forall i \in P \tag{7}$$

$$x_{if_1} + x_{jf_2} \leq 1 \qquad \forall (i,j,f_1,f_2) \in C \tag{8}$$

$$x_{ij} = 0 \qquad \forall (i,j) \in N \tag{9}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in P, \forall j \in F \tag{10}$$

$$y_{ki} \in \{0,1\} \qquad \forall k \in CL, \forall i \in P \tag{11}$$

$$z \in \mathbb{R} \tag{12}$$

The objective function 1 aims at minimizing the maximum s.p. distance between the clients and their nearest located facility. Constraint 2 guarantees that each facility should be hosted at exactly one facility site, while Constraint 3 specifies that p facilities are to be located. Although this constraint is subsumed by Constraint 2, our experiments showed that there are cases where it results in important speed-ups. Hence, we include it in the model. Constraint 4 ensures that each demand node will be served by one facility site, while Constraint 5 guarantees that each demand node will be served by a facility site where a facility has been located. It is a generalization of the Balinski constraint [2] to the case of heterogeneous facilities. Constraint 6 ensures that the variable z will be greater than or equal to all the s.p. distances between demand nodes and their nearest located facility.

Constraint 7 ensures that each facility site can host at most one facility. Constraint 8 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. Finally, Constraint 9 is the distance constraint between facilities and demand nodes. This constraint ensures that any located facility is at a safe distance from every demand node.

Two important decisions regarding the efficiency of the model are how to formulate the requirement that each client will be served by an open facility site, and the modeling of the distance constraints. Regarding the former, the requirement can be captured by a generalization of the Efroymson & Ray constraint, using a Big M constant. This results in fewer constraints compared to using the Balinski Constraint 5, but run times are clearly worse, and therefore the Efroymson & Ray was rejected. Regarding the distance constraints, we chose to follow and adapt the best model described in [4], where essentially the distance constraints are modeled through a special case of clique constraints. Computational results from [4], as well as our own preliminary experiments, showed that this model is better than various other models proposed in the literature [38, 40].

5 CP Models for the pCD

The pCD is modeled as a Constraint Optimization Problem (X, Dom, CL, C, O), where X is the set of decision variables, Dom is the set of finite domains, CL is the set of demand nodes, 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 $|X| = |F| = p$. The domain of each variable $x_i \in X$, denoted by $Dom(x_i)$, includes as values all the points where a facility can be located, i.e. $\forall x_i \in X : Dom(x_i) = P$.
2. Y_1 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 $y_{1ij} \in Y_1$ and a constraint $y_{1ij} = D[x_i, x_j]$. Hence, each $y_{1ij} \in Y_1$ models the distance between x_i and x_j. In CP solvers, this is implemented using the Element global constraint, i.e. $y_{1ij} = Element(D, [x_i, x_j])$.
3. Y_2 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 $y_{2ik} \in Y_2$ and a constraint $y_{2ik} = D[x_i, c_k]$. Hence, each $y_{2ik} \in Y_2$ models the distance between x_i and c_k. In CP solvers, this is again implemented using the Element constraint, i.e. $y_{2ik} = Element(D, [x_i, c_k])$.
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. s.p. distance) between x_i and c_k. Again, this is implemented using the Element constraint, i.e. $s_{ik} = Element(SP, [x_i, c_k])$.
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 $z_k = min(s_{1k}, s_{2k}, ..., s_{pk})$. 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 $y_{1ij} > d_{ij}$.
7. For each variable $y_{2ik} \in Y_2$, there is a distance constraint $y_{2ik} > dk_i$.
8. There is a variable z, s.t. $z = \max(Z)$, capturing the maximum shortest path distance between a client and its closest facility.
9. The objective function is $O = minimize(z)$.

The $s_{ik} = SP[x_i, c_k]$ constraints link the auxiliary variables s_{ik}, and therefore also the z variable and the objective function, with the decision variables. We also considered adding an AllDifferent constraint over all variables in X. Such a constraint is redundant, as the distance constraints already force the variables to take different values, and experiments with and without it showed no noticeable difference. Also, to reduce the number of auxiliary variables, the distance constraints between facilities can be captured using the Table constraint instead of the Element constraint. Our experiments showed that this does not make a difference in very large problems, where, as we will demonstrate, OR-Tools runs out of memory, but its effect on smaller problems remains to be experimentally investigated in detail.

A reason for the failure of OR-Tools to solve large instances of the pCD (and the pDD) is the size of the model it constructs, largely because of the very large domains (which are not uncommon in location problems), along with the auxiliary variables. This is to an extent due to the lazy clause generation mechanism of the solver, which creates a large amount of literals to represent all the variable-value combinations. To bypass this, we propose to use a much simpler model, dropping all the auxiliary variables and relevant constraints, resulting in a model with only the p decision variables and the distance constraints. 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 incumbent solution. If so, then the bound is tightened.

Dropping the auxiliary variables may result in a lighter model, but on the other hand we lose propagation power. If there is no link between the decision variables and the objective function, whether it is explicitly represented, as in the CP model above (items 8,9), or procedurally handled, then any improvement in the cost of the incumbent solution will not be propagated to the decision variables. To partially overcome this problem while keeping a simple model, we apply the following inference techniques:

- Assume that a solution A with better cost than all previously found ones has been located and let C_A denote its cost. We find the clients that determine the value of C_A (i.e. clients with distance C_A from their closest facility). For each one of them, we check if there exists a candidate facility point at a smaller distance than C_A. If there is no such point for some client, then there is no possible way for a facility to be located at a facility point that is closer to this specific client than the value of C_A. Hence, the value of the objective function cannot be further improved, and therefore we terminate search.
- After a variable has been assigned and the distance constraints have been propagated, we check for each client if there exists a value (i.e. available facility point) in any domain of a variable that is located at a distance smaller than the value of C_A, from this specific client. If no such value exists in any domain, then we know that in any feasible solution existing in the sub-tree below the current assignment, at least one client will be assigned to a facility located at an equal or greater distance than C_A. Hence, the cost cannot be improved, and therefore we prune the current branch and continue searching.

Both these techniques are subsumed by propagation in a CP solver that uses the full model of the problem described above. But in the simple model they (partially) compensate for the absence of a link between the optimization function and the decision variables.

