


The p-Dispersion Problem with Distance Constraints

Nikolaos Ploskas ✉ 

University of Western Macedonia, Kozani, Greece

Kostas Stergiou ✉ 

University of Western Macedonia, Kozani, Greece

Dimosthenis C. Tsouros ✉ 

KU Leuven, Belgium

Abstract

In the (maxmin) p-dispersion problem we seek to locate a set of facilities in an area so that the minimum distance between any pair of facilities is maximized. We study a variant of this problem where there exist constraints specifying the minimum allowed distances between the facilities. This type of problem, which we call PDDP, has not received much attention within the literature on location and dispersion problems, despite its relevance to real scenarios. We propose both ILP and CP methods to solve the PDDP. Regarding ILP, we give two formulations derived from a classic and a state-of-the-art model for p-dispersion, respectively. Regarding CP, we first give a generic model that can be implemented within any standard CP solver, and we then propose a specialized heuristic Branch&Bound method. Experiments demonstrate that the ILP formulations are more efficient than the CP model, as the latter is unable to prove optimality in reasonable time, except for small problems, and is usually slower in finding solutions of the same quality than the ILP models. However, although the ILP approach displays good performance on small to medium size problems, it cannot efficiently handle larger ones. The heuristic CP-based method can be very efficient on larger problems and is able to quickly discover solutions to problems that are very hard for an ILP solver.

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

Keywords and phrases Facility location, distance constraints, optimization

Digital Object Identifier 10.4230/LIPIcs.CP.2023.30

1 Introduction

Maximum diversity problems arise in many practical settings from facility location to telecommunications and social network analysis [28]. Arguably, the most famous such problem is the (*maxmin*) *p*-dispersion problem (PDP) [29]. In the PDP we are given a set of candidate locations $P = \{1, 2, \dots, n\}$ for p facilities and an $n \times n$ matrix $(D[i, j])$, $i, j \in P$ with distances between candidate locations i and j . The goal is to select p items from P to locate the facilities such that the minimum distance between any pair of facilities is maximized.

In practice, the PDP occurs whenever a close proximity of facilities is dangerous or for other reasons undesirable. A standard application is concerned with the location of power plants, where we wish to minimize the risk of losing multiple plants in the event of an accident or an enemy attack. To achieve this, locations for the plants are desired so that interplant distances are as large as possible. Similar applications arise in the military sector, as it is common to scatter military installations in order to make it difficult for the enemy to disarm all of them. In a more peaceful context, we may wish to disperse branches of the same franchise, so that mutual competition between similar shops is minimized, or public facilities which have overlapping areas of service, e.g., schools, hospitals, electoral districts,



© Nikolaos Ploskas, Kostas Stergiou, and Dimosthenis C. Tsouros;
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 30; pp. 30:1–30:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

etc. [25, 16, 28]. In telecommunications, we may wish to disperse radio transceivers to service cellular phones so as to minimize interference. Another, more recent, area of application for the PDP, is if distances are not interpreted physically but as a measure of the diversity between members of a group [33].

Another dispersion problem that has been deeply studied is the maxsum p-dispersion problem, called p-defence problem in [29], where we seek to locate p facilities so that the sum of distances between the facilities is maximized [25]. Although the maxsum variant also has many applications, it is recognized that the PDP is better suited to model situations where the close proximity of facilities must be definitely avoided (e.g. for safety reasons). This is because maximizing the sum of distances does not guarantee that no two facilities will be placed close to each other [25].

In this paper we are concerned with a variant of the PDP where hard constraints specifying minimum allowed distances exist between facilities. We call this problem *p-dispersion with distance constraints* (PDDP). Although this problem was put forward by Moon and Chaudhry who first systematically studied location problems with distance constraints and coined the term p-dispersion [29], it has been rather ignored since, despite its relevance to real applications. Distance constraints in dispersion problems can stem from operational needs and regulations, such as clearance distances for safe chemical storage [1], separation distances between packages containing radioactive materials [40], or between portable fire extinguishers in an area [41]. Motivated by such applications, a recent study by Dai et al. considers p-dispersion with distance constraints in the context of circle placement in non-convex polygons [10].

To further motivate our study of the PDDP, consider a scenario where p identical power plants need to be located in an area. Assume that due to safety reasons, any two plants must be more than x km away from one another. If this problem is modeled and solved as a PDP then either of the following two results will hold: 1) The optimal solution places the two closest facilities $y \leq x$ km apart. Then the original problem is infeasible, as the safety requirements cannot be satisfied for all pairs of facilities, 2) The optimal solution places the two closest facilities $y > x$ km apart. Then, as y is the minimum distance between any two plants, all the safety requirements are satisfied and the original problem has been solved. But what if all power plants to be located are not identical? What if there are differences in the plants' sizes, the volume of power generated, the waste produced, etc.?

In such cases, the safety requirements regarding the minimum allowed distances between plants may not be the same for all pairs of plants. For instance, the allowed distance between smaller and less dangerous plants will probably be smaller than between larger ones. Hence, the PDP does no longer suffice to model the problem. This is because an optimal solution that places the closest plants y km apart does not guarantee that the safety distances will be satisfied for all pairs of plants. The case of non-identical (heterogeneous) facilities has not received much attention in the p-dispersion literature, as the predominant explicit or implicit assumption is that the facilities to be located are indistinguishable (homogeneous). But in practice, not all power plants will be identical, and the same holds for the branches of a franchise, for public facilities, and almost any type of facilities that we want to disperse.

We start our study of the PDDP by giving two ILP formulations. The first one is based on the classic formulation for p-dispersion by Kuby [25], while the second is based on a state-of-the-art model proposed by Sayah and Irnich [34]. Both these formulations are extended to deal with heterogeneous facilities and to include distance constraints.

Then, we describe a generic CP model for the PDDP that can be implemented within any standard CP solver. For the purposes of this study, we have implemented this model in OR-Tools and Choco. Experimental results demonstrate that the ILP formulations,

implemented in Gurobi, are more efficient than the CP solvers, as the latter find it very hard to prove optimality, even for small problems, and are usually slower in locating solutions of the same quality as the ILP solver.

We further explore the applicability of CP technology by introducing a specialized heuristic method based on CP. Specifically, using a simpler model of the problem, with fewer variables and constraints, we have devised a specialized Branch&Bound mechanism, which has been implemented in a custom CP solver. This method tries to prune the search tree early by estimating the best cost that can be achieved if the sub-tree rooted at the currently visited node is explored. If the estimated cost does not improve the cost of the best solution found so far then the current branch is abandoned. The estimation is carried out through a simple greedy assignment of the remaining variables. This reasoning achieves significant search tree pruning, albeit by sacrificing completeness.

