Metaheuristic techniques for the capacitated facility location problem with customer incompatibilities

Article *in* Soft Computing · October 2022

DOI: 10.1007/s00500-022-07600-z

CITATIONS
6

READS
362

9 authors, including:

Marcelo Rodrigues de Holanda Maia
Instituto Brasileiro de Geografia e Estatística
11 PUBLICATIONS **45** CITATIONS
SEE PROFILE

Miguel Reula
University of Valencia
12 PUBLICATIONS **49** CITATIONS
SEE PROFILE

Prem Prakash Vuppuluri
Dayalbagh Educational Institute
27 PUBLICATIONS **104** CITATIONS
SEE PROFILE

Alexandre Plastino
Fluminense Federal University
119 PUBLICATIONS **1,799** CITATIONS
SEE PROFILE

Soft Computing
https://doi.org/10.1007/s00500-022-07600-z

OPTIMIZATION

Metaheuristic techniques for the capacitated facility location problem with customer incompatibilities

Marcelo R. H. Maia[1,2] · Miguel Reula[7] · Consuelo Parreño-Torres[3] · Prem Prakash Vuppuluri[4] · Alexandre Plastino[1] · Uéverton S. Souza[1] · Sara Ceschia[5] · Mario Pavone[6] · Andrea Schaerf[5]

Accepted: 19 October 2022

Abstract
We study a novel version of the capacitated facility location problem, which includes incompatibilities among customers. For this problem, we propose and compare on a fair common ground a portfolio of metaheuristic techniques developed independently from each other. We tested our techniques on a new dataset composed of instances of increasing size, varying from medium to very large ones. The outcome is that the technique based on data mining has been able to outperform the others in most instances, except for a few large cases, for which it is overcome by the simpler greedy one. In order to encourage future comparisons on this problem, we make the instances, solution validator and implementations of the metaheuristic techniques available to the community.

Keywords Facility location · Metaheuristics · Multi-sourcing

1 Introduction

We study the classical *Capacitated Facility Location Problem* (CFLP) in which facilities must be selected for opening and customers must be supplied by open facilities. Constraints concern customers whose demand must be completely satisfied and the capacity of facilities that cannot be exceeded. We consider the *multi-source* version of the problem, called MS-CFLP, in which each customer can be supplied by more than one facility. The objective function to minimize is the sum of opening and shipping costs, the latter being determined using an input matrix of per-unit costs.

Following the classification of facility location models proposed by Klose and Drexl (2005), our problem has the following features: network location model, minsum objective,

✉ Marcelo R. H. Maia
mmaia@ic.uff.br

Miguel Reula
miguel.reula@uv.es

Consuelo Parreño-Torres
consuelo.parreno@uv.es

Prem Prakash Vuppuluri
vpremprakash@dei.ac.in

Alexandre Plastino
plastino@ic.uff.br

Uéverton S. Souza
ueverton@ic.uff.br

Sara Ceschia
sara.ceschia@uniud.it

Mario Pavone
mpavone@dmi.unict.it

Andrea Schaerf
andrea.schaerf@uniud.it

1 Institute of Computing, Fluminense Federal University, Av. Gen. M. Souza, Niterói, RJ 24210-346, Brazil

2 ENCE, Brazilian Institute of Geography and Statistics, R. Gen. Canabarro 706, Rio de Janeiro, RJ 20271-020, Brazil

3 Departamento de Estadística e Investigación Operativa, Universidad de Valencia, C/ Doctor Moliner, 50, 46100 Burjassot, Valencia, Spain

4 Department of Electrical Engineering, Faculty of Engineering, Dayalbagh Educational Institute (Deemed to be University), Dayalbagh, Agra, Uttar Pradesh 282005, India

5 DPIA, University of Udine, Via delle Scienze 206, 33100 Udine, Italy

6 DMI, University of Catania, v.le A. Doria 6, 95125 Catania, Italy

7 Departamento de Didáctica de la Matemática, Universidad de Valencia, Avda. Tarongers, 4, 46022 Valencia, Spain

Published online: 31 October 2022

single-stage, capacity constraints, multi-sourcing, single-product, static and deterministic.

We add the constraint that some pairs of customers cannot be supplied by the same facility. This constraint has been inspired by the work by Rabbani et al. (2018) and Marín and Pelegrín (2019), and it models the situation in which customers have demands of different types that can be incompatible (e.g., hazardous wastes). We name the resulting problem as MS-CFLP-CI (for customer incompatibilities).

We propose a portfolio of metaheuristic techniques for this new problem, each developed independently by a different research group. The idea originates from the Metaheuristics Summer School (MESS 2020+1) as a competition of techniques among its participants.

In detail, we propose a MineReduce multi-start iterated local search, a GRASP approach, a permutation-coded evolutionary algorithm and a multi-start greedy technique.

The experimental analysis is performed on the same computing system in order to provide a fair ground for comparison. To this end, the same running time is granted to all techniques, which is related to the size of the instance solved.

All techniques have been tuned on a set of training instances and run on the (unseen) validation instances. Instances have a size ranging from 50 to 3000 facilities and have been created by an ad hoc generator.

The outcome is that the hybrid approach based on data mining outperforms all the others except for a few large cases, for which it is overcome by the simpler greedy one.

In order to foster future comparisons, all the data is made available on the web at the MESS 2020+1 website https://www.ants-lab.it/mess2020/#competition. It also includes the solution validator, which provides against possible misunderstanding in the formulation. Additionally, this material can be retrieved from https://github.com/MESS-2020-1, which also contains the source code of the different solution techniques.

The paper is organized as follows. Section 2 provides the definition of the problem. Section 3 illustrates the related work. The proposed solution techniques are discussed in Sect. 4. Experimental results are shown in Sect. 5. Finally, conclusions are drawn in Sect. 6.

2 Problem definition

Let $\mathcal{J} = \{1, \ldots, J\}$ be a set of *facilities*, such that each facility $j \in \mathcal{J}$ has a *capacity* s_j and an *opening cost* f_j, and $\mathcal{I} = \{1, \ldots, I\}$ a set of *customers*, such that each customer i has a *demand* of quantity of goods d_i to be completely satisfied by one or more facilities. We also have a *shipping cost* c_{ij} that is the cost *per unit* of transporting goods from facility j to customer i. Finally, we are given a set Γ of pairs of *incompatible customers*, such that for each $\langle i_1, i_2\rangle \in \Gamma$, i_1 and i_2 cannot be served by the same facility.

The problem consists in selecting values for the following decision variables: (i) nonnegative integer variables x_{ij} representing the quantity of goods moved from facility $j \in \mathcal{J}$ to customer $i \in \mathcal{I}$; (ii) binary variables y_j, such that a facility j is open if $y_j = 1$, closed otherwise.

The constraints are the following:

- The total quantity of goods taken from an open facility cannot exceed its capacity:

$$\sum_{i\in\mathcal{I}} x_{ij} \le s_j y_j \quad \forall j \in \mathcal{J} \tag{1}$$

- The total quantity of goods brought to a customer must be exactly equal to its demand:

$$\sum_{j\in\mathcal{J}} x_{ij} = d_i \quad \forall i \in \mathcal{I} \tag{2}$$

- Two incompatible customers cannot be supplied by the same facility:

$$x_{i_1 j} = 0 \vee x_{i_2 j} = 0 \quad \forall\langle i_1, i_2\rangle \in \Gamma, \forall j \in \mathcal{J} \tag{3}$$

- Domain of decision variables:

$$0 \le x_{ij} \le d_i \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \tag{4}$$

$$y_j \in \{0, 1\} \quad \forall j \in \mathcal{J} \tag{5}$$

Note that Constraints (1) ensure that if facility j is not open, no demand for any customer can be filled from it. The model above is not linear as it contains disjunctions in Constraints (3), but it could be easily linearized using the classical big M technique.

