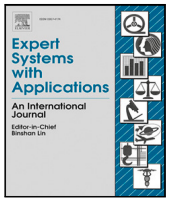
Expert Systems With Applications 255 (2024) 124484

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa



Multi-Neighborhood Simulated Annealing for the Capacitated Dispersion Problem

Roberto Maria Rosati *, Andrea Schaerf

DPIA, University of Udine, via delle Scienze 206, I-33100, Udine, Italy

ARTICLE INFO

Keywords:
Simulated Annealing
Metaheuristics
Multi-Neighborhood Search
Capacitated Dispersion
Diversity maximization

ABSTRACT

We propose a novel Multi-Neighborhood Simulated Annealing approach to address the Capacitated Dispersion Problem. It makes use of three neighborhoods, adapted from similar proposals from the literature. Our search method, properly engineered and tuned, is able to consistently improve the state-of-the-art methods on almost all instances from public benchmarks.

In addition, we highlight the limitations of the current datasets and we propose a new, more challenging one, obtained by sampling data from real maps and population density.

Finally, we propose two compact mathematical models that obtain good bounds on small/medium size instances as well as, with long runs, on large ones.

1. Introduction

In this work, we consider the *Capacitated Dispersion Problem*, which is a classic formulation within the family of *diversity* problems on graphs that underlies many real-world problems, such as facility location and network analysis.

The problem consists in finding a set of nodes in an undirected complete weighted graph that maximizes the minimum distance among selected nodes, subject to a capacity constraint on the selection.

This problem has been tackled in several recent works (Lu et al., 2023; Martíet al., 2021; Mladenović et al., 2022b; Peiró et al., 2021), mainly using metaheuristic approaches. The problem comes along with a public testbed, collected by Peiró et al. (2021), that has been used as a benchmark in all mentioned papers.

We propose a local search approach that uses a portfolio of neighborhoods and a Simulated Annealing metaheuristic to guide the search. The distinctive feature of our approach is the use of the union of different neighborhoods at each iteration. This idea blends with the random draw mechanism of Simulated Annealing, which does not require an exhaustive exploration of the neighborhood, as in other metaheuristics, which would be computationally expensive in this multi-neighborhood setting. The other key feature is the statistically-principled tuning of the rates of the atomic neighborhoods, instead of using predefined ones (typically uniform).

This approach has already proven quite successful on a variety of several discrete optimization problems, consistently obtaining results in line with the state-of-the-art (see Bellio et al., 2021; Ceschia et al., 2022; Rosati et al., 2022).

The neighborhoods that we use are gathered from the literature, though suitably modified and adapted for our context. Furthermore, we make use of a *lexicographic* objective function that facilitates navigation of the plateaus of the original objective function, and of a randomized greedy procedure adapted from Rosenkrantz et al. (2000) to generate the initial solution.

The experimental analysis shows that our multi-neighborhood method, properly engineered and tuned, is able to outperform the aforementioned previous techniques on the available benchmark.

In addition, we highlight severe limitations on the benchmark itself and we propose a novel, more diverse and challenging dataset that aims to overcome these limitations, and could become an additional benchmark for this problem for the future.

For this new dataset, properly split into test and validation instances, we provide the results of both our method and of the method by Lu et al. (2023) and by Mladenović et al. (2022b) obtained by running their code. The results confirm that our method is able to obtain better results on this dataset as well.

Finally, we adapt two models for the *p-dispersion* problem to the CDP, and we implement them using CPLEX. These models, inspired by the works by Kuby (1987) and Sayah and Irnich (2017), use less variables and less constraints than the existing models in the literature for the CDP, proposed by Peiró et al. (2021) and Martíet al. (2021). These models have been able to provide good bounds and optimal results for small and medium size instances. In addition, using very long runs, they also obtain good results on large instances.

* Corresponding author.
E-mail addresses: robertomaria.rosati@uniud.it (R.M. Rosati), andrea.schaerf@uniud.it (A. Schaerf).

https://doi.org/10.1016/j.eswa.2024.124484
Received 2 November 2023; Received in revised form 20 May 2024; Accepted 9 June 2024

The new dataset and all our best solutions are available at https://github.com/iolab-uniud/cdp for inspection and future comparisons.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 provides the problem definition and its mathematical model. Our search method is described in Section 4. In Section 5, we discuss the current and new datasets. Section 6 is dedicated to the experimental analysis. Finally, conclusions and future work are discussed in Section 7.

2. Related work

Diversity and dispersion problems have been studied intensively starting from the 70s (see, e.g., Shier, 1977). We refer to the recent survey by Martíet al. (2022) for an overview of the various formulations and the search methods used to solve them.

The specific version considered in this paper, called Capacitated Dispersion Problem (CDP), has been introduced by Rosenkrantz et al. (2000), who proposed a greedy algorithm and an approximation scheme for both CDP and other versions of the problem. In particular, they prove an approximation factor of 2 when the distances satisfy the triangular inequality. This result is of theoretical interest, but with limited practical use.

The CDP has been recently addressed by using metaheuristic approaches. Peiró et al. (2021) propose a technique based on Greedy Randomized Adaptive Search Procedure (GRASP), Variable Neighborhood Search, and Strategic Oscillation, as well as a mathematical model. They adapted the datasets from other problems, then they compared their method with the CPLEX implementation of the mathematical model, and with the greedy algorithm of Rosenkrantz et al. (2000).

Subsequently, Martíet al. (2021) proposed a metaheuristic method based on Scatter Search and also developed a new mathematical model. They compared their search method and their model with the ones by Peiró et al. (2021).

Lu et al. (2023) apply a solution-based Tabu Search using three distinct neighborhoods: insert, remove and swap. In order to speed up the check of equality of the current solution with tabu ones, they employ hash functions that identify eligible candidate solutions. They improved over all previous results, suggesting that local search methods are suitable for the CDP.

The current state-of-the-art results for the CDP are provided by Mladenović et al. (2022b), that developed various versions of Variable Neighborhood Search, namely Basic VNS, General VNS and General Skewed VNS. In addition to the neighborhoods used by Lu et al. (2023), they also use the neighborhoods 2-out-1-in and 1-out-2-in, that insert or remove two nodes at a time.

Given the substantial gap existing between the results from Lu et al. (2023), Mladenović et al. (2022b) and the previous ones, we only consider the former two for comparison in this paper.

Even though local search methods have been successful in solving the CDP, we are not aware, to the best of our knowledge, of any work that uses a stochastic sampling approach, as all the above-mentioned methods employ exhaustive exploration of the neighborhood. We believe, however, that it is computationally expensive in a problem like the CDP, that presents wide plateaus characterized by many equivalent solutions, and might be a limiting factor in the use of larger neighborhoods.

3. Problem definition

We are given an undirected graph $G = (V, E)$, where V is a set of nodes, and E is a set of edges between nodes in V. Each node i is assigned a capacity c_i and each edge is assigned a value d_{ij} representing the *distance* between nodes i and j. We are also given a single value B representing the total minimum capacity requested for the solution. All values are real.

The graph is assumed complete and all distances are non-negative, but they are not assumed Euclidean, and they do not need to satisfy the triangular inequality.

The problem consists in selecting a set of nodes $S \subseteq V$ such that the sum of the capacities of the nodes in S is at least equal to B and the minimum distance between nodes in S is maximized.

Furthermore, we propose two mathematical models, both adapted to the CDP from works in the literature on the p-dispersion problem. The first one was proposed by Kuby (1987) and later on also studied by Erkut and Neuman (1991), who considered different objectives. The second one is adapted from Sayah and Irnich (2017), who proposed a compact formulation for the p-dispersion problem. The adaptation of a model to the CDP is straightforward, as the only difference is the replacement of the p-dispersion constraint with the capacity constraint.

Both models are significantly more compact than the one by Peiró et al. (2021), that uses an additional variable y_{ij} for each pair (i, j), that takes value 1 if both x_i and x_j have value 1. They also use less variables and less constraints than the one by Martíet al. (2021), which however is based on a completely different paradigm.

3.1. Model 1: adapted from Kuby (1987)

The first mathematical model is based on binary variables $x_i \in \{0, 1\}$, with $i \in V$, that take the value 1 if the node i is selected in the solution, 0 otherwise. In order to express the objective to maximize the minimum distance between the selected nodes in the linear model, we introduce a new, real-valued variable $\bar{d} \in \mathbb{R}$ that represents the minimum distance between the selected nodes. The objective function is then $\max \bar{d}$ and the model is the following one.

$$\max \quad \bar{d} \tag{1}$$

$$\sum_{i \in V} c_i x_i \geq B \tag{2}$$

$$\bar{d} \leq d_{ij} + M(1 - x_i) + M(1 - x_j) \qquad \forall i, j, i < j \in V, i \neq j \tag{3}$$

$$x_i \in \{0, 1\} \qquad \forall i \in V \tag{4}$$

$$\bar{d} \geq 0 \tag{5}$$

The first constraint (2) simply imposes that the sum of the capacities c_i of the selected nodes is at least the minimum capacity B. The second set of constraints (3) imposes that, for every pair of nodes (i, j), the minimum distance $\bar{d}$ is at most equal to the distance d_{ij}, but only if both nodes are selected.

To express this condition, we use the BigM technique, with the constant M that takes a suitably high value, which is multiplied by $(1 - x_i)$ and $(1 - x_j)$. In this expression, if one of the two nodes is not selected, that is, either $(1 - x_i)$ or $(1 - x_j)$ is 1, then the constraint says that $\bar{d}$ is free to take any value below $M + d_{ij}$, or below $2M + d_{ij}$ if both nodes are unselected. If, on the other hand, x_i and x_j are selected, both terms that multiply M take value 0 and $\bar{d}$ is constrained by the distance between the two nodes d_{ij}. Given that the constraint is repeated for all pairs of nodes (i, j), the distance $\bar{d}$ is constrained by the minimum distance between the selected nodes.