6 Enhancing a Heuristic CP-Based Method Through LS

Ploskas et al. proposed a heuristic technique for the pDD 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. Thereafter, at each node, after the currently tried assignment $x_i = a$ has been propagated, an upper bound for the best possible solution is computed, giving an estimation of the best possible cost that can be achieved if the sub-tree 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.

At each node, the bound is computed by applying the greedy heuristic for p-dispersion on the relaxed problem obtained by not considering the distance constraints. Assuming that x_i is the current variable, $x_1 \leftarrow v_1, \ldots, x_{i-1} \leftarrow v_{i-1}$ is the assignment to past variables and v_i is the value under consideration for x_i, the heuristic greedily computes the cost of the "best" assignment for the future variables $x_{i+1}, \ldots, x_p$ [41]. That is, it visits these variables one by one, starting with x_{i+1}, and for each variable x_j, $i+1 \leq j \leq p$, and each value $v_j \in Dom(x_j)$, it finds the minimum distance between v_j and any assignment (location) among variables (facilities) $x_1, \ldots, x_{j-1}$. The value that maximizes this distance is then (temporarily) assigned to x_j. This is repeated until all variables have been assigned.

This heuristic pruning method was embedded in a custom CP solver (thus making the solver incomplete), which was compared to the complete ILP and CP-SAT solvers Gurobi and OR-Tools. Results demonstrated that it can discover much better solutions in large instances that are very hard for the complete solvers. However, the custom solver was able

to discover the optimal solution in only 2 out of the 82 MDPLIB instances of smaller size that are solvable to optimality by the complete solvers. Also, in many cases the cost of the best solution it discovered was quite far from the optimal.

The reason for this is that the greedy heuristic often underestimates the true cost that can be obtained. This leads to faster exploration of the search space, as many branches are pruned, but it also very often results in the omission of optimal/near-optimal solutions. We now describe one way to improve the effectiveness of a solver that follows this reasoning using local search to obtain more accurate bounds when estimating the cost at each node.

6.1 pDD

We propose to invoke LS before the decision is taken to cut off the branch, when the greedy heuristic dictates to do so. We use the assignment obtained by the greedy heuristic as the initial solution to the relaxed problem that omits the distance constraints, and then try to improve it with local moves until either it cannot be improved any more (a local maximum) or its cost becomes higher than that of the incumbent solution. In the former case we cut off the current branch, while in the latter we accept it and continue its exploration.

We use a variant of the best pairwise interchange heuristic for p-dispersion. Our method takes as input the complete assignment $A =< x_1 \leftarrow v_1, \ldots, x_i \leftarrow v_i, x_{i+1} \leftarrow v_{i+1}, \ldots, x_p \leftarrow v_p >$, where $x_{i+1} \leftarrow v_{i+1}, \ldots, x_p \leftarrow v_p$ is the temporary assignment to future variables, the cost C_A of this assignment and the cost of the incumbent solution C_I. It then finds the pairs of facilities (x_l, x_k) that are closest to each other, and therefore have distance equal to C_A (there can be many such pairs, which we call "culprit"), it forms the set of variables X_c involved in such pairs, and for each variable in X_c, it tries to find an alternative location so as to improve the value of C_A as much as possible. Note that for any "culprit" pair, both variables will necessarily be among $x_{i+1}, \ldots, x_p$ (the future ones), and therefore we can freely change their values in the assignment. This stems from the propagation of the updated bound any time a solution that improves the current bound is discovered. CP solvers do this by adding a constraint forcing subsequent solutions to be better than the current one, while the method of [41], that we follow, does it in a slightly different way by integrating the bound's propagation in the solver's arc consistency propagation mechanism.

If at some point, C_A becomes higher than C_I then the method terminates, signaling that the current branch must not be cut off. If this does not happen then the method stops when a local maximum is reached, e.g. when there is no alternative location for any variable in X_c that can increase the value of C_A. Algorithm 1 depicts this process, which, as our experiments showed, is more effective than the best pairwise interchange heuristic which it modifies, especially when a variable is involved in more than one "culprit" pair.

$A[x_j]$ denotes the value that variable x_j takes in the assignment A. Function *Compute_Cost* takes a complete assignment of the variables and returns its cost. The identification of the pairs of facilities that are closest to each other takes place each time a local move is made because any improvement to the cost means that other pairs are now the "culprit" ones.

Naturally, more sophisticated LS methods or meta-heuristics might be able to obtain better bounds. Indeed, we tried several options, including a GRASP meta-heuristic. The LS method of Algorithm 1 was chosen because it achieves a good balance in terms of quality and cpu time. For instance, the GRASP procedure rarely offered overall quality improvements to compensate for the higher run times (and implementation complexity).

Finally, we also use LS to tighten the bound obtained each time a solution that improves the incumbent is discovered. Specifically, once a new incumbent solution $A =< x_1 \leftarrow v_1, \ldots, x_p \leftarrow v_p >$ with cost C_I is discovered, we evaluate the alternative assignments of all

Algorithm 1 $LS(X, Dom, C, A, C_A, C_I)$ for the pDD.

```
repeat until no change in value of C_A
  X_c ← {x_l, x_k ∈ X | D[A[x_l], A[x_k]] = C_A};
  C_temp ← C_A;
  for each x_j ∈ X_c
    for each v_j ∈ Dom(x_j) | v_j ≠ A[x_j]
      A[x_j] ← v_j;
      temp-cost ← Compute_Cost(X, Dom, C, A);
      if temp-cost > C_temp
        C_temp ← temp-cost;
        mark A[x_j] = v_j as best move;
      restore A[x_j];
  if C_temp > C_A
    C_A ← C_temp;
    if C_A > C_I return true;
    make best move;
return false;
```

variables involved in "culprit" pairs. If an assignment that improves C_I (say $x_i \leftarrow v_k$) is found, and by changing the value of x_i to v_k in A, we get a feasible solution (i.e. the distance constraints are satisfied) then C_I is tightened and the process is repeated until no local move that results in a feasible solution can improve C_I. Thereafter, the solver continues search as usual, but with a (hopefully) tightened bound. Using LS to improve the solutions found by a CP solver in optimization problems is a standard way of integrating CP and LS [24].