The objective function is the sum of two components: the cost of opening the selected facilities and the cost to ship the goods from the facilities to the customers.

$$min\ z = \sum_{j\in\mathcal{J}, i\in\mathcal{I}} c_{ij} x_{ij} + \sum_{j\in\mathcal{J}} f_j y_j \tag{6}$$

We assume that all input values are nonnegative integers and the total capacity of facilities is enough for the total demand, i.e., $\sum_{j\in\mathcal{J}} s_j \ge \sum_{i\in\mathcal{I}} d_i$.

Instances are written in the MiniZinc data format (Nethercote et al. 2007). An example of an input file is shown in Fig. 1.

The uncapacitated FLP (without customer incompatibilities), where $s_j = +\infty, \forall j \in \mathcal{J}$, was proven to be $\mathcal{NP}$-hard by Cornuéjols et al. (1983) through a reduction from the node cover problem; hence, also the MS-CFLP-CI is $\mathcal{NP}$-hard.

Fig. 1 An example of an input file in MiniZinc format

```
Facilities = 4;
Customers = 10;

Capacity = [100, 40, 60, 60];
FixedCost = [860, 350, 440, 580];
Demand = [12, 17, 5, 13, 20, 20, 17, 19, 11, 20];
ShippingCost = [|27, 66, 44, 55
                |53, 89, 68, 46
                |17, 40, 18, 61
                |20, 68, 44, 78
                |42, 89, 65, 78
                |57, 55, 49, 31
                |89, 101, 90, 16
                |37, 31, 23, 55
                |76, 60, 63, 44
                |82, 107, 91, 31|];

Incompatibilities = 3;
IncompatiblePairs = [| 1, 10 | 2, 7 | 8, 9 |];
```

3 Related work

Location problems have been widely studied in the literature under the names of plant, warehouse or facility location problems. The literature on this research area is vast. Thus, we decided to focus our review on contributions related to the specific variant of the problem MS-CFLP, i.e., the deterministic multi-source capacitated one, where demands are deterministic, each customer can be served by multiple facilities and each facility has a maximum capacity. A more comprehensive overview of models and algorithms for facility location problems can be found in the surveys by Klose and Drexl (2005), Melo et al. (2009) and Fernández and Landete (2015).

Many solution approaches are ad hoc algorithms that exploit the mathematical structure of the models. In particular, several methods use some form of Lagrangian relaxation (Barahona and Chudak 2005; Avella et al. 2009; Görtz and Klose 2012; Caserta and Voß 2020).

To the best of our knowledge, state-of-the-art algorithms for solving the MS-CFLP are the exact ones proposed by Fischetti et al. (2016) based on Benders decomposition and the Lagrangian-based branch-and-bound by Görtz and Klose (2012). Among heuristic methods, the best performance is obtained by Guastaroba and Speranza (2012) who implemented the MIP-based heuristic called *Kernel Search* and Caserta and Voß (2020) who presented a matheuristic based on the *Corridor Method*. These methods are tested on well-known datasets available from the OR-Library and proposed by Avella and Boccia (2009) and Avella et al. (2009).

Guastaroba and Speranza (2012) solved to optimality instances with up to 1000 facilities and 1000 customers within 1 h and improved most of the best known values for large instances (up to 2000 facilities and 4400 customers). For those instances (Avella et al. 2009), denoted as *Test Bed A*, *B* and *C*, Fischetti et al. (2016) were able to prove optimality for 210 out of 445 cases in 50,000 s, while Caserta and Voß (2020) matched the optimal solutions for 79 of 295 instances (*Test Bed C* was not tested) with an average running time of less than 600 s. Among exact methods, the B&B of Görtz and Klose (2012) remains competitive in terms of running times with Fischetti et al. (2016) for medium-sized instances.

Recently, Avella et al. (2021) devised a new class of valid inequalities that, embedded into a cut-and-branch procedure, improved many upper and lower bounds for 2000 × 2000 instances of *Test Bed C* with a time limit of 1 h.

4 Solution techniques

This section presents the solution techniques autonomously developed by the different research groups in the context of the Metaheuristics Summer School (MESS 2020+1). In detail, MineReduce-based multi-start ILS was implemented by Marcelo Maia, under the supervision of Alexandre Plastino and Uéverton Souza; GRASP by Miguel Reula and Consuelo Parreño-Torres; permutation-coded evolutionary algorithm by Prem Prakash Vuppuluri, and multi-start greedy algorithm by Sara Ceschia, Mario Pavone and Andrea Schaerf (competition organizers).

4.1 MineReduce-based multi-start ILS

MineReduce is an approach based on data mining for problem size reduction that has obtained promising results (Maia et al. 2020). This section describes an application of the MineReduce approach for the MS-CFLP-CI that uses a multi-start iterated local search (Lourenço et al. 2003) as a base metaheuristic.

4.1.1 MineReduce elements

The MineReduce approach applies patterns extracted from good solutions (using a data mining technique) to perform problem size reduction—a process in which a problem instance is reduced to a smaller-size version, the reduced instance is solved, and the solution found is expanded, so that it becomes a solution for the original instance.

The elite set E keeps the η best solutions among those obtained by the end of the iterations in the multi-start structure. When E is considered *stable*, a data mining method is used to extract patterns from its solutions. This happens when one of the following criteria is met: (i) E has not been mined yet, and its contents have not changed for a number of consecutive multi-start iterations greater than $\alpha \times I_{max}$, where I_{max} is the maximum number of iterations, dynamically estimated based on the elapsed time and the number of completed iterations; (ii) E has not been mined yet, $|E| = \eta$ and the elapsed time is greater than half the time limit; or (iii) the contents of E have changed after it was last mined, but they have not changed for a number of consecutive iterations greater than $\alpha \times I_{max}$.

The data mining method returns the β largest frequent itemsets with minimum support γ found in E. They are frequent sets of pairs $\langle i, j \rangle$, i.e., sets of customer-facility assignments with a relative frequency greater than or equal to γ in the best solutions. After these frequent itemsets are mined, their assignments are filled with the corresponding minimum quantities in E, i.e., for each pair $\langle i, j \rangle$ in a frequent itemset, a supplied quantity q is set, which corresponds to the minimum positive supplied quantity for that pair among all solutions in E. Therefore, each final pattern is a set of assignments $\langle i, j, q \rangle$.

Patterns are used to reduce the problem instance size based on the assumption that their elements shall be part of the solution. In the particular case of reducing this problem, an extension to the problem formulation is needed. We introduce incompatibilities between customers and facilities, represented by a set Γ', such that for each $\langle i, j \rangle \in \Gamma'$, i cannot be supplied by j. Hence, an additional set of constraints is introduced:

- A customer cannot be supplied by an incompatible facility:

$$x_{ij} = 0 \quad \forall \langle i, j \rangle \in \Gamma' \tag{7}$$

The reduction process works as follows. Given a problem instance $\mathcal{P}$ and a pattern p, the reduced instance $\mathcal{P}'$ is initialized as a copy of $\mathcal{P}$. Then, for each assignment $\langle i, j, q \rangle$ in p, the following changes are applied to $\mathcal{P}'$: (i) Decrease both s_j and d_i by q; (ii) Set $f_j = 0$; and (iii) For each customer i' incompatible with i (from Γ), add a pair $\langle i', j \rangle$ to Γ'.

As a consequence, the domain of each x_{ij} variable will be reduced, and Constraints (7) will fix the values of many other decision variables, effectively reducing the problem instance size.

Finally, a solution for $\mathcal{P}'$ can be expanded into the equivalent solution for $\mathcal{P}$ by including all assignments from p.