In our experimental analysis we first compare the ILP formulations to the CP approaches on randomly generated problems with 5-30 facilities and at most 80 potential location points. Results show that the CP model implemented in the standard CP solvers OR-Tools and Choco cannot compete with the ILP formulations implemented in Gurobi, as the latter solver is quite efficient in all but the largest class that contains 30 facilities and 80 location points. The heuristic CP approach very quickly finds solutions, often optimal ones, on all instances, including those of the hardest class. We then consider harder problems that are generated using the p-dispersion MDPLIB benchmark library as basis [28]. Results demonstrate that the ILP formulations are efficient on problems with 10 facilities and 100 potential locations, but fail to efficiently handle larger problems. On the other hand, the heuristic CP approach can trivially find solutions of good quality on smaller instances, while it can also handle larger instances that are very hard for the ILP solver, finding solutions of much better quality.

2 Related Work

The maxmin p-dispersion problem, which is \mathcal{NP} -hard on general networks for an arbitrary p [18], was originally mentioned by Shier, as far back as 1977 [36]. However, the term p-dispersion first appeared in the analysis of location problems with distance constraints by Moon and Chaudhry [29]. The first ILP solution was proposed by Kuby [25] while the first specialized algorithm was given by Erkut [15]. Kincaid proposed simulated annealing and tabu search methods [24], Ghosh proposed a multi-start heuristic [18] and Resende et al. applied the GRASP methodology to the maxmin problem [32]. More recently, Sayyady & Fathi [35] and Sayah & Irnich [34] proposed ILP approaches to the maxmin problem, which are able to solve large size problems, and as argued in the comprehensive review on OR methods for dispersion problems given in [28], they tend to make heuristic approaches obsolete, as they can handle problems of similar size.

Regarding distance constraints, Moon and Chaudhry were the first to systematically study location problems with distance constraints [29]. The p-dispersion problem with distance constraints was mentioned by them as a problem that can arise in real-life scenarios, but no approaches towards solving it were proposed. Recently, Dai et al. revisited this problem as part of a study on circle (i.e. facility) dispersion in non-convex polygons [10]. A heuristic method, inspired by the mechanics of the n-body problem in physics, was proposed for the plain p-dispersion problem in non-convex polygons, and this method was also adapted to the case where distance constraints exist between pairs of circles.

Distance constraints have also been considered in the context of other location problems. Some early works considered maximum distance constraints between the demand nodes and the facility locations [7, 8, 23, 38]. Tansel et al. studied the distance constrained *p-center*

problem for the case where the network is a tree [38]. Chaudhry et al. proposed heuristics for selecting a maximum-weight set of locations such that no two are closer than a given distance from each other [6]. Moon and Papayanopoulos 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 [30]. Comley studied the problem of locating a small number of heterogeneous semi-obnoxious facilities that interact with each other as well as with other existing facilities [9].

Berman and Huang 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 pre-specified distance [2]. Drezner et al. 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 [13]. Drezner et al. 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 [14]. Welch and Salhi studied the location of obnoxious facilities with interactions between them [39]. Location problems with distance constraints that restrict the placement of facilities near certain demand points have also been studied, e.g. [31, 4, 27].

There are very few CP-related methods for facility location problems [17, 5, 37] and none of them concerns p-dispersion problems, with or without distance constraints. Regarding our heuristic CP-based method, there are works that follow a similar approach, i.e. sacrificing the completeness of a CP solver to solve optimization problems faster [22, 26, 19]. However, these works typically do this through a more local reasoning, e.g. by adding extra constraints.

Finally, the p-median problem with distance constraints, originally put forward in [29], is being studied in a paper that is currently under review (details are suppressed to preserve anonymity). In such a problem, there exist both facilities and clients that are serviced by the facilities. The goal is to locate p facilities so that the sum of the distances between the clients and their closest facility is minimized. As here, ILP and CP models for this problem are proposed and compared. Results demonstrate that the ILP approach is by far the most efficient on problems with homogeneous facilities, but it is outperformed by a heuristic CP approach on some classes of problems with heterogeneous facilities.

3 Background

In a p-dispersion with distance constraints problem (PDDP), p facilities in a set of facilities F are to be placed in an area. We assume that the set P of potential location points for the facilities is known. We also assume that the distance between each pair of potential locations (i, j) is given in a matrix D . 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. To summarize, in a PDDP 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 candidate facility locations.
- d_{ij} : the lower bound in the allowed distance between each pair of facilities (i, j) , where $i, j \in F$.

The distance between two points i and j can be given by the straight-line (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 location points are given in a 2-d distance matrix D . 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.

A common assumption in the literature on location problems with distance constraints is that the lower distance bound d_{ij} is fixed to a specific value for all the constraints between facilities. 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, as for example in [1, 40]. In case the facilities are heterogeneous, the distance bound may vary from constraint to constraint.

The goal in a PDDP is to locate each facility to a node so that the minimum distance between any two facilities is maximized subject to the satisfaction of all the distance constraints.

4 ILP models

We first give an ILP model for the PDDP, based on the classic formulation of Kuby for p-dispersion [25] and then we give a model based on the state-of-the-art model of Sayah & Irnich [34]. Both formulations are extended to deal with heterogeneous facilities and to include distance constraints between facilities.

4.1 Kuby based model

We make use of the following additional notation:

- $C = \{(i, j, f_1, f_2) \mid i, j \in P, f_1, f_2 \in F, D[i, j] \leq d_{f_1 f_2}\}, \forall i \in P, \forall j \in P$, 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 .
- $x_{ij} = 1$ if a facility $j \in F$ is located at a facility site $i \in P$ and 0 otherwise.
- $y_i = 1$ if any facility is located at a facility site $i \in P$ and 0 otherwise.
- b is the minimum distance between the facilities that we aim to maximize.

For any $i \in P$, variable y_i shows whether or not facility site i will host any facility. These are the variables that are present in Kuby’s formulation for p-dispersion. Given that facilities are considered identical in the p-dispersion literature, in Kuby’s model we only need to know whether a site will host a facility or not. But in the case of the PDDP, where facilities can be different and distance constraints exist between them, we also need to know which particular facility will be hosted by a site. Hence, we introduce $|P| \times |F|$ 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 $i \in P$.

The extension of Kuby's model to capture the PDDP can be expressed as:

$$\max \quad b \quad (1)$$

$$\text{s.t.} \quad \sum_{i \in P} y_i = p \quad (2)$$

$$b \leq M(2 - y_i - y_j) + D[i, j] \quad \forall i, j \in P, j > i \quad (3)$$

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

$$\sum_{j \in F} x_{ij} \leq 1 \quad \forall i \in P \quad (5)$$

$$\sum_{i \in P} x_{ij} = 1 \quad \forall j \in F \quad (6)$$

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

$$x_{ij} \in \{0, 1\} \quad \forall i \in P, \forall j \in F \quad (8)$$

$$y_i \in \{0, 1\} \quad \forall i \in P \quad (9)$$

$$b \geq 0 \quad (10)$$

The objective function 1 aims at maximizing the shortest distance b between located facilities. Constraint 2 specifies that p facilities are to be located. Constraint 3 guarantees that $b \leq D[i, j]$ whenever both locations i and j are chosen via $y_i = y_j = 1$ for the location of facilities, where M represents a Big M constant. Variables b and $y_i, i \in P$, the objective function 1 and Constraints 2, 3, 9 and 10 form the original formulation of Kuby for the p-dispersion problem.