6.2 pCD

The heuristic we have developed for the pCD works in a similar way. The cost of the first feasible solution found is now used as the initial upper bound, as we have a minimization problem. The initial estimation at each node is again performed by solving the relaxed problem without the distance constraints in a greedy fashion. That is, for each variable x_j, $i+1 \leq j \leq p$, and each value $v_j \in Dom(x_j)$, we compute the maximum distance between any client and its closest facility among $x_1, \ldots, x_j$ that we would get if v_j was assigned to x_j. The value that minimizes this distance is then temporarily assigned to x_j. After all future variables have been assigned in this way, the resulting assignment A is given to a local search method that tries to improve it. The only difference between this method and the one of Algorithm 1, is that in the case of the pCD there are no "culprit" pairs of variables. Hence, the second line in Algorithm 1 is omitted and all variables are considered when looking for the best local move. Let us explain this.

In a pCD the cost C_A of any complete assignment A is due to the one or more clients that are at distance C_A from their closest facility. But any relocation of a facility x_i may potentially result in an improvement of the cost (a decrease in the value of C_A), as x_i may move closer to these clients, meaning that they could perhaps now be serviced by x_i. In contrast, in the pDD, if none of the variables in a culprit pair is relocated, the cost will never improve (but could worsen), regardless of the relocation of other facilities.

7 Experiments

We experimented with instances generated in two different ways. The first uses a benchmark library as basis to create pCDs or pDDs. For the former, we use the p-median benchmark dataset [3], while for the latter we use the p-dispersion benchmark library MDPLIB 2.0 [37], as in [41]. In the second generation method we seek to locate a number of facilities in a grid.

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 seconds for all the experiments reported below. The ILP model was solved using Gurobi 9.0.3 [28]. The CP model was written in the CPMpy modeling tool [27] and compiled into CP-SAT OR-Tools [15]. The heuristic CP approach, for both pDDs and pCDs, was implemented in a custom solver written in C. This solver, which is basically a MAC search algorithm [42], implements the simple CP model of Section 5, uses dom/wdeg for variable ordering [5], lexicographic value ordering, arc consistency for the propagation of distance constraints and implements the two inference techniques of Section 5. We also ran experiments with an 8-thread version of OR-Tools, which uses techniques such as large neighborhood search in parallel with the main search, as well as the CP solver Choco [14]. The ILP model is stored in compressed sparse column format, as the constraint matrix can sometimes be too large to be stored as a full array.

7.1 Problem Generation Models

The MDPLIB collects a large number of p-dispersion instances divided into various classes [37]. In the GKD, MDG, and SOM classes, the distances between the potential facility locations are given by Euclidean distances, random real numbers, and random integers, respectively. The instances we tried have 100-1000 potential facility points and 10-30 facilities.

We generated pCDs in two different ways. The *grid generation model* creates problems embedded in a $n \times n$ grid. It takes the following parameters: n, p, $|CL|$, $|P|$. We first randomly select $|CL| + |P|$ among the $n \times n$ nodes. $|CL|$ 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 distance of the shortest path between any two points can be at best equal to the Manhattan distance between them. For each distance constraint $dis(x_i, x_j) > d_{ij}$ between facilities x_i and x_j, d_{ij} is randomly set to an integer number in the interval $[0, max\_euc/2]$, where 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.

The *p-median based generator* takes instances from the p-median benchmark dataset [3], consisting of problems with 100–900 nodes and 5 to 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)** 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 d_{ij}, we find the minimum and maximum distance between all pairs of candidate sites and we set d_{ij} equal to a random number in the range $[\min, \min + (\max - \min)/10]$. Similarly for parameter dk_i.

For each generation model and each setting of the parameters, 10 instances were generated.

7.2 Experiments with the pDD

In Table 1 we compare the LS-enhanced CP-based method, denoted CP_{LS}, to that of [41], denoted CP_G, using the same instances (https://github.com/ploskasnikos/pdispersion), and adding some larger ones that are very hard for the complete solvers. For each class we give the number of the MDPLIB instance on which it is based, and in brackets the numbers of potential location sites and facilities. For each solver configuration we report the total cpu time taken over the 10 instances ($\sum$cpu columns), the number of times when the optimal solution was found (#opt columns), and the mean value of the best solution found within the time limit. In the #opt column for CP_G we give in brackets the number of instances for which the optimal is known. We also give the number of instances in each class where CP_{LS} found a better solution compared to CP_G (#imp columns). In brackets we give the number of instances where CP_{LS} found a worse solution. As baseline, in the last column, for each class we give the mean value of the best solution found by the complete solver that displayed the best performance in the particular class. This is left blank (-) if no complete solver was able to find at least one solution in all instances of a class.

Table 1 Comparing CP_{LS} to CP_G on MDPLIB-generated pDDs.

Class (p,$\lvert P\rvert$)	CP_G $\sum$cpu	#opt	cost	CP_{LS} $\sum$cpu	#opt	#imp	cost	Baseline cost
MDG								
a1 (100,10)	1	0 (10)	4.35	1	10	10 (0)	4.68	4.68
a1 (100,20)	94	0 (0)	1.69	170	0	8 (2)	1.79	1.17
a1 (500,10)	51	0 (0)	5.88	56	0	10 (0)	6.11	6.02
a1 (500,20)	132	0 (0)	2.91	188	0	9 (1)	2.96	1.57
a2 (100,10)	1	0 (10)	4.24	1	10	10 (0)	4.74	4.74
a2 (100,20)	156	0 (0)	1.64	166	0	7 (2)	1.71	1.4
a2 (500,10)	54	0(0)	5.94	66	0	10 (0)	6.22	5.92
b1 (100,10)	1	0 (10)	428.18	1	0	10 (0)	455.94	460.11
b1 (100,20)	63	0 (0)	181.2	133	0	8 (1)	200.49	109.35
b1 (500,10)	45	0 (0)	576.45	53	0	10 (0)	591.56	584.33
b1 (500,20)	107	0 (0)	290.64	128	0	9 (0)	302.75	-
b2 (100,10)	1	0 (9)	428.17	1	9	10 (0)	459.99	459.99
b2 (100,20)	109	0 (0)	159.93	140	0	8 (2)	170.40	113.33
b2 (500,10)	51	0 (0)	574.55	55	0	10 (0)	608.61	581.31
GKD								
d1 (100,10)	2	1 (10)	33.42	2	0	9 (1)	33.85	34.06
d1 (250,10)	11	0 (7)	34.25	21	3	10 (0)	36.06	35.98
d1 (250,20)	>19,837	0 (0)	18.83	>24,663	0	8 (1)	19.24	10.55
d1 (500,10)	108	0 (0)	36.01	99	0	10 (0)	38.21	36.96
d1 (1000, 20)	>36,000	0 (0)	16.99	>36,000	0	6 (1)	17.95	-
d2 (100,10)	2	1 (10)	31.29	3	8	10 (0)	34.22	34.82
d2 (250,10)	10	0 (5)	35.06	22	0	10 (0)	37.1	36.31
d2 (250,20)	>24,865	0 (0)	14.93	>24,998	0	5 (0)	15.39	-
d2 (1000, 10)	846	0 (0)	36.47	1,830	0	10 (0)	39.25	32.72
SOM-GRID								
b5 (200, 20)	11	0 (0)	2	13	0	0 (0)	2	2
g7 (80, 30)	36	10 (10)	2	51	10	0 (0)	2	1.9