4.1.2 Algorithmic description

The MineReduce-based multi-start ILS (MR-MS-ILS) structure is depicted in Algorithm 1, where σ^* is the best solution found. At each iteration of the multi-start structure, an initial solution is generated (line 4) and improved by the ILS (line 5). E is updated by the end of each iteration (line 6), as well as σ^* in case of improvement (line 7). Whenever one of the stabilization criteria is met, P is filled with patterns mined from E (line 3).

Algorithm 1 MR-MS-ILS

```
1: $z(\sigma^*) \leftarrow \infty$; $E \leftarrow \varnothing$; $P \leftarrow \varnothing$
2: while time limit not exceeded do
3:     if STABLE($E, \alpha$) then $P \leftarrow$ MINE($E, \beta, \gamma$)
4:     $\sigma_0 \leftarrow$ INITIALSOLUTION($P, \varepsilon$)
5:     $\sigma \leftarrow$ ILS($\sigma_0, \delta, \kappa$)
6:     UPDATEELITE($E, \sigma, \eta$)
7:     if $z(\sigma) < z(\sigma^*)$ then $\sigma^* \leftarrow \sigma$
8: end while
9: return $\sigma^*$
```

4.1.3 Initial solution

The default initial solution generation (when $P = \varnothing$) works as follows. First, it opens facilities until the total capacity of all open facilities is greater than or equal to the total demand of all customers in $\mathcal{I}$. Then, for each open facility j, a customer i is drawn, and an assignment $\langle i, j, q \rangle$ is added to the solution.

Once all open facilities have been initialized with an assignment, the procedure iterates upon all customers and, for each customer i, adds to the solution the cheapest feasible assignments considering the open facilities only, until the demand of i is fully met or there is no feasible assignment possible with the currently open facilities. When the latter situation happens, the procedure opens another facility and resumes the assignments.

The quantity of goods q chosen for an assignment $\langle i, j, q \rangle$ is always the maximum possible one, which corresponds to the minimum between the residual demand of i and the residual capacity of j.

Two alternative strategies for choosing the next facility to open are considered. The *greedy opening* (GO) strategy chooses the facility j with the lowest ratio f_j/s_j among all closed facilities. The *random opening* (RO) strategy draws a facility from the set of closed facilities using the roulette wheel method, with a selection probability for each facility j proportional to its ratio s_j/f_j. The strategy to be used is defined by the parameter ε.

When $P \neq \varnothing$, patterns have already been mined from E. In this case, the MineReduce approach generates an initial solution through a reduce–optimize–expand process. Firstly, the problem instance is reduced based on one of the patterns in P. Then, a solution for the reduced instance is obtained by applying the default initial solution generation and the ILS. Finally, the solution is expanded into its equivalent for the original instance.

4.1.4 Iterated local search

The strategies applied in this ILS were inspired by an iterated tabu search proposed for the single-source CFLP by Ho (2015). Solution $\hat{\sigma}$ is initialized with the result of a local search on the initial solution σ_0. At each iteration, $\hat{\sigma}$ is perturbed resulting in solution σ', which is then improved by a local search to obtain solution $\bar{\sigma}$. If solution $\bar{\sigma}$ satisfies the acceptance criterion $z(\bar{\sigma}) < (1 + \delta) \times z(\bar{\sigma}^*)$, the search continues from solution $\bar{\sigma}$ (i.e., $\hat{\sigma} \leftarrow \bar{\sigma}$).

Two neighborhood structures for a solution σ are used. Neighborhood $\mathcal{N}_1(\sigma)$ consists of all feasible solutions that can be obtained from σ by reassigning goods supplied to a customer i from a facility j_1 to another facility j_2. This type of move reassigns the maximum possible quantity of goods, which is the minimum between x_{ij_1} and the residual capacity of j_2. Neighborhood $\mathcal{N}_2(\sigma)$ consists of all feasible solutions that can be obtained from σ by exchanging a customer i_1 from a facility j_1 with another customer i_2 from another facility j_2. This type of move makes a full reassignment of the supplied goods, i.e., all goods supplied to i_1 by j_1 are reassigned to j_2 and vice-versa.

A multi-improvement (MI) strategy (Rios et al. 2016; Silva et al. 2022), which applies a sequence of multiple independent moves at each step, is used for neighborhood exploration. Two moves are independent if the feasibility and cost improvement of any of these moves are not affected by the application of the other to the same solution (i.e., if they have no facilities in common).

Let σ be the seed solution from which the local search starts. First, all possible moves from σ to solutions in $\mathcal{N}_1(\sigma) \cup \mathcal{N}_2(\sigma)$ are evaluated, and a priority queue of improving moves M is built in decreasing order of cost improvement. Then, a series of steps are performed until M is empty or the time limit is exceeded. The following actions are taken at each step: (i) every move in M that is independent of all its predecessors is applied, and (ii) M is updated.

At each iteration of the ILS, one of the following perturbation operators is randomly selected and applied to solution $\hat{\sigma}$:

1. *Close one facility*. Randomly choose a facility j that supplies only one customer i and close it. Then reassign the goods supplied to i by j to the remaining open facilities by making the cheapest feasible assignments.
2. *Open one facility*. Randomly choose a closed facility and open it.
3. *Close one facility and open one facility*. Randomly choose an open facility j_1 and a closed facility j_2 such that $s_{j_2} \geq \sum_{i \in \mathcal{I}} x_{ij_1}$. Close j_1 and open j_2, then reassign all supplies from j_1 to j_2.
4. *Close one facility and open two facilities*. Select an open facility j_1 and two closed facilities j_2 and j_3 such that $s_{j_2} + s_{j_3} \geq \sum_{i \in \mathcal{I}} x_{ij_1}$ and the opening cost improvement is maximum. Close j_1 and open j_2 and j_3, then reassign all supplies from j_1 to j_2 and j_3 by making the cheapest feasible assignments.
5. *Open one facility and close two facilities*. Select a closed facility j_1 and two open facilities j_2 and j_3 such that: $s_{j_1} \geq \sum_{i \in \mathcal{I}} (x_{ij_2} + x_{ij_3})$, there are no incompatibilities between customers supplied by j_2 and customers supplied by j_3, and the opening cost improvement is maximum. Open j_1 and close j_2 and j_3, then reassign all supplies from j_2 and j_3 to j_1.

For the perturbation mechanism to be effective, the local search will not consider any move that would reverse a facility opening or closure applied by the perturbation operator at iteration v of the ILS.

4.1.5 Parameters tuning

The irace package (López-Ibáñez et al. 2016) was used to tune the parameters independently on five subsets of the training instances, defined by size ranges. Table 1 presents the best configurations found.

Table 1 Best parameter configurations for MR-MS-ILS

Range	α	β	γ	δ	ε	η	κ
$J \leq 150$	0.07	10	0.4	0.01	RO	5	100
$150 < J \leq 600$	0.03	6	0.9	0.01	GO	10	200
$600 < J \leq 1400$	0.04	6	0.8	0.05	GO	5	100
$1400 < J \leq 2000$	0.03	6	0.8	0.05	GO	5	100
$J > 2000$	0.04	1	1.0	0.02	GO	5	200

4.2 GRASP

In this section, we describe a greedy randomized adaptive search procedure (GRASP) to solve the MS-CFLP-CI. This method has been applied in a large number of problems and has behaved like a very robust metaheuristic procedure (Festa and Resende 2018). Algorithm 2 shows the pseudocode of the algorithm proposed, which consists of a constructive phase, Sect. 4.2.1, and of an improvement phase, Sect. 4.2.2. The construction phase generates a set of β feasible solutions and, in line 4, returns the solution with the lowest objective function $z(\sigma_0)$. If the current solution σ_0 is a candidate for improvement, the improvement phase is carried out in the framework of the variable neighborhood descent (VND) algorithm in line 5. It is a candidate to be improved if it is the first iteration of GRASP or if by improving σ_0 by 5% more than the best improvement so far b, the overall GRASP solution σ^* could be upgraded, $z(\sigma_0) - 1.05b < z(\sigma^*)$. The 5%, as well as other parameters used throughout the algorithm, have been adjusted by performing an extensive computational analysis comparing different parameter values.