As we explained, in the case of the PDDP, we add variables $x_{ij}, \forall (i, j), i \in P, j \in F$, in order to know whether or not a specific facility $j \in F$ is located at a facility site $i \in P$. These variables are linked to the y_i variables via Constraint 4, which specifies that if any facility j is located at a facility site i , then variable y_i equals 1 and 0 otherwise. To ensure that no two variables x_{ij} and $x_{ij'}$ are set to 1 (i.e. each facility site must host at most one facility), we add Constraint 5. To ensure that no two variables x_{ij} and $x_{i'j}$ are set to 1 (i.e. each facility must be hosted at exactly one facility site), we add Constraint 6. Finally, 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 . These pairwise constraints are a special case of clique constraints and are an efficient option to model distance constraints in ILP, as demonstrated in [2].

The original Kuby model for the p-dispersion problem has $|P|+1$ variables and $\sum_{i=1}^{|P|-1} i+1$ constraints, while our extended Kuby based model for the PDDP has $|P| \times |F| + |P| + 1$ variables and $2 \times |P| + |F| + \sum_{i=1}^{|P|-1} i + 1$ constraints, without considering Constraint 7 which can give $(|P| \times |F|)^2$ constraints.

4.2 Sayah and Irnich based model

We now present a model for the PDDP based on the PDP model proposed by Sayah and Irnich, which utilizes the fact that the optimal distance is equal to at least one of the entries of the distance matrix [34]. Let us introduce some additional notation for this model:

- $E = \{(i, j) \in P \times P : i < j\}$: the set of edges between any two candidate facility locations.
- $E(l) = \{(i, j) \in E : D[i, j] < l\}$: a subset of edges given any distance l .
- $L^0 < L^1 < \dots < L^{k_{max}}$: the different nonzero values in $D[i, j]$. The associate index sets are $K = \{1, 2, \dots, k_{max}\}$ and $K_0 = \{0\} \cup K$. By definition, $\emptyset = E(L^0) \subsetneq E(L^1) \subsetneq \dots \subsetneq E(L^{k_{max}}) \subsetneq E$ holds.
- $z_k = 1$ if the location decisions satisfy a minimum distance of at least L^k , $k \in K$ and 0 otherwise.

Similar to the Kuby model, variable y_i , for any $i \in P$, shows whether or not facility site i will host any facility. In addition, variable z_k , for any $k \in K$, indicates whether the location decisions satisfy a minimum distance of at least L^k . These are the variables that are present in the formulation of Sayah and Irnich for p-dispersion. As we explained in Kuby's model, we also need to introduce $|P| \times |F|$ 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 $i \in P$.

The model for the PDDP using Sayah and Irnich's model as basis can be expressed as:

$$\max \quad D^0 + \sum_{k \in K} (L^k - L^{k-1}) z_k \quad (11)$$

$$\text{s.t.} \quad \sum_{i \in P} y_i = p \quad (12)$$

$$z_k \leq z_{k-1} \quad \forall k \in K, k > 1 \quad (13)$$

$$y_i + y_j + z_k \leq 2 \quad \forall k \in K, (i, j) \in E(L^k) \setminus E(L^{k-1}) \quad (14)$$

$$\sum_{j \in F} x_{ij} = y_i \quad \forall i \in P \quad (15)$$

$$\sum_{j \in F} x_{ij} \leq 1 \quad \forall i \in P \quad (16)$$

$$\sum_{i \in P} x_{ij} = 1 \quad \forall j \in F \quad (17)$$

$$x_{if_1} + x_{jf_2} \leq 1 \quad \forall (i, j, f_1, f_2) \in C \quad (18)$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in P, \forall j \in F \quad (19)$$

$$y_i \in \{0, 1\} \quad \forall i \in P \quad (20)$$

$$z_k \in \{0, 1\} \quad \forall k \in K \quad (21)$$

The objective function 11 aims at maximizing the shortest distance between located facilities. Constraints 12, 15, 16, 17, 18, 19 and 20 are also present in the Kuby based formulation, while Constraints 13, 14 and 21 replace Constraints 3 and 10. Constraint 13 models the consistency between the z_k variables in the sense that the z_k variables are non-increasing in k , while Constraint 14 ensures that no pair (i, j) of locations with distance $D[i, j] < L^k$ is chosen simultaneously. The consistency Constraint 13 specifies that any feasible solution fulfills that there exists a unique $k \in K_0$ with $z_1 = z_2 = \dots = z_k = 1$ and $z_{k+1} = z_{k+2} = \dots = z_{k_{max}} = 0$. Variables $y_i, i \in P$, and $z_k, k \in K$, the objective function 11 and Constraints 12, 13, 14, 20 and 21 form the original formulation of Sayah and Irnich for the p-dispersion problem.

The addition of variables $x_{ij}, \forall (i, j), i \in P, j \in F$, in the revised Sayah and Irnich model for the PDPP yields Constraints 15, 16, 17, and 19, as already explained in Kuby's model. In addition, Constraint 18 models the distance constraints between facilities.

The original model of Sayah and Irnich for the p-dispersion problem has $|P| + k_{max} - 1$ variables and $k_{max} + (k_{max} - 1) \times \sum_{i=1}^{|P|-1} i - 1$ constraints, while our extended Sayah and Irnich based model for the PDDP has $|P| \times |F| + |P| + k_{max} - 1$ variables and $2 \times |P| + |F| + k_{max} + (k_{max} - 1) \times \sum_{i=1}^{|P|-1} i - 1$ without considering Constraint 18 which can give $(|P| \times |F|)^2$ constraints.

5 CP approaches to the PDDP

We first give a generic CP model of the PDDP and we then we describe the mechanics of a specialized heuristic CP solver. The PDDP is modeled as a Constraint Optimization Problem (COP) (X, Dom, C, O) , where X is the set of decision variables, 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 $|X| = |F| = p$. 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 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 $y_{ij} \in Y$ and a constraint $y_{ij} = D[x_i, x_j]$. Hence, each $y_{ij} \in Y$ models the distance between x_i and x_j . In CP solvers, this is implemented using the Element global constraint, i.e. $y_{ij} = Element(D, [x_i, x_j])$.
3. For each variable y_{ij} , there is a distance constraint $y_{ij} > d_{ij}$.
4. There is a variable z , s.t. $z = \min(Y)$.
5. The objective function is $O = maximize(z)$.