We can make the following observations. CP_{LS} finds better solutions than CP_G in the vast majority of instances, discovering the optimal solution in 40 MDPLIB instances, compared to only 2 for CP_G. This gives an improvement in the mean cost in all classes of the MDG and

GKD categories, often by large margins. As a result, CP_{LS} finds optimal solutions or near-optimal ones in all of the smaller classes that are within reach of the complete solvers, while improving the bounds in larger classes for which the optimal is unknown (e.g. in b1(100,20) where the value is now almost twice that obtained by the complete solvers). Importantly, the cpu time overheads of CP_{LS} compared to CP_G are not very significant, keeping in mind that the $\sum$cpu columns give total run times over 10 instances until termination or cut-off and that both solver configurations are orders of magnitude faster than the complete solvers [41]. CP_{LS} usually reaches the best solution found by CP_G in similar or faster run times, but often takes longer to terminate because it further improves this solution.

For CP_{LS}, we counted the number of times that the greedy heuristic made an estimation lower than the current bound, meaning that LS was then called, and the number of times that LS managed to increase the estimation to a value greater than the current bound, meaning that the current branch was not cut off. It turns out that in around 4% to 14% of the calls to LS, on average, the current branch was not cut off (details in the Appendix). Despite these relatively low percentages, CP_{LS} obtains significant improvements in solution quality.

Results from the SOM MDPLIB class and from grid-structured problems are also given in [41]. These categories were both very easy in terms of run times for CP_G and CP_{LS}. In these cases, CP_{LS} found the same solutions as CP_G because either CP_G already discovered the (known) optimal solutions in the smaller classes, or the solutions discovered had very little room for improvement (objective values in the range $1 \dots 5$). Hence, we only give indicative results from the hardest class from each category. In the b5 SOM class, CP_{LS} and CP_G display similar performance and probably both discover the optimal in all instances (Gurobi also finds solutions with cost 2, but was unable to prove optimality within the time limit). In the g7 grid class, both CP_{LS} and CP_G discover the known optimal solutions in all instances.

Finally, in some rare cases CP_G can locate slightly better solutions than CP_{LS}. To put it simply, this is because both methods are heuristic. In more detail, a possible scenario is the following: Suppose that the first solution A located has objective value C_A. As the search continues, assume that LS improves the heuristic's estimation at some node and does not cut the current branch (as CP_G would do), leading to a better solution B being later located, with value $\text{C}_B < \text{C}_A$. The discovery of solution B may lead to more branches being then cut off, potentially including a branch that leads to a solution C with value $\text{C}_C < \text{C}_B$. Hence, CP_{LS} will not discover this solution. On the other hand, it is possible that CP_G locates solution C, because using the upper bound of $\text{C}_A > \text{C}_B$ to prune, may lead to weaker pruning, allowing for the branch that leads to solution C to be explored.

7.3 Experiments with the pCD

Table 2 details the classes generated for pCDs using the two 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 $< |V|, p, |P|, |CL| >$. For example, class <400,5,80,320> includes problems with 400 points, 5 facilities, 80 potential locations and 320 clients. The 2nd column gives classes where the number of clients is larger or equal to the number of candidate sites, while the 3rd gives classes where there are more candidate sites than clients. Each grid based class is defined by the parameters $< n, p, |P|, |CL| >$. For example, in class <10,20,80,20> we have a 10×10 grid, 20 facilities, 80 potential locations and 20 clients.

We have experimented with five solvers: Gurobi, OR-Tools, 8-threads OR-Tools, Choco, our custom CP solver. These are all complete solvers. We also experimented with three configurations of our solver that incorporate the pruning heuristic, making the solver incom-

Table 2 Problem classes and their characteristics.

p-median based	$\|CL\| \geq \|FP\|$	$\|FP\| > \|CL\|$	**grid based**	
pmed5	<100,33,40,60>	<100,33,80,20>	g1	<10,20,80,20>
pmed10	<200,67,100,100>	<200,67,160,40>	g2	<20,10,350,50>
pmed15	<300,100,150,150>	<300,100,240,60>	g3	<20,20,350,50>
pmed21	<500,5,100,400>	<500,5,400,100>	g4	<20,25,350,50>
pmed26	<600,5,120,480>	<600,5,480,120>	g5	<30,20,300,50>
pmed31	<700,5,140,560>	<700,5,560,140>	g6	<30,10,500,100>
pmed36	<800,10,400,400>	<800,10,640,160>	g7	<30,20,500,100>
pmed38	<900,5,450,450>	<900,5,720,180>	g8	<30,20,700,200>
			g9	<50,100,1300,200>

plete. The first (CP_G), only uses a greedy heuristic to estimate the cost at each node (as in [41] for the pDD), the second one uses LS to perform the estimation, and the third one (CP_{LS}), adds the LS component that tries to improve any solution found. Choco displayed inferior performance compared to OR-Tools and therefore was not included in extensive experiments, while the 8-thread version of OR-Tools did not demonstrate any significant benefits compared to the single thread one. The basic complete version of our solver (i.e. without the heuristic) was not competitive, despite using a simple model of the problem. Indicative results of this solver and 8-threads OR-Tools are given in the Appendix.