3.2. Model 2: adapted from Sayah and Irnich (2017)

Let $D^0 < D^1 < \cdots < D^{k_{max}}$ be the set of all distinct distances d_{ij}, and let $E(D^k)$ be the set of edges (i, j) with distance $d_{ij} \leq D^k$. Based on this definition, the set of edges with distance $d_{ij} = D^k$ is identified as $E(D^k) \setminus E(D^{k-1})$.

Our second mathematical model is based on two binary variables $x_i \in \{0, 1\}$ with $i \in V$ and $z_k \in \{0, 1\}$ with $k \in K = \{1, 2, \ldots, k_{max}\}$. The first one, similarly to the first model, is the location selection, while

the second one takes value 1 when the location selection is such that the minimum distance is at least D^k. The binary model is the following one.

$$\max \quad D^0 + \sum_{k \in K} (D^k - D^{k-1}) z_k \tag{6}$$

$$\sum_{i \in V} c_i x_i \geq B \tag{7}$$

$$z_k \leq z_{k-1} \qquad \forall k \in K, k > 1 \tag{8}$$

$$x_i + x_j + z_k \leq 2 \qquad \forall k \in K, (i,j) \in E(D^k) \setminus E(D^k - 1) \tag{9}$$

$$x_i \in \{0,1\} \qquad \forall i \in V \tag{10}$$

$$z_k \in \{0,1\} \qquad \forall k \in K \tag{11}$$

Constraint (7) ensures that the capacity of the chosen nodes is at least B. It is the same as in the first model. Constraint (8) imposes that the z_k variables are non-increasing in k. Constraint (9) ensures that, if the minimum distance is D^k, then at most one node is selected from any pair (i, j) at distance $d_{ij} < D^k$.

4. Solution method

We propose a Multi-Neighborhood local search approach for the CDP. In the following sections, we will outline the fundamental components of the local search paradigm, which include the *search space*, the *initial solution strategy*, the *neighborhood relations*, the *cost function* and the *metaheuristic* that drives the search.

4.1. Search space

Our search space is characterized by two disjoint sets. The first is S, that coincides with the solution, that we name the *selection list*, and that includes the nodes that are currently selected. We also store the complementary set $C = V \setminus S$, named *candidate list* because it is the set of nodes that are candidates for insertion in the current solution.

We assume that $|S| \geq 2$, that is, a solution always contains at least two nodes and one edge. While in theory a solution with only one node that alone satisfies the capacity requirement (or even a empty solution, if $B = 0$) might exist, in the current datasets this never happens, therefore we can safely prevent the case of having a solution with one single node, which would have an undefined value of the objective function.

With regard to the capacity constraint, we do not impose its satisfaction and thus we let the search explore both the feasible and infeasible regions. As explained in Section 4.5, capacity violations are included in the cost function with a high value, in order to direct the search toward the feasible region.

To speed up the computation and evaluation of moves, we define several redundant data structures that complement the two sets S and C. For every node, regardless of whether or not it is in the selection list, we store the value of the minimum distance edge that connects the node to the selection list into a variable $d_S(v)$ and we compute it as $d_S(v) = \min_{u \in S} d_{uv}$. We also keep the counter $|d_S(v)|$, telling how many edges with such distance there are, for every node. Finally, recalling that the objective function of the problem is defined as $f(S) := \min_{u,v \in S} d_{uv}$, we define the set $E_R \subset E$ as the set of edges between nodes in S with a distance exactly equal to $f(S)$. We call these edges the *bottleneck* edges, given that they determine the value of the objective function. Rather than keeping track of the full set E_R, we only store and update the value $|E_R|$.

4.2. Initial solution strategy

We generate the initial solution for the local search through a randomized greedy algorithm. Instead of designing a completely novel procedure, we adapted the greedy from Rosenkrantz et al. (2000). The original algorithm is fully deterministic, so we have modified it[1] in order to perform random tie-breaks in the initial sorting of nodes by non-increasing capacity. Indeed, with many nodes having the same capacity, random tie-breaks can lead to different solutions.

An alternative possibility to the greedy start is to employ a completely random approach. We explore this possibility in Section 6.3.

4.3. Neighborhood relations

We propose a Multi-Neighborhood approach that makes use of three neighborhoods:

- Insert: the move Insert$\langle v \rangle$ inserts a node v from the candidate list into the selection list S.
  Preconditions: $v \in C$.
- Remove: the move Remove$\langle v \rangle$ removes a node v from the selection list S and returns it to the candidate list.
  Preconditions: $v \in S$, $|S| > 2$.
- Swap: the move Swap$\langle v, u \rangle$ inserts a node v from the candidate list into the selection list S, and removes a node u from the selection list, returning it to the candidate list.
  Preconditions: $v \in C$, $u \in S$.

We also consider three restricted versions of the Swap neighborhood, that we call SwapR_+, SwapR_- and $\text{SwapR}_\pm$. To describe them, we introduce two additional sets, the restricted selection list S_R and the restricted candidate list C_R. The restricted selection list is defined as $S_R := \{v \in S | d_S(v) = f(S)\}$. Set S_R contains all the nodes in the solution with an edge to another node in the solution with distance equal to the current minimum distance among selected nodes, that is the value of the objective function. We also denote the nodes in S_R as the bottleneck nodes, because they bound the value of the objective function $f(S)$. The restricted candidate list is defined as $C_R := \{v \in C | d_S(v) > f(S)\}$, that is, all the nodes in the candidate sets with minimum distance to the nodes in the selection list greater than the current objective function value.

SwapR_+ restricts the Swap neighborhood with regards to the node inserted in the solution, which is chosen within the restricted candidate list C_R, but allows removal from the whole selection list S. The neighborhood SwapR_- restricts the Swap neighborhood with regards to the node that is removed from the solution list, that is chosen only among the nodes in S_R, but allows any insertion from the whole candidate list C. Finally, $\text{SwapR}_\pm$ restricts both the insertion and the removal to the restricted lists.

We point out that both Lu et al. (2023) and Mladenović et al. (2022b) use restricted versions of their Swap neighborhood, but with relevant differences compared to us. First of all, we employ all four neighborhood variants Swap, SwapR_+, SwapR_- and $\text{SwapR}_\pm$, while Lu et al. (2023) only employ $\text{SwapR}_\pm$ and Mladenović et al. (2022b) only employ SwapR_-. Additionally, our definition of the restricted candidate set C_R is different from the one provided by Lu et al. (2023), as we employ a larger set, that embraces all nodes $v \in C$ with $\min_{u \in S} d_{uv} > \min_{u \in S} d_S(u) = f(S)$. The restricted selection list, on the other hand, is identical in all the algorithms. Thanks to these differences, our

[1] The original pseudo-code (Heuristic: T1:Max-Dist/Cap) contains a typo: vector $D[1 \dots t]$ (step 2) should be instead sorted increasingly, and not decreasingly, otherwise the binary search at step 3 (find the ith element of the array such that GreedyTry$(D[i], B)$ returns success and GreedyTry$(D[i+1], B)$ returns failure) cannot work. The same can be obtained by replacing $i+1$ with $i-1$ in step 3.

neighborhood variants are larger than the Swap that Lu et al. (2023) and Mladenović et al. (2022b) have proposed.

The main reason why we can afford to use larger neighborhoods is that the metaheuristic that guides the search in our method is Simulated Annealing, which employs stochastic move selection, and therefore the computational cost of the evaluation of a move does not depend on the size of the neighborhood. Specifically, in our method, the cost of both evaluating and executing moves is either constant or linear with respect to the size of the solution, except for some rare situations in which the cost is quadratic. In particular, this only happens when there is exactly one bottleneck edge, and the move removes one of its end points.

Conversely, Lu et al. (2023) and Mladenović et al. (2022b) employ Tabu Search and VNS, which perform a complete exploration of the neighborhood at every move selection and therefore might foresee its performances reduced by the use of larger neighborhoods.

4.4. Move selection

We now describe how we select a random move from our Multi-Neighborhood Insert ∪ Remove ∪ Swap. We define three real-valued parameters σ_I, σ_R, and σ_S, with $\sigma_I + \sigma_R + \sigma_S = 1$. They represent the probability of selecting the Insert, Remove, and Swap neighborhoods, respectively.

Given that, however, we have four versions of the Swap neighborhood, we define two internal biases: $b_+ \in [0, 1]$ and $b_- \in [0, 1]$. If the Swap neighborhood is selected, then with probability b_+ we use the restricted candidate list C_R to select the candidate node for insertion, and with probability b_- we use the restricted selection list S_R to choose the node that is removed. Therefore, we can derive that the probability of performing a regular Swap is $\sigma_S(1 - b_+)(1 - b_-)$, the probability of selecting SwapR_+ is $\sigma_S b_+(1 - b_-)$, the probability of selecting SwapR_- is $\sigma_S(1 - b_+)b_-$ and, finally, $\text{SwapR}_\pm$ is selected with probability $\sigma_S b_+ b_-$.

Now that every neighborhood is assigned a probability, we can perform a random selection of a move in two steps (three steps for Swap). First, we select the neighborhood, with a biased random selection that depends on the fixed probabilities σ_I, σ_R, and σ_S. If Swap is chosen, with another biased random selection we choose whether to use the restricted or the full candidate and solution lists, with probabilities b_+ and b_-, respectively. Finally, we draw the specific move, with a uniform selection inside the chosen neighborhood.