This model contains $p + p \times (p - 1)/2 + 1$ variables, with p being decision variables and the rest auxiliary, and $p \times (p - 1) + 1$ constraints.

We also considered the use of an AllDifferent constraint, to speed up propagation. However, the distance constraints already propagate the fact that facilities should be placed at different locations, as they all have bound greater than 0. Experiments with and without the AllDifferent constraint showed no noticeable difference in run times.

5.1 A heuristic CP-based method

We now propose a heuristic technique that tries to prune early the parts of the search tree that do not seem promising, i.e. it is unlikely that exploring them will improve the value of the optimization function. At each node of the search tree this method tries to estimate the best value of the optimization function that can be achieved if we explore the sub-tree rooted at that node. If this value is not better than the cost of the best solution found so far then the current branch is not further explored.

The proposed method can be embedded in a standard CP solver. However, this cannot be done at the modeling level by just specifying variables and posting constraints, because it requires writing specialized code within the solver and possibly a intervention in how the search process works. Naturally, the estimation of the bound at each node cannot always be precise (otherwise we would be able to trivially solve the PDDP), and therefore, a solver that employs this method will not be exact.

To demonstrate our heuristic method, we describe a simple CP solver, specialized for the PDDP, that applies it. This solver uses a simpler model of the problem, dropping the auxiliary variables and relevant constraints. Hence, we now have a model with only the p decision variables and $p \times (p - 1)/2$ distance constraints. The optimization function is handled within the solver in the following way: Whenever a new solution is found, its cost is computed so as to determine if this cost is better than the current best cost. If so, then the best cost found so far is updated. If such an update occurs, it will be propagated to the decision variables, as described below.

The heuristic pruning technique works as follows: 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, an upper bound for the best possible solution under the current assignment is computed. This upper bound gives 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 best cost 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.

The computation of the upper bound estimation at each node is performed in a greedy fashion. 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 and v_i is the value under consideration for x_i , we greedily compute the cost of the “best” assignment for the future variables x_{i+1}, \dots, x_p . That is, we visit 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 $v_j \in \text{Dom}(x_j)$, we find the minimum distance between v_j and any assignment (location) among variables (facilities) x_1, \dots, 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. We then compute the cost of this complete assignment, which gives the upper bound, but in fact, this may be an underestimation of the real cost. If the computed cost is equal to or lower than the current lower bound then the assignment of v_i to x_i is undone and the current branch of the search tree is abandoned, albeit risking to prune the branch leading to the optimal solution. This process is depicted by Algorithm 2.

Algorithm *PDDP_CP_Solver* (Algorithm 1) gives a high level description of the entire solving process.

■ **Algorithm 1** *PDDP_CP_Solver*(X, Dom, C, O).

```

if Propagate( $X, \text{Dom}, C$ ) = FALSE
  return NULL;
 $depth \leftarrow 1$ ;
 $best\_found \leftarrow 0$ ;
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 has not been tried;
    if  $depth = n$ 
       $cur\_cost \leftarrow \text{Compute\_Solution\_Cost}(X, \text{Dom}, C)$ 
      if  $cur\_cost > best\_found$ 
         $best\_found \leftarrow cur\_cost$ ;
    else if Propagate( $X, \text{Dom}, C, x_i \leftarrow a$ ) = TRUE
      if  $best\_found \neq 0$  AND Bound( $X, \text{Dom}, C, x_i \leftarrow a, best\_found$ ) = TRUE
         $depth \leftarrow depth + 1$ ;
        select an unassigned variable  $x_i$ ;
if  $best\_found = 0$  return NULL;
return  $best\_found$ ;

```

Given a PDDP (X, DOM, C, O) , where O is the optimization function of the PDDP, the algorithm starts by propagating the hard constraints in C , as a typical CP solver does. Function *Propagate* enforces arc consistency on the distance constraints. If no failure (empty domain) is detected then the algorithm initializes the depth to 1 and the best cost found ($best_found$) to 0 and commences the search by selecting a variable using a variable ordering heuristic. While the depth of search is greater than 0, denoting that the search space has not been exhaustively searched, a branching decision is made, i.e. a value is selected and assigned to the currently selected variable. If all the variables have been assigned ($depth = n$), meaning that a feasible solution has been found, the cost of this solution is computed and if this cost is higher than the best cost found so far then the latter is updated.

If not all variables have been assigned yet then Function *Propagate* is called to propagate the value assignment just made. If no failure occurs, the heuristic bounding mechanism described above is triggered by calling Function *Bound* (Algorithm 2), provided that at least one feasible solution has already been found. If this function succeeds, meaning that the

30:10 The p-Dispersion Problem with Distance Constraints

estimated cost is better than the best bound found so far then the algorithm moves forward by increasing the depth of search and selecting a new unassigned variable. On the other hand, if either 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.

■ **Algorithm 2** *Bound*($X, Dom, C, x_i \leftarrow v_i, best_found$).

```
for each  $x_j, i + 1 \leq j \leq p$ 
   $val \leftarrow dis \leftarrow -1$ ;
  for each  $v_j \in Dom(x_j)$ 
     $temp \leftarrow$  shortest distance between  $v_j$  and any assigned variable  $x_1 \dots x_{j-1}$ ;
    if  $temp > dis$ 
       $dis \leftarrow temp$ ;
       $val \leftarrow v_j$ ;
   $x_j \leftarrow val$ ;
 $bound \leftarrow Compute\_Solution\_Cost(X, Dom, C)$ ;
if  $bound > best\_found$  return TRUE;
return FALSE;
```

If the value *best_found* remains 0 upon termination 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 -1 occurs, proving that the problem is infeasible.

Function *Propagate* applies arc consistency on the distance constraints, taking into consideration the value of *best_found*, i.e. the best cost found so far, to perform extra pruning, if possible. This is done in typical CP fashion for binary constraints, i.e. using a queue to insert and then process variables that have their domain filtered. Specifically, if a variable x_i is removed from the queue then for each variable x_j constrained with x_i , and each value $v_j \in Dom(x_j)$, we check if there exists a value v_i in $Dom(x_i)$ s.t. the two values satisfy the distance constraint between x_j and x_i **and** the distance between the two values is greater than *best_found*. In other words, for each possible location v_j of x_j we search for a location v_i for x_i s.t. by placing the two facilities at these locations, not only the relevant distance constraint is satisfied, but we can also improve the cost of the optimization function. If no such v_i exists then by placing x_j at v_j there is no way to locate x_i so that we can satisfy the distance constraint and improve the cost. Hence, v_j can be deleted from $D(x_j)$, i.e. it can be removed from consideration as a potential location point for x_j . If such a value deletion occurs then variable x_j is inserted in the queue to propagate the deletion.