Among the configurations of our solver that use the bound estimation heuristic, CP_{LS} displayed the best results. CP_{LS} finds the optimal in 85 out of the 187 instances for which the optimal is known, as opposed to 68 for CP_G, and improves the cost found by CP_G (resp. worsens it) in 47 (resp. 4). Regarding run times, as in the pDD, CP_{LS} usually reaches the best solution found by CP_G in similar or faster run times, but may take longer to terminate because it further improves this solution. It achieves similar run times in classes where CP_{LS} rarely improves the cost, while it can take up to twice the time to terminate in classes where it often improves the cost. The second LS component makes a slight contribution towards the solver's performance. When turned off (i.e. LS only used for branch pruning) then there is an improvement in 43 instances (82 optimal) compared to CP_G.

Interestingly, in only around 0.3% to 3% of the calls to LS, the current branch was not cut off. Despite these very low percentages, CP_{LS} still managed to find better solutions than CP_G in many instances. It is not surprising that the corresponding percentages are higher in pDDs, as intuitively it is more likely to improve the bound's estimation by changing the location of a facility through a local move in a pDD rather than in a pCD, where a local move must place a facility f_i close to all the clients that are at maximum distance from their closest facility, so that they are now served by f_i.

In Table 3, we compare CP_{LS} to Gurobi and OR-Tools. We report the total cpu times, the mean cost of the best solution found, and the number of instances where any solver was cut off (in brackets after $\sum$cpu). If a solver did not terminate, we count $3,600$ secs towards its cpu time sum and record the best solution it was able to find. Column t_b gives the mean cpu time taken by CP_{LS} to find its 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_{LS}. If CP_{LS} was unable to find any solution within the time limit in some instances then t_b is left blank (-). Accordingly, if Gurobi or OR-Tools did not manage to match the solution found by CP_{LS} in some instances, t_m is left blank. The last column gives the number of instances where CP_{LS} found the optimal solution. In some classes, OR-Tools (and Gurobi in one class) suffered memory exhaustion and crashed in all instances without discovering any solutions. This is denoted with MEM in the $\sum$cpu column.

Table 3 Comparing solvers on pCD problems.

	Gur $\sum$cpu	t_m	cost	ORt $\sum$cpu	t_m	cost	CP$_{LS}$ $\sum$cpu	t_b	cost	#opt
$\|CL\| \geq \|FP\|$										
pmed5	2 (0)	0	100.5	1,668 (0)	164	100.5	>7,306 (2)	-	-	5
pmed10	44 (0)	4	63.4	1,963 (0)	196	63.4	4 (0)	0.2	63.4	10
pmed15	123 (0)	12	49.7	19,753 (0)	1,974	49.7	384 (0)	1.3	50.1	9
pmed21	201 (0)	11	45.8	212 (0)	18	45.8	6 (0)	0.3	46.5	6
pmed26	206 (0)	13	43.2	224 (0)	16	43.2	9 (0)	0.3	44	4
pmed31	985 (0)	67	34.7	254 (0)	19	34.7	17 (0)	0.7	35.6	3
pmed36	1,160 (0)	19	35.9	>18,468 (3)	951	35.9	478 (0)	19	37	4
pmed38	146 (0)	14	37.2	3,522 (0)	352	37.2	52 (0)	3	37.3	9
$\|FP\| > \|CL\|$										
pmed5	5 (0)	0	47	249(0)	25	47	407 (0)	40	47	10
pmed10	64 (0)	6	25.3	10,775 (0)	1,076	25.3	343 (0)	2	26.3	7
pmed15	219 (0)	21	21.6	MEM	-	-	7,544 (0)	36	22.3	7
pmed21	252 (0)	9	27.6	>19,747 (1)	225	27.6	101 (0)	5	28.6	2
pmed26	537 (0)	27	24.7	14,901 (0)	460	24.7	113 (0)	5	25.6	2
pmed31	2,535 (0)	48	21.8	>33,041 (5)	912	21.8	227 (0)	8	22.8	0
pmed36	>5,817 (1)	110	19	MEM	-	-	1,697 (0)	110	20.7	0
pmed38	4,345 (0)	69	20.2	>33,218 (6)	2,454	20.2	184 (0)	8	20.9	3
GRID										
g1	4,103 (0)	381	2.1	443 (0)	14	2.1	2 (0)	0.1	2.8	3
g2	5,191 (0)	73	4.1	7,737 (0)	16	4.1	26 (0)	0.1	6.1	0
g3	>36,000 (10)	-	-	>36,000 (10)	159	4.2	205 (0)	14	5.3	0
g4	>36,000 (10)	-	-	>33,302 (9)	-	-	9,300 (0)	922	5.4	1
g5	>36,000 (10)	-	-	>36,000 (10)	818	6.9	3,317 (0)	320	8.4	0
g6	>23,021 (4)	852	7.6	>34,641 (9)	30	7.4	99 (0)	0.5	9.9	0
g7	>36,000 (10)	-	-	>36,000 (10)	-	-	1,264 (0)	100	8.4	0
g8	>36,000 (10)	-	-	>36,000 (10)	-	-	275 (0)	10	9	0
g9	MEM	-	-	MEM	-	-	>36,000 (10)	814	13.7	0

Gurobi outperforms OR-Tools, in run times, in all classes of the p-median based problems, and in fact finds these problems quite easy, terminating within the time limit in all but one instance. CP$_{LS}$ is able to find many optimal or near-optimal solutions, and it does this very fast in most classes. As the t_b and t_m columns indicate, it can take OR-Tools orders of magnitude longer runs to match the solutions found by CP$_{LS}$ (e.g. pmed15 and pmed38). Compared to Gurobi, CP$_{LS}$ is generally able to find good solutions faster, especially in classes with few facilities (e.g. pmed31), but cannot compete in classes that include many facilities and relatively few facility points. In terms of optimal solutions found, CP$_{LS}$ is quite successful in classes with more clients than facility points, as it finds at least 5 optimal solutions in 5 classes, while it locates all of them in the pmed10 class. The performance is not as good in the $|FP| > |CL|$ category, as there are classes where no optimal solution is found. This is not surprising, considering that in such classes, domain sizes are larger.