4.5. Cost function

One issue with the max-min objective function is that very large plateaus might be encountered. In fact, its value is determined by the edges in E_R. If $|E_R| = 1$, then the value is given by a single pair of nodes and removing one of the two nodes (provided that the move does not violate the capacity constraint) is sufficient to improve the objective function.

When $|E_R| > 1$, however, unless there is a node $v \in S_R$ with $|d_S(v)| = |E_R|$, all three neighborhoods are on a plateau with regard to $f(S)$, given that all possible moves are sideways or worsening. This situation makes the search blind about what are the best moves to perform, which might be stuck roaming for a long time on the plateau. However, we know that to improve the objective function we first have to reduce the number of edges in E_R, until we have one single edge that is binding $f(S)$, that we can finally remove and jump to a new objective function value.

Therefore, in order to deal with this situation, we introduce an auxiliary cost component that guides the search on the plateaus toward solutions with smaller $|E_R|$. Given that this is less important than the actual objective, we define a lexicographic objective function as follows: given two solutions S_1 and S_2, we say that S_1 is lexicographically better than S_2 if

$$f(S_1) > f(S_2) \vee \big(f(S_1) = f(S_2) \wedge |E_{R_1}| < |E_{R_2}|\big) \tag{12}$$

In order to use this lexicographic function inside our metaheuristic, we linearize it as follows:

$$f^{lex}(S) = w_{of} f(S) - |E_R| \tag{13}$$

with w_{of} a constant weight, assigned with a suitably high value to ensure that the second component is only relevant if two solutions have the same objective function value. The new term is subtracted and not added because the objective of the problem is to maximize the objective function, while we want to minimize $|E_R|$.

Our lexicographic objective function provides a trajectory for executing a successful sequence of moves on plateaus, so that the need for more complex neighborhoods, such as the 2-out-1-in and 1-out-2-in neighborhoods proposed by Mladenović et al. (2022b), is counterbalanced.

Finally, as we mentioned in Section 4.1, we also allow the exploration of the infeasible region of the search space. Therefore, we also add a penalty term for capacity violations, and we weight it with a constant w_c. Again, given that the problem is a maximization one, we subtract the penalty term. We obtain the following expression to determine the cost function F used in the metaheuristic:

$$F(S) = w_c \min\Big\{0, \sum_{v \in S} c_v - B\Big\} + w_{of} f(S) - |E(S)| \tag{14}$$

The redundant data structures that we introduced in Section 4.1 are used to speed up the evaluation of the differential cost ΔF between two solutions S_1 and S_2, without computing $F(S_1)$ and $F(S_2)$ from scratch.

4.6. Simulated annealing

The metaheuristic that guides the search is Simulated Annealing (SA). We use the classic version, as originally proposed by Kirkpatrick et al. (1983), with the inclusion of a cut-off mechanism to speed up the early stages of the search. The pseudo-code is shown in Algorithm 1, where n is the number of atomic neighborhoods.

The key features of SA are the random nature of the move selection at each iteration and the acceptance criterion that depends on a *temperature* T, which decreases from an initial value T_0 to a final one T_f. The move selection, that in this context is performed in two stages as explained in Section 4.4, is illustrated in lines 9 and 10. According to the *Metropolis* acceptance criterion, improving and sideways moves are always accepted as new states (lines 12-18), whereas worsening ones are accepted with probability $e^{-\Delta F/T}$. The temperature is decreased according to the *geometric cooling* scheme in every N_s sampled iterations (line 26), where α (with $0 < \alpha < 1$) is the *cooling rate*.

The aforementioned *cut-off* mechanism decreases the temperature when a fixed number of moves $N_a < N_s$ has been accepted. The idea behind the cut-off is that the temperature is too high if too many solutions are accepted, and we should decrease it more aggressively. If the cut-off takes place, the number of sampled moves that are not used in the current iteration is redistributed among the future temperatures, increasing N_s accordingly. Instead of using as parameter N_a, we define a parameter $\rho \in (0, 1]$, used to determine the number of initial iterations per temperature N_s that have to be accepted to apply the cut-off, such that $N_a = \rho N_s$.

To determine the length of the SA run, we use a fixed number of iterations $\mathcal{I}$, which corresponds to the number of moves that are drawn and evaluated, regardless of whether they are accepted or not. From $\mathcal{I}$, we compute the parameter N_s using the formula:

$$N_s = \mathcal{I} \Big/ \log_\alpha\left(\frac{T_f}{T_0}\right) \tag{15}$$

This simplifies the determination of the duration of the algorithm execution and makes it independent from the parameters.

Algorithm 1 Multi-Neighborhood Simulated Annealing With Cutoff

procedure MNSA WITH CUTOFF(SearchSpace Σ, Multi-Neighborhood $\mathcal{N}_\cup = \bigcup_{k=1}^{n} \mathcal{N}_k$, CostFunction F, Parameters T_0, T_f, α, N_s, N_a, $\{\sigma_1 \dots \sigma_n\}$)
  $T \leftarrow T_0$
  $S \leftarrow$ GreedyState(Σ)
  $S_{best} \leftarrow S$
  while $T \geq T_f$ **do**
    $n_s \leftarrow 0$
    $n_a \leftarrow 0$
    while $n_s < N_s$ and $n_a < N_a$ **do**
      $\mathcal{N}_k \leftarrow$ RandomNH($\mathcal{N}_\cup, \{\sigma_1 \dots \sigma_n\}$)
      $m \leftarrow$ RandomMove($S, \mathcal{N}_k$)
      $\Delta F \leftarrow F(S \oplus m) - F(S)$
      if $\Delta F \geq 0$ **then**
        $S \leftarrow S \oplus m$
        $n_a \leftarrow n_a + 1$
        if $F(S) > F(S_{best})$ **then**
          $S_{best} \leftarrow S$
        end if
      else
        if RandomReal$(0, 1) < e^{-\Delta F/T}$ **then**
          $S \leftarrow S \oplus m$
          $n_a \leftarrow n_a + 1$
        end if
      end if
      $n_s \leftarrow n_s + 1$
    end while
    $T \leftarrow T \cdot \alpha$
    $N_s \leftarrow N_s + \left(N_s - n_s\right) / \log_\alpha \left(T_f / T\right)$
  end while
  return S_{best}
end procedure

5. Datasets and generators

Recent papers in the literature on CDP have used four datasets, called GKD-b, GKD-c, SOM-a, and MDG-b, composed by 20, 10, 10, and 10 instances, respectively. The number of nodes is fixed, equal to 50/150 (10 each), 500, 50, 500, for all instances of the four datasets.

These datasets[2] were initially proposed in various works and then collected by Martíet al. (2010). They were originally defined for problems without the capacity constraints, therefore they have been completed by Peiró et al. (2021) by adding both the node capacities and the capacity threshold. In this process, each dataset has been duplicated into two groups by considering a threshold equal to either 0.2 or 0.3 of the sum of the capacities of the nodes. Therefore, we eventually have eight groups of instances named by adding the suffix 2 or 3 to the original name: GKD-b2, GKD-b3, GKD-c2, ...

All datasets are artificial. In detail, the datasets GKD-b and GKD-c are created by sampling points on a square and getting the Euclidean distance among all pairs of nodes. Conversely, for SOM-a and MDG-b distances are integers uniformly sampled in the range [0,9] and [0,1000], respectively.

For all instances of datasets GKD-b, GKD-c, and SOM-a, the MILP model by Martíet al. (2021) finds the optimal solutions in a few seconds. In addition, all metaheuristic methods (including ours) find the optimal value quite consistently. The comparison on these datasets could therefore be based only on the number of times that the optimal value is obtained and the time to reach it.

[2] Available at https://www.uv.es/rmarti/paper/mdp.html.

For the above reasons, these instances are easy and we believe that they do not represent a good benchmark anymore. We decided to discard them and to focus on the 20 instances of dataset MDG-b (10 of the MDG-b2 group and 10 of the MDG-b3 group). We use the others solely for validating the implementation of the mathematical models.

The dataset MDG-b is instead quite challenging, and therefore it is a good benchmark for the comparison of optimization techniques. Nonetheless, in our opinion, it has some severe limitations:

- It is too small. Indeed, 20 instances are not enough for an extensive comparison.
- It is too homogeneous, as all instances have the same size and they are duplicated with only the capacity modified.
- Distance values are too random. In fact, they do not even satisfy the triangular inequality.
- The presence of various zeros in the distance matrix is also rather strange, given that they should represent physical distances.

For these reasons, we believe that it is necessary to introduce a new dataset so as to enrich and diversify the current benchmark. Therefore, we design a generator, based on real geographical data, that aims at creating instances that are challenging, realistic, and diverse. Our generator, written in R language, employs the JRC-GEOSTAT 2018 population grid, that covers 38 European countries, including all the European Union member States plus a few neighboring countries. We use the shapefile under the coordinate system EPSG:3035 with a resolution of 1 km^2, which means that it is divided in squared cells with side 1 km. Every cell contains the coordinates and the data of the population living in the cell, updated to the year 2018, which is the most recent validated version of the grid.

The first step consists in cutting a portion of the grid based on a center (x, y) and a radius r, which implies an area of πr^2, minus the potential portion that lies on the sea. To generate an instance with size $|V|$, on this area, we select $|V|$ cells at random, through a biased random selection, with the probability proportional to the population of the cell.

Said A the set of the cells in the area, and p_n the population of the nth cell, the probability of selecting cell n is

$$P(n) = \frac{p_n}{\sum_{m \in A} p_m} \tag{16}$$

After every cell selection, we sample, with a uniform random distribution, the exact coordinates of a point (x_i, y_i) within the $1km^2$ cell boundaries. Therefore, every node in the instance corresponds to a precise point in the map.