Finally, note that the pruning that can be achieved by taking into consideration the current best cost can also be achieved by a standard CP solver that employs the model described further above. Such a solver will typically add a constraint $z > best_found$ once a new solution with better cost than the previously found solutions is located. The propagation of this constraint may result in the filtering of the y_{ij} variables' domains, which in turn will be carried over to the decision variables through the distance constraints $y_{ij} = D[x_i, x_j]$.

6 Experimental Results

We experimented with PDDP instances generated in two different ways. The first is a simple method that randomly generates a PDDP with a desired number of facilities and location points. The second uses the p-dispersion benchmark library MDPLIB 2.0 [28] as a basis to

generate PDDPs. 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.

The ILP models were solved using Gurobi 9.0.3 [21]. The exact CP model was written in the CPMpy modeling tool [20] and compiled into OR-Tools [12]. An implementation in Choco [11] was also tried. The heuristic CP method was implemented in a custom solver programmed in C. This solver essentially implements the *CP_Solve* algorithm (Algorithm 1) described above. Choco and the heuristic solver use the dom/wdeg heuristic for variable ordering [3] and lexicographic value ordering, while OR-Tools uses its default options. A time out of 3,600 seconds was set for each instance. We used only one thread on all solvers, to get a fair comparison.

The ILP models are stored in compressed sparse column format since the constraint matrix can sometimes be too large to be stored as a full array in memory. For example, the largest instance considered in the computational study has a constraint matrix of 5,425,061 rows, 105,038 columns and 11,212,542 nonzeros. Its compressed size is only 12MB, while its size as a full matrix is 530GB.

6.1 Random problems

For an initial evaluation of the proposed approaches to the PDDP, we ran experiments on problems where we try to locate $p \in \{5, 10, 20, 30\}$ facilities in a 10×10 grid, with $|P| \in \{30, 80\}$ potential location points selected randomly among the 100 total points. The distances between the points are computed using the Manhattan distance metric. For each distance constraint $dis(f_i, f_j) > d_{ij}$ between facilities f_i and f_j , d_{ij} was set to a random integer in the interval $[1, max]/2$, where max is the maximum Manhattan distance between any two potential location points. Ten instances were generated and solved for each combination of parameter values.

Table 1 compares the following: Our two ILP formulations implemented in Gurobi, with $Gurobi_k$ denoting our first model, based on that of Kuby, and $Gurobi_s$ denoting our second model, based on that of Sayah & Irnich, and the CP solvers OR-tools and Choco. For each class, in column $\sum cpu$ we give the total cpu time taken by the corresponding solver over all 10 instances, and in brackets we give the number of instances on which the solver timed out. If the solver timed out on at least one instance then $\sum cpu$ gives a lower bound on the actual run time. Column cpu_o gives the mean time taken by the solver to locate the optimal solution. A zero means that less than 0.1 seconds were taken on average. In brackets, we give the number of instances for which the solver found the optimal solution. Note that the optimal solutions are known for all instances because at least one solver terminated within the time limit. When a solver managed to find the optimal solution on at least 8 out of the 10 instances, we compute cpu_o , excluding the instances where it timed out. Otherwise, cpu_o is left blank (-), meaning that the solver failed to find the optimal solution on too many instances for this metric to be meaningful.

From Table 1 it is clear that the ILP approach is more efficient than the CP one in this type of instances. $Gurobi_k$ (resp. $Gurobi_s$) times out on 1 (resp. 2) out of the 70 instances, whereas OR-Tools (resp. Choco) timed out on 40 (resp. 42) instances. With respect to the cases when the optimal solution was located within the time limit, even without proving optimality, $Gurobi_k$ (resp. $Gurobi_s$) found the optimal solution on 69 (resp. 68 instances), whereas OR-Tools (resp. Choco) on 59 (resp. 45) instances. This indicates that the CP solvers find the proof of optimality especially difficult. Regarding the time required to find the optimal solution (when found), all solvers are quite fast on smaller problems (with 5-10

30:12 The p-Dispersion Problem with Distance Constraints

■ **Table 1** Comparing exact solvers on random PDDPs. Cpu times are given in seconds.

Class ($p, P $)	Gurobi _k $\sum \text{cpu}$	cpu _o	Gurobi _s $\sum \text{cpu}$	cpu _o	OR-Tools $\sum \text{cpu}$	cpu _o	Choco $\sum \text{cpu}$	cpu _o
(5,30)	0.7 (0)	0 (10)	1.6 (0)	0 (10)	3 (0)	0 (10)	3.5 (0)	0 (10)
(10,30)	2 (0)	0 (10)	6 (0)	0 (10)	998 (0)	1.7 (10)	>11,517 (2)	50 (10)
(20,30)	87 (0)	7 (10)	51 (0)	4 (10)	>10h (10)	69 (9)	>10h (10)	- (4)
(5,80)	7 (0)	0 (10)	20 (0)	0 (10)	13 (0)	0 (10)	3 (0)	0 (10)
(10,80)	12 (0)	0 (10)	78 (0)	0 (10)	>10h (10)	0.3 (10)	>10h (10)	0 (10)
(20,80)	727 (0)	69 (10)	725 (0)	28 (10)	>10h (10)	- (4)	>10h (10)	- (1)
(30,80)	>22,207 (1)	2,065 (9)	>10,519 (2)	414 (8)	>10h (10)	- (6)	>10h (10)	- (1)

facilities). Problems with 20 facilities and 30 potential locations are also easily manageable by the ILP models, whereas OR-Tools takes more than one minute on average to discover the optimal solution, and even fails to discover it on one instance. Choco fails even worse, failing to find the optimal solution on 6 instances, and failing to find any solution on one instance.

As the size of the problem grows, the ILP models, and even more so the CP solvers, find it harder to solve the instances. On problems with 30 facilities and 80 location points, the ILP models start giving time outs and take a long time to find the optimal solution, when they manage to do so, whereas the CP solvers, and especially Choco, fail to find the optimal on many instances from the (20,80) and (30,80) classes.

Table 2 compares the exact solvers to the heuristic CP solver (denoted CP_h). For this solver, we report the total cpu time taken over the 10 instances of each class ($\sum \text{cpu}$) and the mean cpu time taken to find the best solution it found within the time limit (cpu_b). We also report (in brackets) the number of instances in which the solver managed to find the optimal solution. Then, in the following columns, we give the mean cpu times taken by the exact solvers to find a solution that at least matches the cost of the best solution found by the heuristic solver. In this way, we can evaluate the worth of the heuristic CP method as a heuristic for the PDDP. If the exact solvers manage to quickly match the best solution found by the heuristic one then there is not much point in using the heuristic solver. Whereas, if the heuristic solver quickly discovers a solution that the exact ones take very long to match then it is worth considering this approach for the PDDP. If an exact solver only managed to find a solution as good as that found by the heuristic solver in some instances, we give in brackets the number of times that this occurred, and we consider only these instances for the computation of the mean cpu time.