Regarding grid based problems, the results differ significantly. OR-Tools now performs better than Gurobi, as the latter finds 6 out of the 9 classes very hard and did not manage to find any solution in any instance of these classes. OR-Tools also finds some of these classes very hard, but managed to discover solutions in at least one instance from every class, but not in all instances (hence the blank t_m and cost columns). CP$_{LS}$ is much more robust, being able to find solutions in all instances of all classes, and doing this very fast in some classes (including hard ones, such as g8).

Regarding solution quality, in classes where one or both of the complete solvers solved all instances, the cost of the solutions discovered by CP_{LS} is worse than that discovered by OR-Tools (and in some cases Gurobi). Also, CP_{LS} managed to find only 4 of the known optimal solutions. On the other hand, CP_{LS} obtained solutions in all instances, even the very large ones with 1,300 facility points. This ability to handle the very large instances is due to both the simpler model of the problem and the pruning heuristic. The simpler model means that propagation is not very costly, and therefore an initial solution is sooner or later located. Thereafter, the heuristic takes over and prunes many branches, allowing for the solution to be quickly improved.

On the other hand, the propagation performed by OR-Tools is costly, because of the many auxiliary variables and relevant constraints, meaning that for very large grid problems, which typically have few solutions, it is either unable to reach a solution within the time limit, even if the memory requirements are manageable, or takes very long to improve the ones found. Of course, OR-Tools could also use the simpler model, but in this case the pruning heuristic should be written into the solver as a specialized constraint to make it competitive.

Regarding Gurobi, a factor that seems to affects its performance is the ratio of candidate facility sites to facilities. If this is small then Gurobi quickly locates the optimal solution and proves optimality. In contrast, when it is large then Gurobi finds the problems harder. On the other hand, the performance of CP_{LS}, and OR-Tools to a large extent, is dependant on the number of facilities/variables which mainly determines the size of the search space these solvers explore.

Another important factor that affects the performance of the solvers is the number of solutions. Gurobi seems to benefit from the presence of many solutions in an instance, while this does not hold for CP_{LS} 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 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 minutes of cpu time on average for the 10 instances of pmed38 with $|CL| \geq |FP|$. The enumeration was stopped after 10 minutes, meaning that the actual number of solutions could be much higher. Further to this, focusing on two contrasting instances of class g1; one having only 58 feasible solutions and another with 87,950,294 solutions, Gurobi took 3,590 secs to match the best solution of CP_{LS} in the former, while it managed this in 17 secs in the latter.

When there are few solutions then 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, it starts with a good initial bound and is able to quickly improve it. The low number of solutions in grid problems is also a possible explanation for the lower quality of solutions discovered by CP_{LS} compared to OR-Tools in classes that OR-Tools can handle. In such cases, erroneous pruning by the heuristic may be costly, as the optimal and near-optimal solutions are few, whereas when there are many solutions, (as in the MDPLIB and p-median benchmarks), mistakes are not as costly, as there may be many branches that lead to good solutions.

8 Conclusion

We have enhanced a recent heuristic approach towards solving the p-dispersion problem with distance constraints in two directions. First, we showed how LS can significantly improve the effectiveness of the method by computing better bound estimations at each node, and

therefore performing more focused branch pruning, and to a lesser extent by tightening the bound each time a new solution is discovered. Second, we showed that the entire method is applicable to other location problems, using the p-center problem as an example. Results demonstrated that a solver that heuristically prunes the search space during search is more robust than standard ILP and CP solvers. It would be very interesting to investigate the applicability of the approach to other types of constraint optimization problems.

References

1 T. Argo and E. Sandstrom. Separation distances in NFPA codes and standards (tech. rep.). Fire Protection Research Foundation. 2014.

2 M.L. Balinski. Integer programming: methods, uses, computations. *Management Science*, 12(3):253–313, 1965.

3 J.E. Beasley. A note on solving large p-median problems. *European Journal of Operational Research*, 21(2):270–273, 1985.

4 O. Berman and R. Huang. The minimum weighted covering location problem with distance constraints. *Computers and Operations Research*, 35(12):356–372, 2008. doi:10.1016/j.cor.2006.03.003.

5 F. Boussemart, F. Heremy, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI'04*, pages 482–486, 2004.

6 H. Calik and B.C. Tansel. Double bound method for solving the p-center location problem. *Comput. Oper. Res.*, 40(12):2991–2999, 2013. doi:10.1016/j.cor.2013.07.011.

7 H. Cambazard, D. Mehta, B. O'Sullivan, and L. Quesada. A computational geometry-based local search algorithm for planar location problems. In *Proceedings of the 9th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2012)*, pages 97–112, 2012.

8 S. Chaudhry, T. McCormick, and I. D. Moon. Locating independent facilities with maximum weight: Greedy heuristics. *International Journal of Management Science*, 14(5):383–389, 1986.

9 R. L. Church and M. E. Meadows. Results of a new approach to solving the p-median problem with maximum distance constraints. *Geographical Analysis*, 9(4):364–378, 1977.

10 V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979. doi:10.1287/moor.4.3.233.

11 W. Comley. The location of ambivalent facilities: Use of a quadratic zero-one programming algorithm. *Applied Mathematical Modeling*, 19(1):26–29, 1995.

12 Z. Dai, K. Xu, and M. Ornik. Repulsion-based p-dispersion with distance constraints in non-convex polygons. *Annals of Operations Research*, 307:75–91, 2021. doi:10.1007/s10479-021-04281-z.

13 M.S. Daskin. *Network and discrete location: models, algorithms, and applications, 2nd edn.* Wiley, Hoboken, 2013.

14 Choco development team. An Open-Source java library for constraint programming. https://choco-solver.org/.

15 OR-Tools development team. OR-Tools, CP-SAT solver. https://developers.google.com/optimization/cp/cp_solver.

16 T. Drezner, Z. Drezner, and A. Schöbel. The weber obnoxious facility location model: A big arc small arc approach. *Computers and Operations Research*, 98:240–250, 2018. doi:10.1016/j.cor.2018.06.006.