After we have sampled the $|V|$ points in the space, the procedure computes the distance matrix $|V| \times |V|$. We set as distance the time needed to cover the path between points by car, expressed in minutes. For this, we employ the Open Source Routing Machine, which returns the most time-effective route between two nodes. We point out that a real road distance matrix is not symmetric because of factors such as one-way directions or steepness. However, given that the problem requires a symmetric matrix, we only compute d_{ij} for $i < j$, and then we assign $d_{ji} = d_{ij}$.

In our generation procedure, the cell population is used not only to bias the sampling of the points, but it is also assigned as the capacity of the node: if the node i was extracted from cell n, then $c_i = p_n$. The minimum capacity requirement is computed as a random value $B \in (B_{min}, B_{max}]$. Hereby, we set $B_{min} = max_{i \in V}(c_i)$ and $B_{max} = \min\{0.3 \sum_{i \in V} c_i, 0.9 \sum_{m \in A} p_m\}$. These values ensure that the capacity requirement B is always less than 30% of the total capacity, or 90% of the population if this is less than the total capacity. The radius r is chosen at random in the interval $[r_{min}, r_{max}]$, with $r_{min} = 5$ km and $r_{max} = 90$ km. Finally, the possible centers (x, y) are taken from a pool of various cities and rural areas across Europe.

As a possible real-world interpretation of this construction, consider that we want to decide where to place drugstores. The capacity of

Table 1
Features of the validation instances.

#	GeoData				Size			Distances					
	city	r	A	pop	$\|V\|$	B(%)	B	avg	min	1st	med	3rd	max
1	Bilbao	28	2197	1 131 415	627	0.11	918 402	22.2	1	13	20	30	74
2	Bremen	51	8161	1 735 993	1335	0.20	693 367	41.7	1	27	41	54	136
3	Cagliari	50	6318	678 220	565	0.19	421 770	39.7	1	19	38	57	159
4	Catania	51	5468	1 390 504	714	0.24	870 107	40.4	1	20	39	58	149
5	Cuneo-Fossano	34	3613	549 625	479	0.29	196 562	42.7	1	30	43	56	104
6	Exeter	77	12 464	1 912 208	1259	0.18	552 502	73.4	1	46	69	96	229
7	Foggia	36	4021	409 910	701	0.04	177 467	36.4	1	24	37	49	118
8	Fontenay-Niort	86	21 804	1 822 107	974	0.16	180 151	85.2	1	59	87	111	215
9	Gottingen	29	2617	415 321	1206	0.08	164 395	29.9	1	20	30	39	81
10	Gottingen	88	24 309	3 803 418	1358	0.05	129 498	90.0	1	60	90	118	211
11	Jena	8	193	114 691	757	0.01	40 680	11.6	1	8	11	15	45
12	Lublin	44	6073	869 345	1042	0.03	97 965	38.9	1	22	39	53	125
13	Milan	32	3205	4 992 261	568	0.24	909 420	31.1	1	21	30	40	84
14	Milan	53	8797	7 612 631	692	0.27	906 640	44.6	1	30	43	57	131
15	Munich	7	145	988 689	1343	0.05	697 453	12.6	1	9	12	16	33
16	Munich	90	25 433	6 024 374	798	0.16	463 017	60.8	1	40	59	79	179
17	Naples	57	7158	5 222 684	1346	0.23	2 124 227	45.8	1	25	40	58	292
18	Nowe Miasto	89	24 833	2 104 112	1487	0.03	122 757	104.8	1	68	104	141	247
19	Palermo	88	12 199	2 159 507	1186	0.06	369 869	81.7	1	40	83	117	222
20	Pamplona	23	1649	379 770	512	0.05	251 664	11.1	1	7	10	13	76
21	Rome	15	697	2 607 377	700	0.19	1 344 222	19.9	1	14	20	26	49
22	Sofia	47	6917	1 608 481	1034	0.18	1 237 857	28.7	1	13	21	39	200
23	Split	41	4412	390 868	614	0.04	134 182	40.0	1	11	28	48	264
24	Suzzara	68	14 493	3 822 210	1447	0.27	947 949	71.9	1	49	73	95	170
25	Tampere	19	1125	344 110	664	0.16	278 974	18.7	1	13	18	24	68
26	Ulm	54	9141	1 903 978	1241	0.19	411 900	53.7	1	37	54	70	133
27	Ulm	8	193	191 864	845	0.04	124 649	10.9	1	8	11	14	31
28	Vienna	53	8797	3 083 512	1469	0.10	1 207 533	32.1	1	17	29	45	121
29	Wroclav	48	7209	1 302 256	1256	0.02	107 873	39.5	1	22	39	55	129
30	Zagreb	53	8797	1 528 544	980	0.15	600 667	39.1	1	20	38	55	141

the node, which is the population living in the 1 km^2 where it is situated, is an approximation of the number of people placed at walking distance within the facility, which we want to satisfy by at least B. On the other hand, we want to avoid that two drugstores are placed too close, therefore we desire to maximize the minimum distance between stores. The same reasoning can be applied to other contexts, such as the placement of schools or public offices, that is convenient to build in dense areas to reduce commuting times and the use of private vehicles, while avoiding placing them too close to each other to ensure that their zones of influence do not overlap.

Using our generator, we have created a dataset composed of 200 instances with size $|V|$ between 300 and 1500, that we call GIS (given that they come from geographical data). We use 30 instances (chosen at random, with no specific choice criterion) as validation instances and the rest as training ones. We also tested all of them with our models presented in Section 3. We verified that no GIS instance could be solved to optimality within the one hour time limit, for which they appear to be rather challenging. All GIS instances are available at https://github.com/iolab-uniud/cdp, divided into training and validation ones.

Table 1 shows the features of the validation instances. The table reports, from left to right, the number of the instance, the name of the city, the radius r in km, the size A in km^2, the population, the number of vertices in the instance ($|V|$), the capacity requirement B (both percentage of total capacity available and absolute value), and finally some information on the distribution of the distances (average, minimum, first quartile, median, third quartile, and maximum). The name of the city simply indicates the city or town chosen as center of the area. Sometimes, if the radius is large, other cities might be included.

6. Experimental analysis

In this section we present our experimental methodology and our results, and we provide graphics and insights on the behavior of our algorithm as well as on the characteristics of the new GIS instances in relation to the algorithm performances. Our code is implemented in C++ and compiled using g++ (v. 11.4.0) in `-O3` mode. The experiments were run on a AMD Ryzen Threadripper PRO 3975WX 32-Cores (3.50 GHz) with Ubuntu Linux 22.04.3. The MILP models are also implemented in C++, through the CONCERT interface for CPLEX (v. 22.1).

In order to ensure a fair comparison of the performance of the models, the time limit is set to 2880 s, corresponding approximately to one hour on the CPU used by Martíet al. (2021). The Simulated Annealing does not run on a fixed runtime but on a fixed number of iterations. We fixed to 50 millions the number of iterations in the tuning phase and, in the validation one, to 100 millions for MDG-b instances and to 300 millions for GIS instances. We highlight that, while it does not provide certainty on the running time, using a fixed number of iterations always allow to replicate a specific single experiment using the same random seed, which is not possible using a time cut-off, not even on the same machine. Except when explicitly stated, one single core was dedicated to each experiment.

6.1. Tuning

We tuned our solver using *irace*, which is a tool for automatic algorithm configuration based on iterated racing (López-Ibáñez et al., 2016). Given that the datasets MDG-b and GIS have different features, we performed two separate tuning procedures.

Specifically, the tuning was only done in the training instances for the GIS dataset, whereas it has been done on all instances for MDG-b, because there are no dedicated instances available for training purposes.

The tuning was carried out in two stages. In the first stage, we tuned the parameters of Simulated Annealing and the weight w_{of}. In the second stage, we tuned the neighborhood rates and biases. The value w_c is preventively fixed to 10000. Regarding the neighborhood rates, we only tune σ_I and σ_R, because σ_S is computed as $1-\sigma_I-\sigma_R$. Table 2 shows the results of the tuning procedure, separately for MDG-b and GIS dataset.

Table 2
Parameter tuning for the Multi-Neighborhood Simulated Annealing.

Name	Description	Initial value		
		Range	MDG-b	GIS
T_0	Start temperature	[2, 2000]	194.49	123.58
T_f	Final temperature	[0.001,1.000]	0.3758	0.0046
α	Cooling rate	[0.980,0.995]	0.988	0.981
ρ	Cut-off	[0.02, 0.30]	0.160	0.113
w_s	Weight of $f(S)$ in cost function F	[1,400]	11	234
σ_I	Rate of Insert neighborhood	[0.00,0.25]	0.131	0.177
σ_R	Rate of Remove neighborhood	[0.00,0.25]	0.129	0.012
b_+	Bias toward C_R	[0.00,1.00]	0.449	0.576
b_-	Bias toward S_R	[0.00,1.00]	0.359	0.954

Table 3
Results of the MILP models aggregated by dataset.