Algorithm 2 Greedy randomized adaptive search procedure

```
1: function GRASP($\beta$)
2:     $z(\sigma^*) \leftarrow \infty$;
3:     while time limit not exceeded do
4:         $\sigma_0 \leftarrow$ CONSTRUCTIONPHASE ($\beta$)
5:         if CANDIDATE ($\sigma_0$) then $\sigma \leftarrow$ VND($\sigma_0$)
6:         end if
7:         if $z(\sigma) < z(\sigma^*)$ then $\sigma^* \leftarrow \sigma$
8:         end if
9:     end while
10:    return $\sigma^*$
11: end function
```

4.2.1 Construction phase

The construction phase runs β times a randomized heuristic algorithm in order to generate β feasible solutions. Only the best of those β feasible solutions moves on to the improvement phase.

The randomized heuristic algorithm consists of a combination of greedy and random routines. The first and third stages are randomized, while the second one is completely deterministic. The following vectors will be used to describe it.

- For each facility $j \in \mathcal{J}$, the fixed opening cost per unit is calculated, i.e., f_j/s_j and we generate the vector o^f, sorting it by increasing order of these quotients, and in the event of a draw, by increasing capacity of the facility j.
- For each customer $i \in \mathcal{I}$, we generate a vector $best^f[i]$ with the five facilities from which it is cheapest to serve it, sorting them by increasing shipping cost, and in the event of a draw, by increasing opening cost of the facility j.
- For each facility $j \in \mathcal{J}$ which is the most suitable for some customer, the vector $best^c[j]$ contains all the customers for which facility j is the most suitable. These facilities are ordered by non-increasing difference between the customer's shipping cost from its second-best facility and the customer's shipping cost from facility j; in case of a tie, by non-increasing difference between the customer's shipping cost from its third-best facility and the customer's shipping cost from facility j; and, in case of another tie, by non-increasing difference between the customer's shipping cost from its fourth-best facility and the customer's shipping cost from facility j. If there is still a tie, it is ordered by non-increasing demand of the corresponding customer.
- We use the vector $rand^c$ to refer to a vector composed of all randomly ordered customers. It is randomized before each algorithm's stage.

The three stages of the randomized heuristic algorithm are listed below.

Stage 1. Initialization We go through vector $rand^c$. Given a customer i, we try to satisfy its demand by using the most suitable facilities for it, those of $best^f[i]$. We select an unopened facility in $best^f[i]$ (if they are all open, we move on to the next customer) and allocate the unassigned demand of customer i. Let this facility be j, we go through vector $best^c[j]$, assigning if possible the demand of its customers to the facility j. Once the facility has covered δ% of its capacity, the procedure is finished, and this allocation becomes part of the partial solution. Otherwise, if all vector $best^c[j]$ has been traversed and this amount has not been covered, facility j is closed and, therefore, the customers we would have allocated are de-allocated.

Stage 2. Ordered facilities We go through vector o^f. Given a facility j, satisfying $j \in \bigcup_{i \in \mathcal{I}} best^f[i][1]$. We select the first unassigned customer in $best^c[j]$, i. Next, we assign the demand of customer i to the facilities in the vector $best^f[i]$ that are already open, have availability and are not assigned to incompatible customers. If not all of the demand of customer i has been allocated and facility j is unopened, we open it and allocate the demand of the customer. When facility j is no longer available, the stage moves on to another facility.

Stage 3. Remaining allocation Finally, a third stage is applied in order to achieve feasible solutions, so we have to cover the unassigned demand of all the customers. We go through vector $rand^c$. Given a customer with unassigned demand, customer i, we go through already open facilities where it can be assigned (there is availability and there are no customers incompatible with i assigned) by non-increasing shipping cost, assigning the goods of the customer until the customer is completely served. If the process finishes and there is still unassigned demand of customer i, we go through vector $best^f[i]$, opening facilities not already opened and assigning the goods of the customer to the facility until the customer is completely served. If the process finishes and there is still unassigned demand of customer i, the unopened facilities are randomly cycled through, opening the facilities and assigning the goods of the customer to them until the customer is completely served.

4.2.2 Variable neighborhood descent

In this algorithm, we have implemented a sequential VND in which the search is performed by exploring neighborhood structures of a current solution one after another. The algorithm requires as input an initial solution σ_0 and an ordered list N of neighborhood structures that must be examined. Once an improving solution is detected in some neighborhood structure, it replaces the current solution and the search restarts from it by exploring the first neighborhood structure in the list LS. Otherwise, if there are no improving solutions in the neighborhood LS_k, the search is continued by exploring N_{k+1} and so on up to the $LS_{k_{max}}$.

We have designed and implemented three local search algorithms describing three neighborhood structures ($k_{max} = 3$). In all of them, the first-improvement search strategy is used, replacing the current solution and restarting the search from it each time a better solution is obtained. Each local search algorithm is run iteratively until a total of $m_N = 20$ iterations without improvement are reached. They all are based on the well-known destruction-and-repair method and share the repair phase but differ in the destruction phase. All details are described below:

Destruction phase This phase destroys a part of the current solution. The selection of the elements to be removed is completely randomized. In LS_1, we choose between 1% and 5% of the open facilities in the current solution. Then, the corresponding demand units of those customers that were served by these facilities are left unassigned. In LS_2, between 1% and 5% of the opened facilities are also randomly selected but, instead of closing them, we unassign up to 50% of the customers' demands that were served from this facility. Finally, in LS_3, we select between 1 and 5% of the customers and completely unassign them.

Repair phase It goes through the unassigned customers. Given a customer i, the vector of facilities to which it can be assigned (without incompatible customers and with sufficient availability) is split into two sets according to the following criteria:

Set 1 Consists of already opened facilities and of closed facilities whose capacity is greater than or equal to the unassigned demand of the customer i.
Set 2 Facilities that do not meet the conditions to be in set 1.

Within both sets, facilities are ordered by increasing unit cost. Calculating the unit cost as the shipping cost plus, if the facility is still closed, the quotient between its fixed opening cost and the maximum demand of customer i that could be allocated by opening the corresponding facility. The repair phase goes through both sets, first through set 1 and then through set 2, assigning the goods of the customer to the facilities until the customer is completely served.

4.3 Permutation-coded evolutionary algorithm (PcEA)

This section describes an evolutionary algorithm for the problem that encodes solutions as concatenated permutations of customers and facilities. Two different sets of variation operators are probabilistically applied separately to the customer and facility genes to evolve solutions in a 1-elitist, steady-state evolutionary framework. If there are no improvements for a certain number of (empirically determined) iterations, the population is reinitialized to a set of random solutions seeded with the globally best solution obtained thus far. The key elements of the proposed permutation-coded evolutionary algorithm (PcEA) are discussed further as follows, and pseudocode is given in Algorithm 3.

4.3.1 Solution representation

Chromosomes are encoded as concatenated permutations of customers and facilities. A permutation of customers indicates the sequence in which customer demands are to be satisfied, and the associated facility permutation represents the order in which facilities are to be opened to serve customer demands. A chromosome is thus comprised of $|customers| + |facilities|$ genes.

4.3.2 Objective function decoder

Each chromosome is decoded as follows: Facilities are opened in permutation order until the total capacity of the opened facilities matches the combined customer demand. Thereafter, in the sequence defined by the customers' permutation, each customer is allocated goods from the open facility having the lowest allocation cost possible while ensuring that there are no conflicts with customers that have already been allocated goods from that facility. If no open facility satisfies this no-conflict requirement, then the next facility in the facility permutation is opened, and the corresponding customer demand is satisfied. If all facilities have already been opened, then the facility with the lowest increase in cost is chosen for allocation. In the pseudocode, the decoder is invoked in lines 4 and 23.