17 Z. Drezner, P. Kalczynski, and S. Salhi. The planar multiple obnoxious facilities location problem: A Voronoi based heuristic. *Omega*, 87:105–116, 2019.

18 S. Elloumi, M. Labbé, and Y. Pochet. A new formulation and resolution method for the p-center problem. *INFORMS J. Comput.*, 16(1):83–94, 2004. doi:10.1287/ijoc.1030.0028.

19 E. Erkut. The discrete p-dispersion problem. *European Journal of Operational Research*, 46(1):48–60, 1990.

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

21 E. Erkut and S. Neuman. Comparison of four models for dispersing facilities. *Information Systems and Operational Research*, 29:68–86, 1991.

22 E. Erkut, Y. Ülküsal, and O. Yeniçerioglu. A comparison of p-dispersion heuristics. *Computers & Operations Research*, 21(10):1103–1113, 1994. doi:10.1016/0305-0548(94)90041-8.

23 M. M. Fazel-Zarandi and J. C. Beck. Solving a location-allocation problem with logic-based benders' decomposition. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP 2009)*, pages 344–351, 2009.

24 F. Focacci, F. Laburthe, and A. Lodi. Local search and constraint programming. In F.W. Glover and G.A. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 369–403. Kluwer / Springer, 2003. doi:10.1007/0-306-48056-5_13.

25 J. B. Ghosh. Computational aspects of the maximum diversity problem. *Operations Research Letters*, 19(4):175–181, 1996. doi:10.1016/0167-6377(96)00025-9.

26 C. Gomes and M. Sellmann. Streamlined constraint reasoning. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP 2004)*, pages 274–289, 2004.

27 T. Guns. Increasing modeling language convenience with a universal n-dimensional array, CPpy as python-embedded example. In *Proceedings of the 18th Workshop on Constraint Modelling and Reformulation*, 2019.

28 LLC Gurobi Optimization. Gurobi optimizer reference manual. , 2023. URL: https://www.gurobi.com.

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

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

31 N. Isoart and J.-C. Régin. A k-Opt Based Constraint for the TSP. In *Proceedings of the 27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, 2021.

32 B. M. Khumawala. An efficient algorithm for the p-median problem with maximum distance constraints. *Geographical Analysis*, 5(4):309–321, 1973.

33 J. Krarup, D. Pisinger, and F. Plastria. Discrete location problems with push-pull objectives. *Discrete Applied Mathematics*, 123(1-3):363–378, 2002. doi:10.1016/S0166-218X(01)00346-8.

34 M. J. Kuby. Programming models for facility dispersion: The p-dispersion and maxisum dispersion problems. *Mathematical and Computer Modelling*, 10(10):792, 1988.

35 A.A. Kuehn and M.J. Hamburger. A heuristic program for locating warehouses. *Management Science*, 9:643–666, 1963.

36 M.Z. Lagerkvist and M. Rattfeldt. Half-checking propagators. In *Proceedings of the 19th Workshop on Constraint Modelling and Reformulation*, 2020.

37 R. Marti, A. Martinez-Gavara, S. Perez-Pelo, and J. Sanchez-Oro. A review on discrete diversity and dispersion maximization from an OR perspective. *European Journal of Operational Research*, 299(3):795–813, 2022. doi:10.1016/j.ejor.2021.07.044.

38 I. D. Moon and S. Chaudhry. An analysis of network location problems with distance constraints. *Management Science*, 30(3):290–307, 1984.

39 I. D. Moon and L. Papayanopoulos. Minimax location of two facilities with minimum separation: Interactive graphical solutions. *Journal of the Operations Research Society*, 42:685–694, 1991.

40 A.T. Murray, R.L. Church, R.A. Gerrard, and W.S. Tsui. Impact models for siting undesirable facilities. *Papers in Regional Science*, 77(1):19–36, 1998.

41 N. Ploskas, K. Stergiou, and D.C. Tsouros. The p-dispersion problem with distance constraints. In Roland H. C. Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming, CP 2023, August 27-31, 2023, Toronto, Canada*, volume 280 of

LIPIcs, pages 30:1–30:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.CP.2023.30.

42 D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In A.G. Cohn, editor, *Proceedings of the Eleventh European Conference on Artificial Intelligence, Amsterdam, The Netherlands, August 8-12, 1994*, pages 125–129, 1994.

43 D. Sayah and S. Irnich. A new compact formulation for the discrete p-dispersion problem. *European Journal of Operational Research*, 256(1):62–67, 2017. doi:10.1016/j.ejor.2016.06.036.

44 F. Sayyady and Y Fathi. An integer programming approach for solving the p-dispersion problem. *European Journal of Operational Research*, 253(1):216–225, 2016. doi:10.1016/j.ejor.2016.02.026.

45 D. R. Shier. A min-max theorem for p-center problems on a tree. *Transportation Science*, 11:243–252, 1977.

46 S.Y.D. Sorkhabi, D.A. Romero, J.C. Beck, and C. Amon. Constrained multi-objective wind farm layout optimization: Novel constraint handling approach based on constraint programming. *Renewable Energy*, 126(C):341–353, 2018.

47 B.C. Tansel, R.L. Francis, T.J. Lowe, and M.L. Chen. Duality and distance constraints for the nonlinear p-center problem and covering problem on a tree network. *Operations Research*, 30(4):725–744, 1982. doi:10.1287/opre.30.4.725.

48 S.B. Welch and S. Salhi. The obnoxious p facility network location problem with facility interaction. *European Journal of Operations Research*, 102:302–319, 1997.

49 49 C.F.R. §175.701. Separation distance requirements for packages containing class 7 (radioactive) materials in passenger-carrying aircraft. Title 49 Code of Federal Regulations, Part 175. 2021.

50 29 C.F.R. §1910.157. Portable fire extinguishers. Title 29 Code of Federal Regulations, Part 157. 2021.