B	Dataset	$\|V\|$	Martíet al. (2021)			Peiró et al. (2021)		
			LB	UB	time	LB	UB	time
	GKD-b	50	112.3	112.3	0.1	112.3	112.3	2.0
	GKD-b	150	118.7	118.7	0.6	118.7	118.7	669.9
0.2	GKD-c	500	9.4	9.4	5.7	6.8	20.5	3600.2
	SOM-a	50	4.1	4.1	0.0	4.1	4.1	1.3
	MDG-b	500	0.0	125.0	3643.5	11.5	973.8	3600.2
	GKD-b	50	97.8	97.8	0.0	97.8	97.8	2.5
	GKD-b	150	108.1	108.1	0.3	108.1	108.2	1519.5
0.3	GKD-c	500	8.4	8.4	1.4	4.9	22.8	3600.1
	SOM-a	50	2.1	2.1	0.0	2.1	2.1	1.0
	MDG-b	500	3.1	60.2	3650.9	0.9	992.0	3600.1
B	Dataset	$\|V\|$	Model 1			Model 2		
			LB	UB	time	LB	UB	time
	GKD-b	50	112.33	112.33	3.7	112.33	112.33	18.1
	GKD-b	150	118.73	118.73	203.7	118.73	127.93	1908
0.2	GKD-c	500	7.73	22.92	3600	8.98	22.92	3600
	SOM-a	50	4.10	4.10	5.1	4.10	4.10	2.6
	MDG-b	500	33.28	1000.00	3600	29.29	1000.00	3600
	GKD-b	50	97.79	97.79	8.6	97.79	97.79	24.9
	GKD-b	150	108.11	108.11	287.6	108.11	116.75	2930
0.3	GKD-c	500	6.09	22.92	3600	7.00	22.92	3600
	SOM-a	50	2.10	2.10	8.3	2.10	2.10	2.2
	MDG-b	500	13.24	1000.00	3600	2.07	1000.00	3600

6.2. Results

Table 3 shows the results obtained by our models. We report the average solution found, the upper bound, and the running time,[3] which might be less than the time limit in case the optimal solution is found. We observe that all models find the optimal solutions on all instances with up to 150 nodes, even though our Model 2 fails at proving optimality on all instances. The model by Martíet al. (2021) solves to optimality in a few seconds the instances from the class GKD-c. On MDG-B instances, our Model 1 shows better performances than the other models, but, differently from Martíet al. (2021), it is not capable of finding good bounds. Instead, the performance of our Model 2 is only good when $B = 0.2$.

In Table 4 we report the detailed results obtained by both our models on MDG-b instances. Furthermore, we show the results obtained by the models running on 16 cores, in parallel, for 12 h, for a total of 196 CPU hours per run. The reason for this test was to understand if MDG-b instances are out of reach or if the model could find optimal solutions within a longer time limit. The results tell that both models obtain good results with longer runs, but never proven optimal. In some cases, which are marked in boldface, they get solutions that are as good as or better than the best metaheuristic solution. In particular, on longer runs Model 2 is the one which shows the best performances, especially on the instances with $B = 0.2$, coherently with what we have observed on short runs. Only on instances MDG-b_01_n500_b02_m50 and MDG-b_10_n500_b02_m50, however, both models are capable of finding solutions that are as good as the metaheuristic ones. In all cases, the upper bounds are very far from the lower bounds.

Even though the very long runs are performed with no practical applicability in mind, they are useful to confirm that MDG-b is the only challenging dataset and that we are still far from knowing the optimal solutions.

For what concerns metaheuristics, Table 5 shows the results obtained by 40 repetitions per instance of our solver on the MDG instances. We compare our results with those from Lu et al. (2023) and Mladenović et al. (2022b), obtained respectively on 40 and 20 repetitions per instance. We do not report other results from the literature, because they lag quite behind.

Following the line set by both Lu et al. (2023) and Mladenović et al. (2022b), beside the average we also report the best and worst solution found for each instance in the batch of 40 runs. Furthermore, we report the average running time, because our solution method works on a fixed number of iterations, which cause the running time to change from run to run. Lu et al. (2023) and Mladenović et al. (2022b) used a fixed time limit of 300 s and 60 s, respectively, so we do not report it in the table. Lu et al. (2023) also solve the instances on a shorter time limit of 10 s, but they get worse results, that we do not report.

The outcome is that our solver finds the best average solution on 19 out of 20 instances, and only on the instance MDG-b_10_n500_b03_m50 it performs worse than Mladenović et al. (2022b), by just 0.01. For all the other instances we improve the previous results, with an average gap that goes from as little as 0.11 on the instance MDG-b_08_n500_b02_m50 to as much as 3.01 on the instance MDG-b_08_n500_b03_m50. Regarding the best solution, we always improve or match the one found by the others, and regarding the worst solution found on the batch of forty runs, we are below Mladenović et al. (2022b) on one instance. Overall, we can say that our solver performs better than the previous ones from the literature, and that Mladenović et al. (2022b) is the closest competitor.

In Table 6 we report the results obtained on the GIS dataset by our solution method and by the solvers by Mladenović et al. (2022b) and Lu et al. (2023) on 40 runs per instance. Additionally, we also report the results obtained by our MILP models and by our adaptation of the greedy algorithm by Rosenkrantz et al. (2000), because its performances are surprisingly good on certain GIS instances.

Adopting the baseline of $300s$ used by Lu et al. (2023), we run both the code by Lu et al. (2023) and Mladenović et al. (2022b) within a time limit of 300 s, while the running time of our method, that uses a fixed number of iterations, spans from 90 to 347 s. The models had a time limit of one hour. The running time of the greedy is not reported because it is always less than 1 s. For the solver by Mladenović et al. (2022b), we tested the possible values of $\{0.05, 0.10, 0.20, 0.30\}$[4] for the *skewing* parameter, and we finally chose 0.30, which got the best results on the GIS instances.

The models perform fairly well on certain instances, and the results where the performance of the model is at least as good as the best metaheuristic are marked in italic. However, both models struggle to find good bounds. Actually, they did not find better bounds than the longest distance between two nodes in any of the instances (see column "distances - max" in Table 1), which is a trivial one to determine and it is likely to be very far from the optimal solution. Much more surprising are the results obtained by the greedy algorithm proposed by Rosenkrantz et al. (2000) Even though in general its results lag behind the local search solvers, it is capable to be very competitive

[3] As explained at the beginning of the current section, to compare the models on the same computational resources the time limit for Models 1 and 2 is set to $2280s$. In the table we have rescaled the times to $3600s$ to ease the comparison.

[4] Upon recommendation from the authors of the solver.

Table 4
Model results on the MDG-b dataset.

Instance	Model 1				Model 2			
	1 h		12 h ×16vCPU		1 h		1 2h × 16vCPU	
	LB	UB	LB	UB	LB	UB	LB	UB
MDG-b_01_n500_b02_m50	33.6	1000.0	**64.6**	861.2	61.7	1000.0	**64.6**	1000.0
MDG-b_02_n500_b02_m50	25.6	1000.0	58.5	919.1	41.6	1000.0	**63.0**	1000.0
MDG-b_03_n500_b02_m50	20.1	1000.0	58.8	918.1	6.4	1000.0	**60.9**	1000.0
MDG-b_04_n500_b02_m50	32.8	1000.0	54.7	914.8	2.1	1000.0	**57.4**	1000.0
MDG-b_05_n500_b02_m50	47.7	1000.0	54.2	912.8	36.0	1000.0	**58.9**	1000.0
MDG-b_06_n500_b02_m50	37.5	1000.0	57.5	917.2	12.2	1000.0	**60.6**	1000.0
MDG-b_07_n500_b02_m50	32.9	1000.0	52.5	922.5	42.4	1000.0	57.4	1000.0
MDG-b_08_n500_b02_m50	31.4	1000.0	57.2	882.2	25.4	1000.0	**61.1**	1000.0
MDG-b_09_n500_b02_m50	41.9	1000.0	59.2	919.6	20.1	1000.0	59.9	1000.0
MDG-b_10_n500_b02_m50	29.3	1000.0	**63.0**	906.9	45.0	1000.0	**63.0**	1000.0
MDG-b_01_n500_b03_m50	13.2	1000.0	27.5	829.7	3.2	1000.0	**29.0**	985.9
MDG-b_02_n500_b03_m50	11.4	1000.0	27.4	844.2	1.1	1000.0	29.0	1000.0
MDG-b_03_n500_b03_m50	10.4	1000.0	28.7	843.9	2.0	1000.0	30.0	984.7
MDG-b_04_n500_b03_m50	9.5	1000.0	29.9	843.2	3.2	1000.0	29.9	911.5
MDG-b_05_n500_b03_m50	12.9	1000.0	29.2	847.5	1.7	1000.0	29.5	984.2
MDG-b_06_n500_b03_m50	14.9	1000.0	28.9	856.8	2.1	1000.0	28.8	704.5
MDG-b_07_n500_b03_m50	14.5	1000.0	29.6	831.5	2.9	1000.0	29.4	509.7
MDG-b_08_n500_b03_m50	17.5	1000.0	31.6	844.2	1.6	1000.0	**32.3**	455.7
MDG-b_09_n500_b03_m50	15.2	1000.0	28.7	835.1	1.8	1000.0	29.7	709.4
MDG-b_10_n500_b03_m50	12.9	1000.0	30.4	824.1	1.1	1000.0	30.4	692.5

Table 5
Results comparison between us and benchmark.