■ **Table 2** Comparing solvers on random PDDPs. Cpu times are given in seconds.

Class ($p, P $)	CP _h $\sum \text{cpu}$	cpu _b	Gurobi _k	Gurobi _s	OR-Tools	Choco
(5,30)	0	0 (9)	0	0	0	0
(10,30)	0	0 (10)	0	0	1.7	50
(20,30)	2	0.2 (9)	5	1.7	65	160 (4)
(5,80)	0.1	0 (10)	0	0	0	0
(10,80)	1	0.1 (10)	0	0	0.3	0
(20,80)	2	0.2 (0)	3.5	3	6	0
(30,80)	36	3 (10)	2,218 (9)	1,051 (8)	1,760 (6)	3,453 (1)

As the results in Table 2 demonstrate, CP_h is very fast as it managed to complete all 10 instances of each class in at most 2 seconds, except for the (30,80) class, on which it only took 36 seconds. Importantly, it also discovered the optimal solution in 58 out of the 70

instances, including all instances of the hard class (30,80) class. On the other hand, it did not manage to discover the optimal in any instance of the (20,80) class. Comparing the run times (cpu_b) of CP_h to the exact solvers, results from all but the last class show that Gurobi manages to quickly match the best solution found by CP_h , even if the latter is faster on average. This does not always hold for OR-Tools which takes more than 1 minute on average in the (20,30) class. Choco takes 50 seconds on average in the (10,30) class and it manages to match the best solution of CP_h in only 4 instances of the (20,30) class. The benefits of the heuristic method in hard problems are demonstrated by the (30,80) class where it takes 3 secs on average to locate the optimal solution, whereas Gurobi_k and Gurobi_s require 2,218 and 1,051 secs respectively. OR-Tools found the optimal solution in only 6 out of the 10 instances of this class, taking 1,760 secs on average, while Choco managed to find the optimal in only one instance in 3,453 secs.

Finally, Table 3 takes a closer look at the performance of CP_h , with respect to the effect of the heuristic pruning method. To investigate this, we report the results obtained by the solver when the heuristic is deactivated (i.e. Function *Bound* is not called). In this case, the solver, denoted as CP_{-h} , operates as a typical CP solver. As in Table 1, we give the total cpu time taken, and in brackets the number of time outs, and the mean time taken to locate the optimal solution (the number of instances where the optimal solution was found is given in brackets). In the following column (cpu_h) we give the mean cpu time taken by CP_{-h} to find a solution that at least matches the cost of the best solution found by CP_h , and in brackets, the number of times that it managed to do so. The next two columns give the mean numbers of visited search tree nodes for CP_{-h} and CP_h (the entry is left blank if there were time-outs). The last two columns give the average number of calls to Function *Bound* in CP_h and the percentage of fails caused by this function (i.e. the percentage of branches pruned by the heuristic).

■ **Table 3** A closer look at the performance of the custom solver, with and without the heuristic.

(p, $ P $)	$\text{CP}_{-h} \sum \text{cpu}$	$\text{CP}_{-h} \text{cpu}_o$	$\text{CP}_{-h} \text{cpu}_h$	$\text{CP}_{-h} \text{nodes}$	$\text{CP}_h \text{nodes}$	calls	%fails
(5,30)	0.2 (0)	0 (10)	0 (10)	1,726	156	140	94
(10,30)	466 (0)	5 (10)	5 (10)	12.5M	229	218	95
(20,30)	>21,827 (3)	1 (10)	1 (10)	-	4,917	781	80
(5,80)	6 (0)	0.6 (10)	0.6 (10)	7,624	501	477	97
(10,80)	>10h (10)	- (5)	520 (5)	-	877	868	98
(20,80)	>10h (10)	- (0)	- (0)	-	835	816	98
(30,80)	>10h (10)	- (7)	1,044 (7)	-	17,207	12,995	81

Table 3 demonstrates the pruning power of our proposed heuristic. Without its use, the solver is able to handle the easier classes of problems, but not the harder ones, in accordance with standard CP solvers (Table 1). The solver is very successful compared to OR-Tools and Choco on the (20,30) class, as it times out in only 3 instances and finds the optimal solution in 1 sec on average. However, the solver fares badly on the (20,80) class where it is unable to find the optimal in any instance and actually finds worse solutions than CP_h in all instances. As for the (30,80) class, CP_{-h} finds the optimal solution in 7 instances, which is better than OR-Tools and Choco, but needs 1,044 secs on average to match the solution found by CP_h . Regarding the pruning achieved by the heuristic when it is activated, it is impressive that in most of the classes, there is a very high percentage of pruned branches over the total calls to the heuristic (up to 98%), which explains its success in speeding up search.

6.2 MDPLIB

The MDPLIB collects a large number of dispersion benchmarks (for both maxmin and maxsum p-dispersion) divided into various classes [28]. 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. We took instances from these classes, having 100-250 potential facility points and 10-20 facilities, and we generated 10 PDDPs for each instance by randomly adding distance constraints between the facilities. For each distance constraint $dis(f_i, f_j) > d_{ij}$ between facilities f_i and f_j , d_{ij} was set to a random number in the interval $[1, max]/2$, where max is the maximum distance between any two potential location points, as specified in the base MDPLIB instance.

Table 4 compares the exact solvers on the MDPLIB-based problems. We exclude Choco as it is clearly inferior to OR-Tools. For each class we give the number of the MDPLIB instance on which it is based, and in brackets the numbers of potential locations and facilities to be located, e.g. a1(100,10) for MDG. For each solver, 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 of the optimization function's value for the best solution found within the time limit. In the \sum cpu columns we give in brackets the number of instances in which the solvers timed out. In the #opt columns we give in brackets the number of times when optimality was proved. Finally, in some cases, a solver did not manage to find any solution within the time limit. In such cases there is a subscript in the value of cost, giving the number of instances in which at least one solution was found. In such a case, the value of cost is computed over these instances only. In GKD classes with 250 points and 20 facilities, OR-Tools suffered memory exhaustion and crashed. This is denoted with MEM in the \sum cpu column.

The data in Table 4 demonstrates that the PDDPs generated using MDPLIB instances as basis can be very hard for both the ILP and CP approaches. None of the solvers terminates within the time limit on any instance with 20 facilities, while problems with 15 and 10 facilities are also quite hard. In addition, there are some instances of the larger classes (e.g. GKDD1(250,20)) where the solvers are unable to discover any solution within 1 hour of cpu time, let alone the optimal one. Comparing ILP to CP, Gurobi, with any of the two formulations, is in general more efficient than OR-Tools. Gurobi_k (resp. Gurobi_s) found the optimal solution in 77 (resp. 74) out of the 220 instances, and proved optimality in 73 (resp. 70) instances, whereas OR-Tools did not prove optimality in any instance (as it timed out on all of them) and found the optimal in 10 instances only. However, OR-Tools often managed to find better solutions than Gurobi in hard classes with 20 facilities, as the cost columns indicate, for instance in classes MDGa2(100,20) and MDGb2(100,20). Comparing Gurobi_k to Gurobi_s, there is no clear winner in terms of run times, but the latter managed to find solutions of better quality than the former in most of the classes. However, Gurobi_s proved optimality or found the optimal solution in slightly fewer instances than Gurobi_k.