Algorithm 3 Permutation-coded evolutionary algorithm for MS-CFLP-CI

```
1:  $z(\sigma^*) \leftarrow \infty$;
2:  $P \leftarrow$ INIT( )   ▷ Initialize the population with random permutations
3:  for each $p \in P$ do
4:      if $z(p) < z(\sigma^*)$ then
5:          $\sigma^* \leftarrow p$
6:      end if
7:  end for
8:  while time limit not exceeded do
9:      option ← SELECTSUBPOPULATION( )   ▷ Choose either of customer or facility sub-populations with equal probability
10:     if option = CUSTOMER then
11:         Set OX as Recombination operator
12:     else   ▷ if option = FACILITY
13:         Set APX as Recombination operator
14:     end if
15:     Set SWAP as Mutation operator
16:     Select parent(s) using 3-T's with replacement
17:     pr ← RANDOM( )
18:     if pr < $p_c$ then
19:         $\sigma \leftarrow$ RECOMBINATION( )
20:     else
21:         $\sigma \leftarrow$ MUTATION( )
22:     end if
23:     if $z(\sigma) < z(\sigma^*)$ then
24:         $\sigma^* \leftarrow \sigma$
25:     end if
26:     REPLACEWORST($\sigma$)
27:     if no improvements for MAX_IMPROVE iterations then
28:         $P \leftarrow$ REINIT($\sigma^*$)   ▷ Reinitialize the population, seeding it with the best solution obtained thus far
29:     end if
30: end while
31: IMPROVE($\sigma^*$)
32: return $\sigma^*$
```

4.3.3 Evolutionary framework: operators and policies

The PcEA is a steady-state, 1-elitist evolutionary algorithm that uses two different recombination operators, one for the customer permutations and the other for facility permutations. The population is initialized to a fixed set of chromosomes, each comprised of a random permutation of customers, concatenated with a random permutation of facilities (line 2 of the pseudocode). If there is no improvement in the objective function value for a preset number of iterations, the population is re-initialized to a set of randomly generated chromosomes and seeded with the global best solution obtained thus far (line 28). Parents are selected using tournaments of size three with replacement (line 16). If two solutions have the same objective function value, then the solution with a lesser number of violations is preferred. Ties are broken arbitrarily. For evolving better customer permutations, Davis' order crossover (OX) operator (Davis 1991) is applied. The OX crossover operates in linear time and has the added benefit of preserving the relative order of the customers inherited from good parents. The OX crossover takes two parental customer permutations, i_1 and i_2, with corresponding facility permutations j_1 and j_2, and produces two offspring customer permutations. The offspring that results in the best total cost when combined with either j_1 or j_2 replaces the worst solution in the population if it has a lower cost (line 26). Facilities listed earlier in the facility permutation have a greater chance of being included in the set of opened facilities as compared to those that occur further down in the permutation. This implies that such facilities would appear early in parental facility permutations. Good offspring should include facilities that were likely used by their parents. A suitable crossover operator that enables offspring to inherit such "good" parental facilities is the alternating-position (APX) crossover (Larrañaga et al. 1997), which builds the offspring by first selecting alternate genes from each parent and then removing duplicates by deleting each latter duplicate entry in the list. For example, consider parent permutations $p_1 = (1, 0, 3, 2, 4)$ and $p_2 = (3, 1, 4, 0, 2)$. Starting with the first location in p_1, we generate the string $(1, 3, 0, 1, 3, 4, 2, 0, 4, 2)$. Retaining only the first occurrence of each number, we get the offspring $(1, 3, 0, 4, 2)$. Mutation is performed using a simple, constant-time swap mutation operator. The algorithm attempts to improve the global best solution (line 31) by checking, for each opened facility, if it is possible to assign the customers it is allocated to another facility at a lower cost, and reallocating accordingly. This improved solution is then returned by the algorithm.

4.3.4 Parameter selection

Several calibration trials were performed using a fractional factorial design approach for setting the parameter values. In the course of the trials, the probabilities of crossover (p_c) and mutation (p_m) were set to 0.6 and 0.4, respectively. The probability of evolving either the customer or facility sub-populations was kept equal. The population was comprised of 40 chromosomes and was re-initialized in the absence of any improvement in the fitness of the best solution of the population over MAX_IMPROVE = 10,000 iterations.

4.4 Multi-start greedy algorithm

Our last technique is a multi-start greedy (MG) algorithm, whose pseudocode is reported in Algorithm 4.

The greedy procedure selects at each step a triple $\langle i, j, q\rangle$, where i is a customer, j is a facility, and q is the quantity of goods to be moved from j to i. The algorithm stops when all customers are fully supplied (function UNSERVEDCOSTUMERS in line 3).

The procedure for selecting the triple iterates upon all pairs $\langle i, j\rangle$ searching for the "best" feasible one. The quantity q to be moved from j to i is the maximum possible one, which corresponds to the minimum between the residual amount of i and the residual capacity of j at that step (see line 24). An assignment is feasible if it does not violate customer incompatibilities with previous assignments (see function COMPATIBLE in line 8).

The best pair is the one that corresponds to the minimum supply cost (per unit), but taking into account also the opening cost in the case that the assignment is the first one to the facility j. The opening cost is not accounted for in full, as this choice would penalize too much the assignments to currently closed facilities (given that a certain number of facilities have to be opened anyway). That is, we count the opening cost multiplied by the fraction of s_j moved to i, in turn, multiplied by a factor α, which is a parameter of the method (see line 12). For example, if $f_j = 400$, $s_j = 50$ and $x_{ij} = 10$, then the share of the opening cost considered in the computation of the best pair is $400 \times (10/50) \cdot \alpha = 80\alpha$.

The algorithm also implements a *random tie-break* strategy (see line 18), which selects in a uniform random way in case of equal cost. Given that an exactly equal cost is rather unlikely, we consider as equal all choices within a given threshold β above the current best pair $\langle i, j\rangle$. The threshold β is the second and last parameter of the method, which is tuned on the training instances. Given that the whole procedure runs faster than the granted time, it is repeated several times as long as the timeout is not exceeded. The best solution obtained upon all runs is returned.

The tuning procedure for parameters α and β was performed using the tool JSON2RUN (Urli 2013), which uses an

Algorithm 4 Multi-start greedy algorithm

```
1: $z(\sigma^*) \leftarrow \infty$;
2: while time limit not exceeded do
3:   while UNSERVEDCOSTUMERS() do
4:     for all $i \in \mathcal{I}$ do
5:       if RESIDUALAMOUNT($i$)>0 then
6:         best_cost $\leftarrow \infty$;
7:         for all $j \in \mathcal{J}$ do
8:           if RESIDUALCAPACITY($j$)>0 and COMPATIBLE($i$, $j$) then
9:             if LOAD($j$)>0 then
10:              amortized_fixed_cost $\leftarrow 0$
11:            else
12:              amortized_fixed_cost $\leftarrow \alpha \cdot f_j \cdot \frac{\text{RESIDUALCAPACITY(j)}}{s_j}$
13:            end if
14:            cost $\leftarrow c_{ij}$ + amortized_fixed_cost
15:            if cost < best_cost then
16:              best_cost $\leftarrow$ cost, best_c $\leftarrow i$, best_w $\leftarrow j$
17:            else
18:              if cost < best_cost + $\beta$ and RANDOMBREAKTIE() then
19:                best_cost $\leftarrow$ cost, best_c $\leftarrow i$, best_w $\leftarrow j$
20:              end if
21:            end if
22:          end if
23:        end for
24:        q $\leftarrow$ min(RESIDUALAMOUNT(best_c),RESIDUALCAPACITY(best_w))
25:          $\sigma \leftarrow$ ASSIGN(best_c,best_w,q)
26:      end if
27:    end for
28:  end while
29:  if $z(\sigma) < z(\sigma^*)$ then
30:    $\sigma^* \leftarrow \sigma$
31:  end if
32: end while
33: return $\sigma^*$
```

F-Race procedure (Birattari et al. 2010) for selecting the best configuration. The winning configuration turned out to be $\alpha = 0.25$ and $\beta = 0.288$.