Inst.		Lu et al. (2023)			Mladenović et al. (2022b)			MNSA			
B	#	best	avg	worst	best	avg	worst	time	best	avg	worst
0.2	1	64.6	61.5	50.8	64.60	63.04	62.10	48.17	64.60	**63.16**	62.10
0.2	2	60.8	56.3	51.6	60.40	59.42	58.70	46.81	61.40	**60.21**	58.70
0.2	3	60.9	59.8	54.8	60.90	60.59	59.90	54.82	60.90	**60.75**	58.20
0.2	4	56.6	54.4	48.8	56.90	55.75	54.80	49.94	57.40	**56.63**	55.50
0.2	5	58.2	54.0	51.0	58.20	57.23	57.10	64.72	58.90	**58.66**	56.90
0.2	6	60.6	55.7	49.4	60.60	58.38	56.60	56.05	60.60	**59.62**	56.80
0.2	7	59.3	54.0	47.9	58.00	55.40	53.40	59.12	59.30	**58.41**	54.60
0.2	8	60.4	56.2	48.5	60.10	59.46	58.60	56.09	60.60	**59.90**	59.30
0.2	9	60.2	57.5	52.6	59.70	58.10	57.00	52.54	60.90	**60.08**	58.90
0.2	10	63.0	61.3	52.7	63.00	**62.48**	62.00	50.12	63.00	62.47	62.00
0.3	1	27.2	24.2	19.6	28.70	27.73	27.30	56.55	29.00	**28.40**	27.70
0.3	2	27.6	25.5	21.5	28.20	27.73	27.30	54.31	29.10	**28.67**	28.00
0.3	3	30.0	28.0	24.9	29.40	28.61	28.10	57.66	30.10	**29.95**	29.10
0.3	4	28.0	25.3	23.2	29.70	29.07	28.90	54.22	30.00	**29.85**	29.60
0.3	5	27.9	25.3	23.0	29.70	28.69	28.30	55.25	30.00	**29.48**	28.50
0.3	6	28.2	26.0	22.8	28.90	28.28	28.00	52.81	29.00	**28.90**	28.60
0.3	7	28.4	26.0	20.3	29.40	28.54	28.00	57.15	30.50	**30.08**	29.30
0.3	8	31.1	28.3	24.2	31.90	31.73	31.10	51.12	31.90	**31.84**	31.70
0.3	9	29.7	25.9	20.8	29.70	28.88	28.20	61.54	29.80	**29.76**	29.30
0.3	10	29.1	26.7	21.7	30.40	29.55	29.40	59.60	30.50	**30.02**	29.70

on a few instances. The reason might be that the GIS instances use realistic distances, which are effectively exploited by the greedy algorithm, while this was not possible on the MDG-b dataset. The results obtained by our method are presented in the columns MNSA. Our solver outperforms or matches the average performance of the others on the whole dataset (in 13 cases as sole best performer and in 17 cases as ties with other methods). Additionally, in 4 instances (GIS-6, 19, 23 and 25), our method finds a best solution that is unmatched by all others. The VNS method proposed by Mladenović et al. (2022b) also obtains a fair share of good solutions, while the solver by Lu et al. (2023) does not seem to be competitive on the GIS dataset.

6.3. Ablation analysis

Our solution method contains several adaptations from pre-existing local search algorithms in the literature as well as novel additions. While in Section 6.2 we have already shown the overall effectiveness of our solution approach, we are now interested in understanding the contribution of individual components of our solution method. We study hereafter the contribution of the randomized greedy start, of the lexicographic cost function and of the larger Swap neighborhood, in the form of an ablation analysis. Therefore, we selectively remove from our solution method these components and we evaluate the new performance of the method.

In order to ablate the randomized greedy start, we replace it with a random initial solution strategy. The random initial solution is generated starting from S equal to the empty set. Then, until the capacity requirement B is satisfied, and in any case while $|S| < 2$, we iteratively add a random node $v \in C$ and insert it in S. Therefore, the initial solution is generated completely at random, only ensuring that the capacity requirement is satisfied.

As per the lexicographic cost function, we replace it with the natural objective function of the problem. For the GIS instances, a new tuning of the neighborhood rates was necessary, as the configuration in Table 2 is very unbalanced against the Remove neighborhood, and we suffer a dramatic drop in the performance of the method if the lexicographic cost function is removed without performing any corrections. The new probabilities and biases used are $\sigma_I = 0.026$, $\sigma_R = 0.018$, $b_+ = 0.546$, $b_- = 0.985$.

Table 6
Results on GIS dataset.

	Model 1 (1 h)		Model 2 (1 h)		Rosenkrantz et al.			Lu et al.			Mladenović et al.			MNSA			
#	LB	UB	LB	UB	max	avg	min	max	avg	min	max	avg	min	time (s)	max	avg	min
1	*6*	74	*6*	74	6	5.40	5	6	**6.00**	6	6	**6.00**	6	184.98	6	**6.00**	6
2	2	136	4	136	5	**5.00**	5	5	**5.00**	5	5	**5.00**	5	304.06	5	**5.00**	5
3	*4*	159	*4*	159	4	3.90	3	4	3.05	3	4	**4.00**	4	170.17	4	**4.00**	4
4	3	149	3	149	4	3.70	3	3	3.00	3	4	**4.00**	4	202.59	4	**4.00**	4
5	*5*	104	*5*	104	5	4.30	4	4	3.92	3	5	**5.00**	5	157.19	5	**5.00**	5
6	5	229	5	229	6	6.00	6	6	5.10	5	6	6.00	6	282.28	7	**7.00**	7
7	7	118	*8*	118	7	6.50	6	7	6.97	6	8	7.05	7	181.47	8	**8.00**	8
8	9	215	7	215	11	**11.00**	11	9	7.85	7	10	10.00	10	347.17	11	**11.00**	11
9	6	81	6	81	7	6.50	6	7	6.10	6	7	**7.00**	7	286.77	7	**7.00**	7
10	28	211	11	211	34	32.40	32	31	28.57	26	34	**34.00**	34	95.57	34	**34.00**	34
11	*11*	45	*11*	45	10	9.00	8	11	**11.00**	11	11	**11.00**	11	182.52	11	**11.00**	11
12	14	125	*16*	125	16	15.90	15	15	13.35	12	16	15.95	15	141.15	16	**16.00**	16
13	6	84	*7*	84	7	6.10	6	6	6.00	6	7	**7.00**	7	170.53	7	**7.00**	7
14	7	131	7	131	8	**8.00**	8	7	6.67	6	8	**8.00**	8	187.33	8	**8.00**	8
15	5	33	5	33	5	5.00	5	5	4.94	4	6	5.18	5	320.46	6	**5.92**	5
16	10	179	10	179	12	11.30	11	10	9.40	9	12	11.10	11	218.46	12	**11.93**	11
17	2	292	4	292	5	4.90	4	5	4.02	4	5	**5.00**	5	300.81	5	**5.00**	5
18	40	247	33	247	42	41.20	40	40	32.02	11	43	**43.00**	43	90.52	43	**43.00**	43
19	13	222	13	222	15	14.90	14	14	12.47	10	15	14.00	13	95.73	16	**16.00**	16
20	6	76	6	76	6	5.60	5	6	6.00	6	7	6.18	6	160.02	7	**7.00**	7
21	*5*	49	5	49	5	4.40	4	5	4.85	4	5	**5.00**	5	197.35	5	**5.00**	5
22	*4*	200	*4*	200	4	**4.00**	4	4	3.90	3	4	**4.00**	4	246.52	4	**4.00**	4
23	6	264	7	264	7	6.50	6	7	6.52	6	7	7.00	7	204.61	8	**7.15**	7
24	3	170	5	170	6	5.80	5	5	5.00	5	6	5.07	5	315.18	6	**6.00**	6
25	4	68	4	68	4	4.00	4	4	4.00	4	4	4.00	4	189.27	5	**4.97**	4
26	7	133	4	133	7	7.00	7	6	6.00	6	8	**8.00**	8	276.75	8	**8.00**	8
27	6	31	*7*	31	6	6.00	6	7	6.60	6	7	**7.00**	7	200.72	7	**7.00**	7
28	4	121	4	121	5	5.00	5	5	5.00	5	6	5.05	5	333.48	6	**6.00**	6
29	15	129	19	129	22	21.10	21	20	16.65	14	22	21.15	20	96.19	22	**22.00**	22
30	4	141	4	141	5	4.10	4	4	4.00	4	5	4.11	4	238.89	5	**5.00**	5

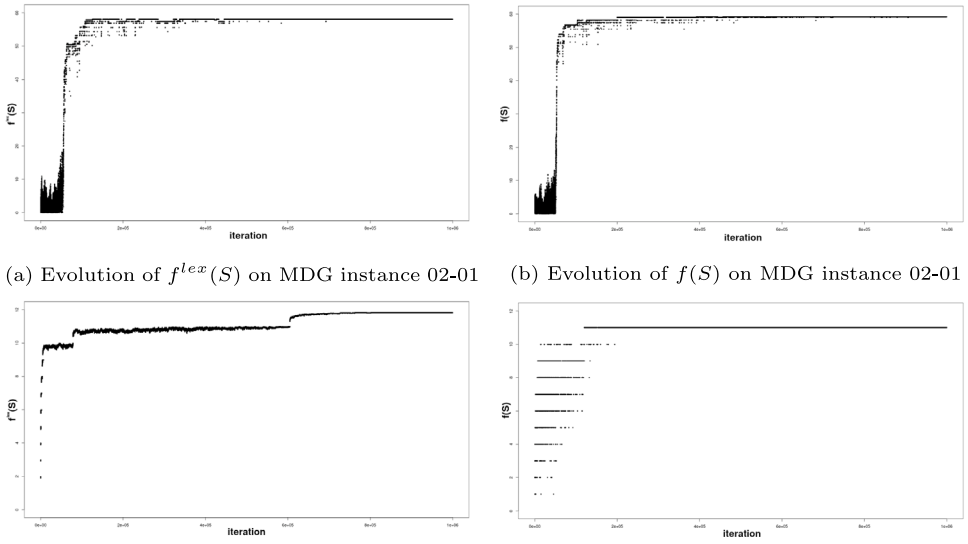
Table 7
Ablation analysis.