Table 5 compares CP_h to the exact solvers. For CP_h, we report the total cpu time it takes over the 10 instances of each class (and the number of time-outs in brackets), the mean time it takes to find its best solution (cpu_b), and the mean of the optimization function's value for the best solution it finds. For the exact solvers, we report the mean times taken to match the value of the best solution found by CP_h (cpu_h) columns, and the number of instances on which the solvers managed to find a solution that matches or improves the best solution found by CP_h (in brackets). If this number is 0 or close to 0 then the entry in the cpu_h column is left blank (-), as it is impossible or meaningless to compute the value of cpu_h. If a value in the cpu_b column is blank (-), e.g. GKDD1(100,20), then most of the 10 instances in this class were infeasible (in brackets we give the number of infeasible instances).

■ **Table 4** Comparing exact solvers on MDPLIB-generated PDDPs. Cpu times are given in seconds. The best mean value of the optimization function for each class is denoted in bold.

Class (p, P)	Gurobi _k ∑cpu	#opt	cost	Gurobi _s ∑cpu	#opt	cost	OR-Tools ∑cpu	#opt	cost
MDG									
a1(100,10)	>25,745 (3)	9 (7)	4.67	11,208 (0)	10 (10)	4.68	>10h (10)	7 (0)	4.59
a1(100,20)	>10h (10)	0 (0)	0.80 ₈	>10h (10)	0 (0)	1.16	>10h (10)	0 (0)	1.17
a2(100,10)	15,327 (0)	10 (9)	4.74	11,412 (0)	10 (10)	4.74	>10h (10)	1 (0)	4.36
a2(100,20)	>10h (10)	0 (0)	0.78 ₈	>10h (10)	0 (0)	0.83 ₉	>10h (10)	0 (0)	1.40
b1(100,10)	11,852 (0)	10 (10)	460.11	>29,171 (2)	10 (8)	460.11	>10h (10)	0 (0)	430.25
b1(100,20)	>10h (10)	0 (0)	102.42	>10h (10)	0 (0)	109.35	>10h (10)	0 (0)	77.33
b2(100,10)	10,305 (2)	8 (8)	459.65	27,894 (2)	8 (8)	459.99	>10h (10)	1 (0)	413.85
b2(100,20)	>10h (10)	0 (0)	106.77 ₉	>10h (10)	0 (0)	81.09	>10h (10)	0 (0)	113.33
GKD									
d1(100,10)	4,743 (0)	10 (10)	34.06	5,415 (0)	10 (10)	34.06	>10h (10)	0 (0)	33.04
d1(100,20)	>10h (10)	0 (0)	-	>10h (10)	0 (0)	-	>10h (10)	0 (0)	-
d1(250,10)	>15,448 (4)	6 (6)	35.98	34,753 (9)	3 (1)	35.27	>10h (10)	0 (0)	-
d1(250,20)	>10h (10)	0 (0)	9.95 ₄	>10h (10)	0 (0)	10.55 ₂	MEM	0 (0)	-
d2(100,10)	>5,998 (1)	9 (9)	34.34	2,602	10 (10)	34.82	>10h (10)	0 (0)	31.18
d2(100,20)	>10h (10)	0 (0)	-	>10h (10)	0 (0)	-	>10h (10)	0 (0)	-
d2(250,10)	>22,334 (6)	4 (4)	35,94	>33,595 (8)	2 (2)	36.31	>10h (10)	0 (0)	-
d2(250,20)	>10h (10)	0 (0)	-	>10h (10)	0 (0)	-	MEM	0 (0)	-
SOM									
a21(100,10)	>20,260 (1)	10 (9)	5	9,585 (0)	10 (10)	5	>10h (10)	1 (0)	4.1
a21(100,15)	>34,887 (9)	1 (1)	2.2	>35,371	1 (1)	2.2	>10h (10)	0 (0)	2
a21(100,20)	>10h (10)	0 (0)	1 ₆	>10h (10)	0 (0)	1 ₆	>10h (10)	0 (0)	1 ₉
a41(150,15)	>10h (10)	0 (0)	2.6	>10h (10)	0 (0)	2.6	>10h (10)	0 (0)	3
a41(150,20)	>10h (10)	0 (0)	1 ₉	>10h (10)	0 (0)	1.11 ₉	>10h (10)	0 (0)	1
b5(200,20)	>10h (10)	0 (0)	1.4	>10h (10)	0 (0)	2	>10h (10)	0 (0)	-

Results from Table 5 demonstrate the efficiency of the heuristic CP approach. Regarding run times, CP_h times out in only 7 instances and terminates quickly in all instances of all classes, except for the hard GKDD1(250,20) and GKDD2(250,20). CP_h proved the infeasibility of the 16 infeasible instances of classes GKDD1(100,20) and GKDD2(100,20) and found solutions in the other 4, whereas none of the other solvers managed to do so in any instance. Of course, the fast proof of infeasibility is not due to the bounding mechanism of CP_h, as this is not invoked, but most likely due to the lightweight model and mechanics of the custom-written solver.

Regarding the quality of the solutions, as a downside, CP_h finds the optimal in only 2 out of the 80 instances of smaller size, for which the optimal is known (excluding infeasible ones). However, in these classes, the exact solvers (and especially OR-Tools) can be orders of magnitude slower than CP_h in discovering solutions of the same quality as CP_h, which reaches its best solution in less than 0.1 secs in most cases. This is evident in class SOMa41(150,15) where CP_h found its best solution in less than 0.1 secs on average, while the exact solvers took more than 1,000 secs to match the solution quality of CP_h, on instances where they managed to do this. However, the best solution discovered by these solvers (including OR-Tools) is typically better than the best solution discovered by CP_h.

■ **Table 5** Comparing solvers on MDPLIB-generated PDDPs. We denote with bold the mean cost of CP_h on classes where it was better than the mean cost of all the other solvers.