5 Experimental results

We now describe the experimental results. We first introduce the datasets used and the other computational settings (Sect. 5.1), and then we present and discuss the results (Sect. 5.2).

5.1 Settings

The search methods have been tested on artificial instances created by a generator specifically designed for this purpose. Instances have been partitioned into two datasets: the training instances used for tuning the parameters and the validation instances used for the comparison. Instances are available at the MESS 2020+1 website https://www.ants-lab.it/mess2020/#competition and at https://github.com/MESS-2020-1, along with the solution validator used to check the correctness of the results.

Table 2 Features of the training instances

Instance	J	I	SI	AOC	ASC	DR
wlp01	50	115	383	648.00	52.4447	0.451
wlp02	100	253	1718	678.50	53.4662	0.484
wlp03	150	345	3447	671.73	52.2599	0.440
wlp04	200	479	6292	674.80	53.5689	0.446
wlp05	250	601	9750	664.60	53.7353	0.472
wlp06	300	705	13, 701	640.50	52.6482	0.452
wlp07	400	1012	27, 635	649.85	53.0710	0.499
wlp08	500	1277	43, 632	629.38	52.7900	0.489
wlp09	600	1483	59, 477	646.70	53.0887	0.480
wlp10	700	1733	83, 291	647.23	52.7992	0.475
wlp11	800	2020	109, 668	640.52	52.8867	0.491
wlp12	900	2159	126, 847	653.07	52.9303	0.453
wlp13	1000	2305	142, 457	633.63	52.9190	0.446
wlp14	1200	2927	232, 119	650.87	52.7535	0.471
wlp15	1400	3445	320, 634	645.94	52.8993	0.474
wlp16	1600	4067	450, 067	640.66	52.7801	0.490
wlp17	1800	4373	520, 406	649.83	52.9662	0.464
wlp18	2000	4908	657, 679	647.71	52.3566	0.473
wlp19	2500	5882	927, 868	650.94	52.8090	0.453
wlp20	3000	7800	1, 653, 786	655.39	52.6730	0.496

Table 3 Features of the validation instances

Instance	J	I	SI	AOC	ASC	DR
wlp21	75	172	879	618.27	51.1787	0.471
wlp22	175	428	4744	642.17	53.4794	0.461
wlp23	275	694	13, 197	625.78	52.8361	0.516
wlp24	450	1128	34, 546	667.69	52.4473	0.464
wlp25	650	1619	71, 398	637.80	52.3215	0.482
wlp26	850	2007	109, 325	656.33	52.5074	0.461
wlp27	1100	2847	224, 656	652.87	52.0253	0.495
wlp28	1500	3474	323, 388	647.10	52.7387	0.445
wlp29	1900	4522	556, 567	649.75	52.4321	0.461
wlp30	2750	6965	1, 335, 044	647.70	52.4357	0.487

The features of the instances are shown in Tables 2 and 3, where J and I are the numbers of facilities and customers, SI is the number of incompatibilities, AOC is the average opening cost, ASC is the average supply cost, and DR is the ratio between the total demand and the total capacity.

All the experiments were run on an AMD Ryzen Threadripper PRO 3975WX 32-Cores (3.50 GHz) with Ubuntu Linux 20.4. One single core was dedicated to each experiment.

We make comparisons based on two different timeouts, both based on the number of facilities J. The first is the one proposed for the MESS 2020+1 competition, which grants $10\sqrt{J}$ seconds for solving each instance, whereas the second grows linearly, granting exactly J seconds per instance.

5.2 Comparison results

The results are shown in Tables 4 and 5 for the timeouts $10\sqrt{J}$ and J, respectively. Each solution method was run for ten repetitions for each instance, collecting the best (minimum) and average values of the objective function. Values in boldface are the best average ones among the different techniques. Best known values are highlighted in underlines.

For completeness, the tables include results on both training (wlp01–wlp20) and validation instances (wlp21–wlp30).

We notice that we have only a marginal improvement by moving from the competition's timeout to the linear one. More precisely, the improvement is only 0.5% in general.

The results shown in Tables 4 and 5 clearly set forth the fact that the best technique is MR-MS-ILS. In order to have a more quantitative measure of the respective positioning, in Table 6 we show the average ranks throughout all runs (10×4) upon all instances. In detail, each column shows the average ranks obtained by the solution techniques in ten runs, distinct for each combination of dataset and timeout. To give an intuitive interpretation of these numbers, we mention that if a search method had been better than all the others in all runs, it would have obtained the ranks 1, 2, …, 10 for its ten runs, resulting in an average rank of 5.5 (as the average of the numbers 1, 2, …, 10).

Table 6 highlights that there is no remarkable difference between the columns, showing that the ranks are not significantly affected by the specific dataset and the timeout. It also confirms that MR-MS-ILS is indisputably superior, with MG and GRASP following with similar results among them, and PcEA coming last.

In order to give a better insight into the behavior of the proposed techniques, in Fig. 2 we show the average time needed to reach the best solution in comparison with the allotted time (red line above) for the validation instances with the second timeout. We see that such time varies considerably depending on the technique and the size of the instance.

6 Conclusions

We have proposed a portfolio of metaheuristic techniques for solving the multi-source capacitated facility location problem with customer incompatibilities, which consists in selecting the facilities to be open and the shipping plan, with an additional constraint about incompatibilities between pairs of customers that cannot be served by the same facility.

To solve this problem, we designed four metaheuristic methods developed autonomously by different research

Table 4 Comparative results on training and validation instances with the competition timeout equal to $10\sqrt{J}$ seconds