Instance	MNSA	no lex	no large swap	random start
1	**6.00**	**6.00**	**6.00**	**6.00**
2	**5.00**	**5.00**	**5.00**	**5.00**
3	**4.00**	3.88	**4.00**	**4.00**
4	**4.00**	3.50	**4.00**	**4.00**
5	**5.00**	**5.00**	**5.00**	**5.00**
6	**7.00**	6.00	**7.00**	**7.00**
7	**8.00**	7.10	7.53	**8.00**
8	**11.00**	**11.00**	**11.00**	10.32
9	**7.00**	**7.00**	**7.00**	**7.00**
10	**34.00**	**34.00**	32.98	33.90
11	**11.00**	**11.00**	11.00	**11.00**
12	**16.00**	15.85	15.70	**16.00**
13	**7.00**	6.55	**7.00**	**7.00**
14	**8.00**	**8.00**	**8.00**	**8.00**
15	**5.92**	5.00	5.35	5.88
16	**11.93**	11.43	11.40	11.90
17	**5.00**	**5.00**	**5.00**	**5.00**
18	**43.00**	**43.00**	41.33	**43.00**
19	**16.00**	**16.00**	15.15	**16.00**
20	**7.00**	6.00	6.08	**7.00**
21	**5.00**	4.75	**5.00**	**5.00**
22	**4.00**	**4.00**	4.00	**4.00**
23	7.15	7.00	7.00	**7.18**
24	**6.00**	**6.00**	**6.00**	**6.00**
25	**4.97**	4.00	4.80	**4.97**
26	**8.00**	**8.00**	**8.00**	**8.00**
27	**7.00**	**7.00**	**7.00**	**7.00**
28	**6.00**	5.00	**6.00**	**6.00**
29	**22.00**	21.50	20.88	**21.98**
30	**5.00**	4.03	**5.00**	**5.00**
sum(GIS)	29	15	17	26

Instance	MNSA	no lex	no large swap	random start
0.2 - 1	**63.16**	63.07	61.73	62.99
0.2 - 2	60.21	**60.25**	56.97	60.22
0.2 - 3	60.75	**60.83**	58.31	60.80
0.2 - 4	**56.63**	56.57	54.97	56.52
0.2 - 5	58.66	**58.71**	54.69	58.63
0.2 - 6	59.62	59.51	54.64	**59.77**
0.2 - 7	**58.41**	57.75	53.77	58.17
0.2 - 8	59.90	60.04	57.88	**60.09**
0.2 - 9	60.08	60.01	57.12	**60.10**
0.2 - 0	62.47	**62.56**	61.84	62.50
0.3 - 1	28.40	28.49	26.71	**28.53**
0.3 - 2	28.67	28.75	26.87	**28.78**
0.3 - 3	29.95	**29.96**	28.88	**29.96**
0.3 - 4	29.85	**29.87**	28.39	29.86
0.3 - 5	29.48	29.57	28.05	**29.64**
0.3 - 6	28.90	**28.91**	27.42	28.89
0.3 - 7	30.08	30.02	29.10	**30.20**
0.3 - 8	**31.84**	31.82	31.09	**31.84**
0.3 - 9	**29.76**	29.75	27.15	29.75
0.3 - 0	30.02	**30.04**	28.54	29.96
sum(MDG)	5	8	0	9
sum	34	23	17	35

Finally, to test the method without the larger Swap neighborhood, we avoid to sample it by setting $b_+ = 1.000$ and $b_- = 1.000$, so that only $\text{SwapR}_{\pm}$ is chosen by the algorithm.

Table 7 provides the results of the ablation analysis for MDG-b and GIS instances, carried over 40 runs per instance. The first column (MNSA) simply contains the results in Tables 5 and 6. For each instance, we mark in bold the configuration that obtains the best results. We also show a counter (sum) of the number of times each configuration is the best, separately for instance type as well as for the both datasets. The analysis of these results yield many interesting considerations. First of



(a) Evolution of $f^{lex}(S)$ on MDG instance 02-01 (b) Evolution of $f(S)$ on MDG instance 02-01

(c) Evolution of $f^{lex}(S)$ on GIS-16 (d) Evolution of $f(S)$ on GIS-16

Fig. 1. Search patterns on MDG-b and generated instances.

all, we observe that the behavior is not uniform on the two datasets. In the GIS instances, the table confirms that the configuration that uses all the components is the most effective in general. The decrease in the performance is greater if we remove either the lexicographic cost function or the larger Swap neighborhood, while starting the search from a random solution has a more limited effect, except for instance GIS-8. On the MDG-b instances, the situation is different. First of all, we observe a dramatic drop in the performance if we do not use the larger Swap neighborhood. On the other hand, if we remove the lexicographic cost function or if we start the search from a random solution, there is a little increase in the number of times these methods are the best. The differences, however, are small, and the only consideration that we can draw is that the larger Swap neighborhood is an essential components on MDG-b instances, while the others are less critical. Considering the sum on both dataset, we observe that the best configurations are the ones that use both the lexicographic cost function and the larger Swap, regardless of whether we start the search from a Random or from a Greedy solution. This might be explained by the fact that Multi-Neighborhood Search is provided with strong mechanisms against being trapped in local optima, therefore the choice of the initial solution is not critical. These observations are in line with results obtained in previous studies (see, e.g. Rosati et al., 2022).

6.4. Algorithmic insights

Fig. 1 illustrates the evolution in the cost and objective function value over short executions (1 million iterations) of our solver on a MDG-b instance and on a GIS instance. The y-axis represents the lexicographic function, respectively objective function, and the x-axis shows the iteration count. For both the MDG-b and the GIS instance we show on the left (Figs. 1(a) and 1(c)) the evolution of the lexicographic function (described in Section 4.5), with the configuration displayed in Table 2. The lexicographic cost function is divided by w_{of}, so that we can show the distance in its natural scale. On the right (Figs. 1(b) and 1(d)) we show the evolution of the algorithm that only uses the objective function, in the parameter configuration discussed in Section 6.3. In the case of the MDG-b instance, we observe that there is not a remarkable difference with or without the lexicographic objective, which is in accordance with the results of the ablation analysis. This is different from what we observe in the case of the GIS instance in Figs. 1(c) and 1(d). First of all, in Fig. 1(c) we see that the lexicographic objective is actively driving the metaheuristic in the exploration of the large plateaus, that can only be escaped thanks to the gradient offered by the auxiliary cost component. This is especially noticeable for the plateau found at distance $f(S) = 11$, that is eventually climbed to reach the final solution of cost $f(S) = 12$. We can also appreciate that the smaller scale of the bottleneck edge component prevents it from interfering with the main objective. In the case of not using the lexicographic objective we see instead that the algorithm performs many jumps in the value of $f(S)$ in the beginning of the search, but it gets stuck in the plateau with cost $f(S) = 11$ and it is not able to find a better solution of cost $f(S) = 12$.

Fig. 2 is aimed at illustrating the geographical dimension of the new instances. It shows two maps of the solutions obtained by our method on two different generated instances. We can observe that in the area on the left (Sofia, Bulgaria, GIS-22) the population concentration in the city together with a rather high capacity requirement (18%) makes it hard to sparsify the points. On the other hand, the instance on the right (Perugia, Italy, from the training set) has a lower capacity requirement and the more sprawled population distribution makes it easier to distribute the points over the map, even though a certain degree of concentration in denser portions of the area remains unavoidable.

Finally, Fig. 3 shows the pairwise distribution of different instance features for the 30 validation instances of dataset GIS. Besides the size of the graph $|V|$, we consider the relative capacity requirement B, the radius r, the ratio between the average distance and the radius $\mu(d)/r$, the interquartile range of the distances $IQR(d)$, the population in the area pop, and the ratio between the required capacity and the population B/pop.

We can appreciate how all the selected features distribute evenly along their domains, that was one of the goals of our generator. The color of the points indicates the average relative difference between the performance of the three solvers. There are some instances that present higher variability in the performance of the solvers, but no clear correlation on the features emerges.

7. Conclusions and future work

We proposed a multi-neighborhood approach guided by Simulated Annealing for the capacitated dispersion problem. The outcome has been that our approach, properly engineered and tuned, is able to outperform the current best results in the literature on almost all instances.

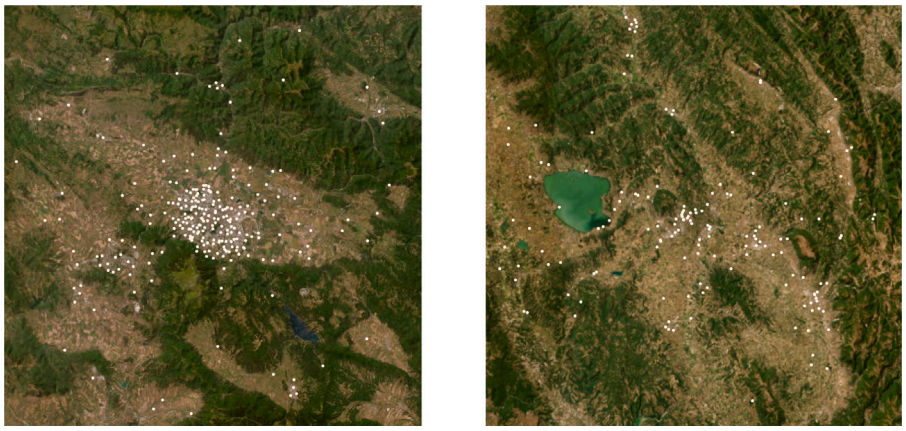


Fig. 2. Example of solutions on different urban areas. Left: Sofia, Bulgaria, $|V| = 1034$, $r = 47$ km, $B = 18\%$; Right: Perugia, Italy, $|V| = 597$, $r = 43$ km, $B = 4\%$; Maps are courtesy of ESRI.