Appendix

In Table 4 we give mean results regarding the pruning of the heuristic and the effect of LS. CP_{LS_h} is the solver configuration that uses LS only within the branch pruning heuristic to perform a potential improvement in the estimation of the bound. We denote by R_a the ratio of branches accepted (not cut off) by LS (within the branch pruning heuristic) to the total number of times that it was called. We denote by R_b the ratio of the total number of branches pruned by the heuristic to the number of times the heuristic was called. We denote by R_i, the ratio of the number of times that LS further improved a feasible solution to the number of times that it was called to do so. The ratios are presented as percentages and are computed as means over all the instances of a problem category (MDG and GKD in pDDs - p-median and grid in pCDs).

The high percentages of R_b explain the efficiency of the solver that applies this type of pruning, as large amounts of the search tree are cut off (more than 90% of the calls to the heuristic, on average, result in branch pruning, in both categories of the pCD and one of the pDD). As expected, the basic CP_G variant usually displays a higher pruning ratio compared to CP_{LS}, but not always, because in some instances, especially p-median based pCDs, the latter quickly finds very good solutions, tightening the bound, and thereby resulting in heavy pruning. Regarding the effect of LS as a means to improve newly discovered solutions, in the GKD classes of pDDs and the p-median classes of pCDs there was an improving ratio R_i of around 60%, meaning that LS was able to improve an incumbent solution more than half of the times it was called to do so.

Table 4 Ratios for pDDs and pCDs.

	R_a	R_b	R_i	R_a	R_b	R_i
pDD	MDG			GKD		
CP_G	-	91.78%	-	-	85.18%	-
CP_{LS}	4.16%	91.22%	35.64%	14.15%	79.94%	58.89%
CP_{LS_h}	4.19%	90.82%	-	14.12%	79.54%	-
pCD	p-median			Grid		
CP_G	-	90.55%	-	-	95.35%	-
CP_{LS}	0.36%	91.78%	60.91%	2.76%	93.79%	31.34%
CP_{LS_h}	0.61%	90.24%	-	3.08%	92.96%	-

In Table 5 an indicative comparison between the basic solver fo pCDs (denoted CP_{de}), i.e. the solver with the branch pruning heuristic deactivated, and CP_{LS} is presented. For each class, we report the total cpu times taken by the solvers, the mean cost of the best solution found, and the number of instances where any solver reached the cut off limit of 1 hour without terminating (in brackets after the total cpu time). Column t_b gives the mean cpu time taken by the solvers to find their best solution. Column t_m gives the mean cpu time taken by CP_{de} to find the first solution that matches (or improves) the cost of the best solution found by CP_{LS}. If CP_{de} did not manage to match the solution found by CP_{LS} in some instances, t_m is left blank. Finally, we also report the total nodes visited by each solver.

Table 5 Comparing CP solver with/without the branch pruning heuristic in pCDs.

	CP_{de}					CP_{LS}			
Class	$\sum$cpu	t_b	t_m	cost	$\sum$nodes	$\sum$cpu	t_b	cost	$\sum$nodes
$\|CL\| \geq \|FP\|$									
pmed21	2,775 (0)	19	8	45.9	72,202,355	6 (0)	0.3	46.5	2,995
pmed36	>21,600 (6)	406	-	37.8	307,856,397	478 (0)	19	37	15,731
pmed38	>12,355 (3)	286	-	37.5	269,869,194	52 (0)	3	37.3	8,960
$\|FP\| > \|CL\|$									
pmed21	>36,000 (10)	1,292	-	29.4	3,479,429,138	101 (0)	5	28.6	26,045
pmed36	>36,000 (10)	643	-	22.7	1,365,670,278	1,697 (0)	110	20.7	74,489
pmed38	>36,000 (10)	430	-	21.1	2,996,757,611	184 (0)	8	20.9	24,921
GRID									
g3	>32,788 (9)	381	17	4.1	67,219,121	205 (0)	14	5.3	154,500
g6	>36,000 (10)	751	88	9.2	250,773,552	99 (0)	0.5	9.9	11,230
g8	>31,509 (8)	918	20	7.6	23,653,359	275 (0)	10	9	55,194

Not surprisingly, the behaviour of CP_{de} resembles somewhat that of OR-Tools. It is by far slower than its incomplete version CP_{LS}, and typically discovers worse solutions, in p-median based problems. On the other hand, it is still slower in grid problems (though by smaller margins), but it discovers better solutions than CP_{LS}. The huge numbers of visited nodes compared to CP_{LS}, demonstrate the pruning power of the branch pruning heuristic, although results from the grid classes indicate that this massive pruning can sometimes be costly, in terms of solution quality, when problems have a relatively small amount of solutions.

In Table 6 we give an indicative comparison between two different configurations of OR-Tools, one single threaded and one with 8 threads. For each class, we report the total cpu times taken, the mean cost of the best solution found, and the number of instances where the cut off limit of 1 hour without terminating is reached (in brackets after the total cpu time).

Table 6 Comparing ORtools with 1 and 8 threads.

	ORt_1 $\sum$cpu	t_b	cost	ORt_8 $\sum$cpu	t_b	cost
$\|CL\| \geq \|FP\|$						
pmed5	1,668 (0)	167	100.5	872 (0)	86	100.5
pmed10	1,963 (0)	196	63.4	1,902 (0)	188	63.4
pmed21	212 (0)	18	45.8	203 (0)	16	45.8
pmed26	224 (0)	19	43.2	195 (0)	18	43.2
$\|FP\| > \|CL\|$						
pmed5	249 (0)	25	47	249 (0)	24	47
pmed10	10,775 (0)	1,077	25.3	10,726 (0)	1,071	25.3
pmed21	>19,747 (1)	497	27.6	10,173 (0)	319	27.6
pmed26	14,901 (0)	604	24.7	9,775 (0)	525	24.7
GRID						
g1	443 (0)	30	2.1	272 (0)	14	2.1
g2	7,737 (0)	98	4.1	8,178 (0)	158	4.1
g3	>36,000 (10)	396	4.2	>35,977 (9)	354	4.1
g6	>34,641 (9)	358	7.4	>35,334 (9)	732	7.4

Results demonstrate that the 8 thread configuration for OR-Tools did not result in significantly improved run times (sometimes it is detrimental). In most classes, the mean times taken for the discovery of the best solution were close. But importantly, the 8 thread variant very rarely manages to improve the cost of the best solution found. Finally, the two versions crashed in the same classes that are denoted with MEM in Table 3.