Inst.	MR-MS-ILS		GRASP		PcEA		MG	
	min	avg	min	avg	min	avg	min	avg
wlp01	28,978	**29,092.3**	30,040	30,195.4	29,006	29,736.8	34,377	34,377.0
wlp02	54,493	**54,882.4**	56,492	56,821.7	56,925	57,789.6	59,933	60,247.3
wlp03	66,927	**67,683.9**	69,027	69,450.5	73,027	74,116.9	73,349	73,680.5
wlp04	89,857	**90,207.1**	92,455	93,042.2	98,576	100,104.5	98,367	98,615.2
wlp05	111,627	**112,236.1**	113,694	114,304.7	123,293	126,425.7	115,846	117,354.7
wlp06	118,681	**119,549.1**	120,797	121,942.1	133,108	137,503.5	126,265	127,688.3
wlp07	176,631	**177,599.6**	180,191	181,591.0	199,563	202,793.1	183,670	184,205.8
wlp08	204,862	**206,574.3**	212,179	214,415.6	233,202	236,736.1	211,878	212,843.8
wlp09	240,299	**241,249.7**	250,934	252,634.8	273,294	277,149.3	246,270	246,917.2
wlp10	264,252	**265,507.4**	282,518	285,182.7	308,317	314,341.1	275,649	276,605
wlp11	315,760	**317,057.7**	329,788	332,845.5	362,415	366,547.8	322,269	323,449.5
wlp12	323,993	**326,631.9**	344,734	347,702.0	374,867	379,300.5	332,749	333,909.7
wlp13	343,793	**345,926.4**	371,068	372,925.6	396,380	401,975.9	349,964	351,697.2
wlp14	431,981	**432,921.9**	468,742	470,702.7	498,599	502,115.0	436,872	438,916.3
wlp15	499,596	**505,033.1**	540,567	544,737.7	576,254	579,838.0	501,671	505,401.4
wlp16	579,364	583,798.1	615,768	619,139.1	658,583	669,751.4	581,757	**583,453.1**
wlp17	605,310	**606,812.2**	655,090	662,262.5	702,080	710,160.3	617,832	619,390
wlp18	677,396	**680,067.6**	736,207	740,016.9	774,778	783,993.2	687,117	688,304.8
wlp19	807,447	815,478.9	876,105	880,930.1	937,785	946,733.5	807,573	**809,986.9**
wlp20	1,043,150	**1,048,531.0**	1,140,090	1,142,525.0	1,192,820	1,198,344.0	1,050,020	1,053,036
wlp21	38,474	**38,653.0**	39,892	40,282.8	39,233	40,466.1	44,298	44,443.7
wlp22	79,378	**79,746.2**	79,348	80,078.4	84,301	86,434.6	85,732	86,462.4
wlp23	127,873	**128,307.5**	129,719	130,669.6	143,077	144,204.9	134,271	135,725.6
wlp24	182,248	**182,922.0**	191,882	193,015.9	209,596	212,898.2	191,017	192,262.3
wlp25	247,975	**251,053.7**	264,454	267,853.5	289,981	292,921.6	257,904	258,974.2
wlp26	322,663	324,775.3	334,346	336,539.9	364,831	369,576.6	322,085	**323,866**
wlp27	421,190	**422,517.2**	457,577	460,811.3	481,565	488,519.6	430,559	431,642.5
wlp28	493,849	**497,937.1**	536,825	543,859.7	566,881	573,423.6	502,019	504,837.7
wlp29	658,764	665,442.3	689,560	697,430.5	743,620	754,246.0	648,531	**650,840.2**
wlp30	943,009	949,442.8	1,014,500	1,017,282.0	1,072,730	1,076,453.0	940,479	**942,920.2**
Avg	349,994.0	352,254.6	374,153.0	376,706.4	399,956.2	404,486.7	355,677.4	357,068.5

groups, participating in the competition of the Metaheuristics Summer School (MESS 2020+1).

We performed an extensive experimental analysis by using a training dataset for tuning the algorithms and a validation dataset to compare and rank the methods. The benchmark datasets and the solution checker are publicly available on the MESS 2020+1 website. Results are shown for two different timeouts and for both datasets. The outcome is a clear ranking independent of the timeout and the dataset.

In the future, we plan to apply our search methods to other versions of the facility location problem, including different constraints and objectives. In addition, we could test our methods on other datasets with different sizes and structures and compare with state-of-the-art results.

Possible directions for improvements lie in the hybridization of the different methods. Indeed, some of the components from one method could be used profitably inside the others—for example, the greedy algorithm as the initial solution selection for the other methods.

Table 5 Comparative results on training and validation instances with a timeout equal to J seconds

Inst.	MR-MS-ILS		GRASP		PcEA		MG	
	min	avg	min	avg	min	avg	min	avg
wlp01	28,913	**29,115.8**	30,041	30,149.5	29,455	29,754	34,377	34,377
wlp02	54,337	**54,802.9**	56,043	56,725.3	56,703	57,847.9	60,055	60,347.9
wlp03	67,266	**67,540.8**	68,609	69,476.1	70,796	72,662.8	73,064	73,677.8
wlp04	89,544	**90,060.3**	92,301	92,797.8	97,317	99,675.1	97,624	98,247.5
wlp05	111,640	**112,175.6**	112,795	113,542.8	120,270	123,091.1	116,841	117,319.3
wlp06	119,049	**119,613.1**	120,892	121,759.5	134,270	135,687.4	126,802	127,334.7
wlp07	176,044	**177,044.3**	178,863	180,693.6	195,751	198,910.9	183,445	183,797.8
wlp08	203,358	**205,710.1**	212,670	213,795.5	227,613	232,665.3	211,885	212,515.1
wlp09	239,174	**240,312.5**	248,099	251,159.7	271,341	273,808.6	245,345	245,922.1
wlp10	263,295	**264,925**	281,576	283,672.7	301,321	309,782.7	274,494	275,508.5
wlp11	313,229	**315,728.7**	327,830	331,875.2	355,126	360,683.7	321,833	322,804.4
wlp12	324,766	**325,507.1**	341,030	344,437	369,333	375,447.1	332,666	333,413.6
wlp13	341,823	**345,307.9**	366,602	368,888.9	390,328	396,320.3	349,548	350,811.3
wlp14	429,168	**430,919.1**	462,335	468,817.1	491,340	496,191.2	436,583	438,183.4
wlp15	497,061	**501,122.4**	538,535	541,858.4	570,659	575,833.8	502,617	504,380.3
wlp16	578,979	**581,756.9**	613,451	617,507.9	658,618	664,376	580,481	581,807.7
wlp17	603,288	**604,889**	654,713	660,503.1	697,541	704,478.4	616,041	617,354.1
wlp18	674,421	**677,531.8**	734,574	739,435.7	778,266	782,867.4	683,724	686,295.7
wlp19	805,663	808,405.2	873,218	878,551.5	920,158	941,345.1	805,365	**807,075.4**
wlp20	1,039,400	**1,041,971**	1,137,770	1,142,545	1,173,020	1,197,136	1,044,990	1,047,811
wlp21	38,420	**38,567.3**	39,978	40,214.7	39,781	40,600.8	44,175	44,391.6
wlp22	78,813	**79,336.6**	79,599	80,138.5	83,092	84,975.6	85,964	86,273.8
wlp23	127,076	**128,028.2**	129,527	130,163.8	139,348	142,287.9	133,931	135,290.9
wlp24	181,199	**182,220.6**	189,718	191,675	207,508	209,439	190,960	192,024.5
wlp25	249,547	**250,928.1**	265,817	267,378.8	283,119	288,489.6	257,796	258,596.2
wlp26	321,935	**323,134**	331,780	333,906.5	358,093	365,582.5	322,082	323,315.8
wlp27	418,930	**420,459.1**	451,413	455,969.9	477,195	481,836.7	428,214	430,329
wlp28	493,746	**495,644.8**	535,637	540,882.2	568,640	573,048.9	502,483	503,812.5
wlp29	656,512	659,751.1	694,579	697,376.3	734,162	744,444.2	646,626	**648,973.2**
wlp30	940,179	942,409.2	1,012,550	1,016,481	1,066,990	1,072,834	936,728	**939,816.9**
Avg	348,892.5	350,497.3	372,751.5	375,412.6	395,571.8	401,070.1	354,891.3	356,060.3

Table 6 Average ranks obtained by the solvers for 10 repetitions for each instance

Dataset	Training		Validation	
Timeout	Competition	Linear	Competition	Linear
MR-MS-ILS	6.59	6.12	8.43	7.81
GRASP	22.34	20.43	22.41	22.1
PcEA	33.81	33.5	33.59	33.41
MG	19.27	19.96	17.57	18.68