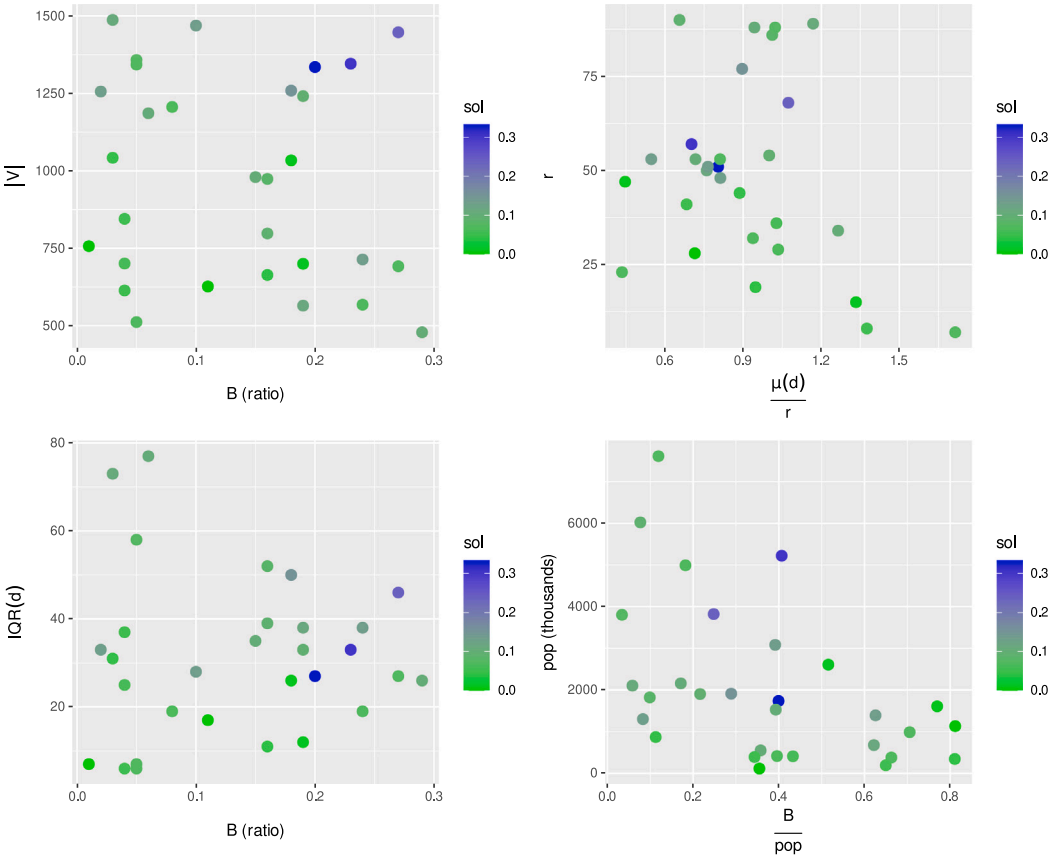


Fig. 3. Instance features and the corresponding average difference solution between solvers (color).

We assessed the contribution of individual components as well, and we showed that both the novel lexicographic cost function and the large Swap neighborhood are essential to the performance of the method, and are needed to tackle different types of instances.

We also proposed a new dataset that in our opinion has all the features it takes to become a new benchmark for this problem. At the same time, we propose to dismiss four out of five of the current datasets, as they are not challenging anymore, and their perpetuation brings a waste of work and computational time.

Finally, we developed and implemented two new compact mathematical models adapted to the CDP from pre-existing models in the literature for similar problems. We show that they are able to provide competitive results, especially on the MDG instances and on certain GIS instances.

For the future, we plan to extend and refine our neighborhoods, also including the ones proposed by Mladenović et al. (2022b), in order to understand if this would improve our results.

In addition, we plan to investigate the possibility of performing a feature-based tuning (Bellio et al., 2016), also including some reinforcement learning mechanisms, in order to reduce the computational cost of the tuning procedure and to make the algorithm more adaptive to the heterogeneous instance space (see, e.g. Ceschia et al., 2024).

Another possible research direction consists in strengthening the mathematical models by valid inequalities, like the ones proposed by Sayah and Irnich (2017), or adapting the procedure designed

by Pisinger (2006) for the computation of upper bounds for the p-dispersion problem to the CDP.

Furthermore, we would like to adapt our method to other versions of the dispersion problem, for example the generalized dispersion problem (see, e.g., Martinez-Gavara et al., 2021; Mladenović et al., 2022a), in order to see if it remains competitive, and which modifications are necessary to reach good results.

Finally, we plan to perform an instance space analysis (see, e.g., Smith-Miles & Muñoz, 2023) to identify the features of an instance that make it harder or easier for one specific search method.

CRediT authorship contribution statement

Roberto Maria Rosati: Conceptualization, Data curation, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Andrea Schaerf:** Conceptualization, Investigation, Methodology, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

We have shared the link to our data in the article.

Acknowledgments

This research has been funded by the Italian Ministry of University and Research under the action PRIN 2020 (Prot. 2020LNEZYC), project "Models and algorithms for the optimization of integrated healthcare management".

The authors acknowledge the European Commission (Eurostat, Joint Research Centre and DG Regional Policy - REGIO-GIS) for providing the GEOSTAT 1 km^2 population grid.

We thank Anna Martínez-Gavara, Giri Kumar Tayi, and S. S. Ravi for answering our questions on their work, and Raca Todosijević and Dragan Urošević for providing us the executable of their solver.

References

Bellio, R., Ceschia, S., Di Gaspero, L., & Schaerf, A. (2021). Two-stage multi-neighborhood simulated annealing for uncapacitated examination timetabling. *Computers & Operations Research*, *132*, Article 105300.

Bellio, R., Ceschia, S., Di Gaspero, L., Schaerf, A., & Urli, T. (2016). Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. *Computers & Operations Research*, *65*, 83–92.

Ceschia, S., Di Gaspero, L., Rosati, R. M., & Schaerf, A. (2022). Multi-neighborhood simulated annealing for the minimum interference frequency assignment problem. *EURO Journal on Computational Optimization*, *10*, Article 100024.

Ceschia, S., Di Gaspero, L., Rosati, R. M., & Schaerf, A. (2024). Reinforcement learning for multi-neighborhood local search in combinatorial optimization. In *Machine learning, optimization, and data science* (pp. 206–221). Cham: Springer Nature Switzerland.

Erkut, E., & Neuman, S. (1991). Comparison of four models for dispersing facilities. *INFOR: Information Systems and Operational Research*, *29*(2), 68–86.

Kirkpatrick, S., Gelatt, D., & Vecchi, M. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680.

Kuby, M. J. (1987). Programming models for facility dispersion: The p-dispersion and maxisum dispersion problems. *Geographical Analysis*, *19*(4), 315–329.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, *3*, 43–58.

Lu, Z., Martínez-Gavara, A., Hao, J. K., & Lai, X. (2023). Solution-based tabu search for the capacitated dispersion problem. *Expert Systems with Applications*, *223*, Article 119856.

Martí, R., Gallego, M., & Duarte, A. (2010). A branch and bound algorithm for the maximum diversity problem. *European Journal of Operational Research*, *200*(1), 36–44.

Martí, R., Martínez-Gavara, A., Pérez-Peló, S., & Sánchez-Oro, J. (2022). A review on discrete diversity and dispersion maximization from an OR perspective. *European Journal of Operational Research*, *299*(3), 795–813.

Martí, R., Martínez-Gavara, A., & Sánchez-Oro, J. (2021). The capacitated dispersion problem: An optimization model and a memetic algorithm. *Memetic Computing*, *13*, 131–146.

Martinez-Gavara, A., Corberan, T., & Martí, R. (2021). GRASP and tabu search for the generalized dispersion problem. *Expert Systems with Applications*, *173*, Article 114703.

Mladenović, N., Todosijević, R., & Urošević, D. (2022). Dispersion problem under capacity and cost constraints: Multiple neighborhood tabu search. In *International conference on mathematical optimization theory and operations research* (pp. 108–122). Springer.

Mladenović, N., Todosijević, R., Urošević, D., & Ratli, M. (2022). Solving the Capacitated Dispersion Problem with variable neighborhood search approaches: From basic to skewed VNS. *Computers & Operations Research*, *139*, Article 105622.

Peiró, J., Jiménez, I., Laguardia, J., & Martí, R. (2021). Heuristics for the capacitated dispersion problem. *International Transactions in Operational Research*, *28*(1), 119–141.

Pisinger, D. (2006). Upper bounds and exact algorithms for p-dispersion problems. *Computers & Operations Research*, *33*(5), 1380–1398.

Rosati, R. M., Petris, M., Di Gaspero, L., & Schaerf, A. (2022). Multi-neighborhood simulated annealing for the sports timetabling competition ITC2021. *Journal of Scheduling*, *25*(3), 301–319.

Rosenkrantz, D. J., Tayi, G. K., & Ravi, S. (2000). Facility dispersion problems under capacity and cost constraints. *Journal of Combinatorial Optimization*, *4*, 7–33.

Sayah, D., & Irnich, S. (2017). A new compact formulation for the discrete p-dispersion problem. *European Journal of Operational Research*, *256*(1), 62–67.

Shier, D. R. (1977). A min-max theorem for p-center problems on a tree. *Transportation Science*, *11*(3), 243–252.

Smith-Miles, K., & Muñoz, M. A. (2023). Instance space analysis for algorithm testing: Methodology and software tools. *ACM Computing Surveys*, *55*(12), 1–31.