Class ($p, P $)	CP_h $\sum \text{cpu}$	cpu_b	cost	Gurobi _k cpu_h	Gurobi _s cpu_h	OR-Tools cpu_h
MDG						
a1 (100,10)	0.5 (0)	0	4.35	243 (10)	55 (10)	592 (10)
a1 (100,20)	81 (0)	8	1.69	- (0)	- (1)	- (0)
a2 (100,10)	0.6 (0)	0	4.24	137 (10)	22 (10)	873 (7)
a2 (100,20)	137 (0)	12	1.64	- (0)	- (0)	- (0)
b1 (100,10)	0.4 (0)	0	428.18	10 (10)	345 (10)	187 (10)
b1 (100,20)	54 (0)	4.5	181.20	- (0)	- (0)	- (1)
b2 (100,10)	0.7 (0)	0	428.17	27 (10)	41 (10)	723 (3)
b2 (100,20)	92 (0)	8	159.93	- (0)	- (0)	- (1)
GKD						
d1 (100,10)	1.8 (0)	0.1	33.27	94 (10)	224 (10)	717 (5)
d1 (100,20)	405 (0)	- (9)	-	- (0)	- (0)	- (0)
d1 (250,10)	10 (0)	0.6	34.24	800 (8)	2,074 (6)	- (0)
d1 (250,20)	>18,689 (3)	1,140	16.94₉	- (0)	- (0)	- (0)
d2 (100,10)	1.6 (0)	0	31.29	91 (9)	179 (10)	1,277 (6)
d2 (100,20)	962 (0)	- (7)	-	- (0)	- (0)	- (0)
d2 (250,10)	8 (0)	0.5	35.06	121 (6)	997 (8)	- (0)
d2 (250,20)	>23,767 (4)	1,597	13.28₉	- (0)	- (0)	- (0)
SOM						
a21 (100,10)	0 (0)	0	4	7 (10)	3 (10)	106 (10)
a21 (100,15)	0 (0)	0	2	39 (10)	45 (10)	206 (10)
a21 (100,20)	11 (0)	1	1	41 (6)	49 (6)	522 (9)
a41 (150,15)	2 (0)	0	3	1,170 (6)	1,741 (6)	1,465 (10)
a41 (150,20)	12 (0)	1	1.9	- (0)	- (1)	- (1)
b5 (200,20)	10 (0)	0.3	2	- (4)	1,272 (10)	- (0)

But the power of CP_h as a heuristic method for PDDP is truly evident on the larger classes with 20 facilities where it discovers solutions of (much) better quality than the exact solvers, and excluding the two hard GKD classes with 250 location points and 20 facilities, it does this very fast. Also, CP_h finds solutions in all 20 instances of the two hard GKD classes, while Gurobi_k (resp. Gurobi_s) in only 4 (resp. 2), and OR-Tools in none.

7 Conclusion

We have studied a variant of the p-dispersion problem where distance constraints exist between the facilities to be dispersed. We proposed ILP formulations and a CP model for this problem. We also devised a heuristic CP-based method built around a bounding technique that prunes the search tree by reasoning about the best possible value of the optimization function at each node. Experimental results demonstrated that although the ILP formulations are more efficient than the CP model, they fail to efficiently handle problems with more than 10 facilities, whereas on such problems the heuristic CP method manages to find solutions of better quality than the ILP and CP models in orders of magnitude shorter run times.

References

- 1 T. Argo and E. Sandstrom. Separation distances in NFPA codes and standards (tech. rep.). Fire Protection Research Foundation. 2014.
- 2 O. Berman and R. Huang. The minimum weighted covering location problem with distance constraints. *Computers and Operations Research*, 35(12):356–372, 2008.
- 3 F. Boussemart, F. Heremy, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI'04*, pages 482–486, 2004.
- 4 J. Brimberg and H. Juel. A minisum model with forbidden regions for locating a semi-desirable facility in the plane. *Location Science*, 6(1):109–120, 1998.
- 5 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.
- 6 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.
- 7 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.
- 8 V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.
- 9 W. Comley. The location of ambivalent facilities: Use of a quadratic zero-one programming algorithm. *Applied Mathematical Modeling*, 19(1):26–29, 1995.
- 10 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.
- 11 Choco development team. An Open-Source java library for constraint programming. <https://choco-solver.org/>.
- 12 OR-Tools development team. OR-Tools, CP-SAT solver. https://developers.google.com/optimization/cp/cp_solver.
- 13 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.
- 14 Z. Drezner, P. Kalczyński, and S. Salhi. The planar multiple obnoxious facilities location problem: A Voronoi based heuristic. *Omega*, 87:105–116, 2019.
- 15 E. Erkut. The discrete p-dispersion problem. *European Journal of Operational Research*, 46(1):48–60, 1990.
- 16 E. Erkut and S. Neuman. Analytical models for locating undesirable facilities. *European Journal of Operational Research*, 40(3):275–291, 1989.
- 17 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.
- 18 J. B. Ghosh. Computational aspects of the maximum diversity problem. *Operations Research Letters*, 19(4):175–181, 1996.
- 19 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.
- 20 T. Guns. 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*, 2019.
- 21 LLC Gurobi Optimization. Gurobi optimizer reference manual. , 2023. URL: <https://www.gurobi.com>.
- 22 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.

30:18 The p-Dispersion Problem with Distance Constraints

- 23 B. M. Khumawala. An efficient algorithm for the p-median problem with maximum distance constraints. *Geographical Analysis*, 5(4):309–321, 1973.
- 24 R. K. Kincaid. Good solutions to discrete noxious location problems via meta-heuristics. *Annals of Operations Research*, 40(1):265–281, 1992.
- 25 M. J. Kubly. Programming models for facility dispersion: The p-dispersion and maximum dispersion problems. *Mathematical and Computer Modelling*, 10(10):792, 1988.
- 26 Mikael Zayenz Lagerkvist and Magnus Rattfeldt. Half-checking propagators. In *Proceedings of the 19th workshop on Constraint Modelling and Reformulation*, 2020.
- 27 A. Maier and H.W. Hamacher. Complexity results on planar multifacility location problems with forbidden regions. *Mathematical Methods of Operations Research*, 89:433–484, 2019.
- 28 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.
- 29 I. D. Moon and S. Chaudhry. An analysis of network location problems with distance constraints. *Management Science*, 30(3):290–307, 1984.
- 30 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.
- 31 C.S. Orloff. A theoretical model of net accessibility in public facility location. *Geographical Analysis*, 9:244–256, 1977.
- 32 M. G. Resende, R. Marti, M. Gallego, and A. Duarte. Grasp and path relinking for the max-min diversity problem. *Computers and Operations Research*, 37(3):498–508, 2010.
- 33 B. Saboonchi, P. Hansen, and S. Perron. MaxMinMin p-dispersion problem: A variable neighborhood search approach. *Computer & Operations Research*, 52:251–259, 2014.
- 34 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.
- 35 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.
- 36 D. R. Shier. A min-max theorem for p-center problems on a tree. *Transportation Science*, 11(3):243–252, 1977.
- 37 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.
- 38 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.
- 39 S.B. Welch and S. Salhi. The obnoxious p facility network location problem with facility interaction. *European Journal of Operational Research*, 102:302–319, 1997.
- 40 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.
- 41 29 C.F.R. §1910.157. Portable fire extinguishers. Title 29 Code of Federal Regulations, Part 157. 2021.