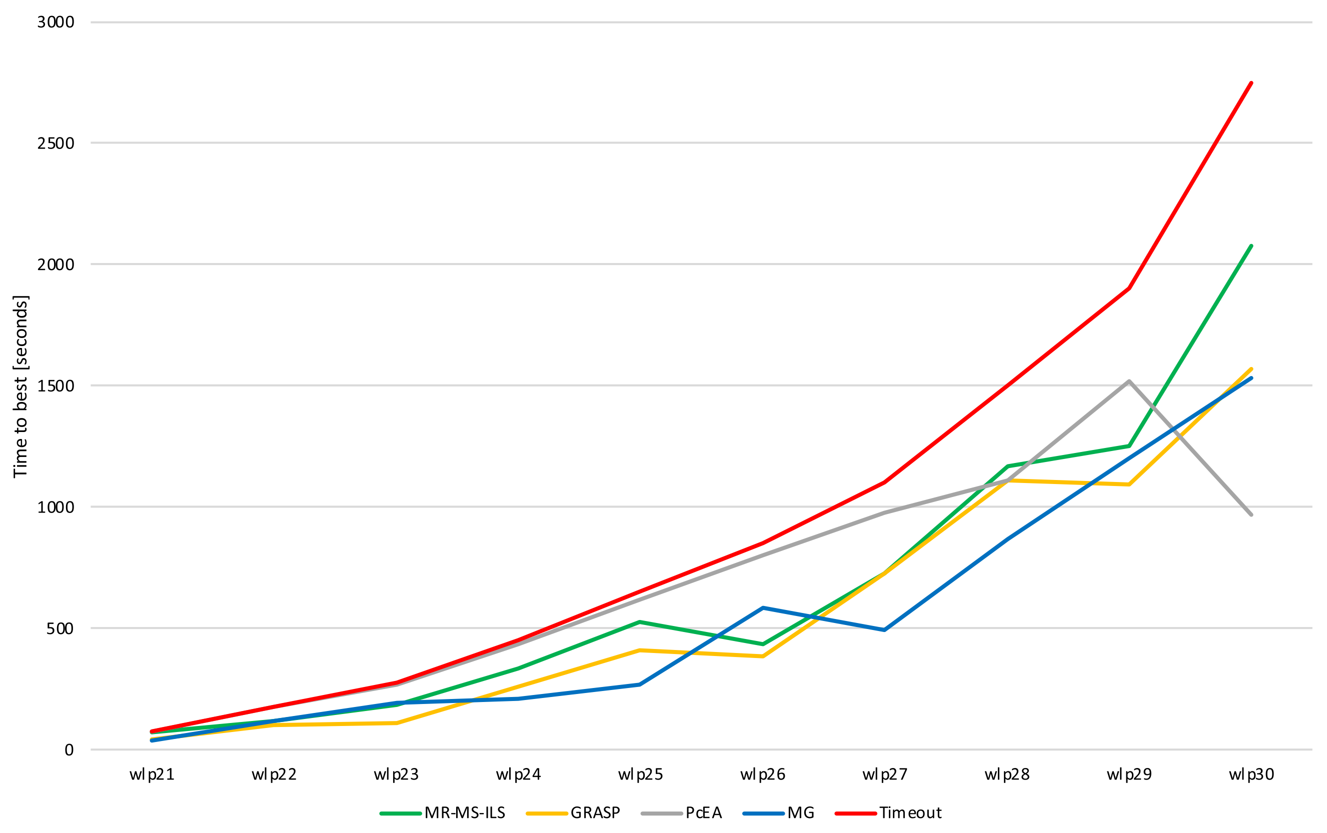

Fig. 2 Time to converge to the best solution for validation instances with timeout equal to J

Author Contributions MRHM designed the MR-MS-ILS algorithm under the supervision of AP and USS. MR and CP-T designed the GRASP algorithm. PPV designed the PcEA algorithm. SC, MP and AS formulated the problem definition, generated the benchmark datasets, designed the MG algorithm and conducted the experiments. The manuscript was jointly written by all authors, who read and approved its final version.

Funding Marcelo R. H. Maia has received research support from the Brazilian Institute of Geography and Statistics (IBGE, Brazil). Alexandre Plastino and Uéverton S. Souza have received research support from Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq, Brazil) (Grant Numbers 310444/2018-7 and 309832/2020-9). Uéverton S. Souza has received research support from Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ, Brazil) (Grant Number E-26/201.344/2021).

Data availability The datasets generated during the current study are available at the MESS 2020+1 website, https://www.ants-lab.it/mess2020/#competition, and at https://github.com/MESS-2020-1.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

References

Avella P, Boccia M (2009) A cutting plane algorithm for the capacitated facility location problem. Comput Optim Appl 43(1):39–65

Avella P, Boccia M, Sforza A et al (2009) An effective heuristic for large-scale capacitated facility location problems. J Heuristics 15(6):597

Avella P, Boccia M, Mattia S et al (2021) Weak flow cover inequalities for the capacitated facility location problem. Eur J Oper Res 289(2):485–494

Barahona F, Chudak FA (2005) Near-optimal solutions to large-scale facility location problems. Discrete Optim 2(1):35–50

Birattari M, Yuan Z, Balaprakash P et al (2010) F-race and iterated F-race: an overview. In: Experimental methods for the analysis of optimization algorithms. Springer, Berlin, pp 311–336

Caserta M, Voß S (2020) A general corridor method-based approach for capacitated facility location. Int J Prod Res 58(13):3855–3880

Cornuéjols G, Nemhauser G, Wolsey L (1983) The uncapacitated facility location problem. Technical report, Cornell University Operations Research and Industrial Engineering

Davis LD (1991) Handbook of genetic algorithms. Van Nostrand Reinhold, New York

Fernández E, Landete M (2015) Fixed-charge facility location problems. In: Location science. Springer, pp 47–77

Festa P, Resende MG (2018) GRASP. In: Marti R, Pardalos P, Resende M (eds) Handbook of heuristics. Springer, pp 465–488

Fischetti M, Ljubić I, Sinnl M (2016) Benders decomposition without separability: a computational study for capacitated facility location problems. Eur J Oper Res 253(3):557–569

Görtz S, Klose A (2012) A simple but usually fast branch-and-bound algorithm for the capacitated facility location problem. INFORMS J Comput 24(4):597–610

Guastaroba G, Speranza MG (2012) Kernel search for the capacitated facility location problem. J Heuristics 18(6):877–917

Ho SC (2015) An iterated tabu search heuristic for the single source capacitated facility location problem. Appl Soft Comput 27:169–178

Klose A, Drexl A (2005) Facility location models for distribution system design. Eur J Oper Res 162(1):4–29

Larrañaga P, Kuilpers C, Poza M et al (1997) Decomposing Bayesian networks: triangulation of the moral graph with genetic algorithms. Stat Comput 7:19–34

López-Ibáñez M, Dubois-Lacoste J, Pérez Cáceres L et al (2016) The irace package: iterated racing for automatic algorithm configuration. Oper Res Perspect 3:43–58

Lourenço HR, Martin OC, Stützle T (2003) Iterated local search. Springer, Boston, pp 320–353

Maia MRH, Plastino A, Penna PHV (2020) MineReduce: an approach based on data mining for problem size reduction. Comput Oper Res 122(104):995

Marín A, Pelegrín M (2019) Adding incompatibilities to the simple plant location problem: formulation, facets and computational experience. Comput Oper Res 104:174–190

Melo MT, Nickel S, Saldanha-Da-Gama F (2009) Facility location and supply chain management—a review. Eur J Oper Res 196(2):401–412

Nethercote N, Stuckey PJ, Becket R et al (2007) MiniZinc: towards a standard CP modelling language. In: Bessière C (ed) CP 2007, LNCS, vol 4741. Springer, pp 529-543

Rabbani M, Heidari R, Farrokhi-Asl H et al (2018) Using metaheuristic algorithms to solve a multi-objective industrial hazardous waste location-routing problem considering incompatible waste types. J Clean Prod 170:227–241

Rios E, Coelho IM, Ochi LS et al (2016) A benchmark on multi improvement neighborhood search strategies in CPU/GPU systems. In: 2016 international symposium on computer architecture and high performance computing workshops (SBAC-PADW), pp 49–54

Silva JCN, Coelho IM, Souza US et al (2022) Finding the maximum multi improvement on neighborhood exploration. Optim Lett 16:97–115

Urli T (2013) json2run: a tool for experiment design & analysis. CoRR arXiv:1305.1112