Les Cahiers du GERAD ISSN: 0711–2440

Heuristics for the dynamic facility location problem with modular capacities

A. Silva, D. Aloise,
L. C. Coelho, C. Rocha

G–2019–29

April 2019

Heuristics for the dynamic facility location problem with modular capacities

Allyson Silva [a]

Daniel Aloise [b]

Leandro C. Coelho [c]

Caroline Rocha [d]

[a] *CIRRELT & Faculté des Sciences de l'Administration, Université Laval (Québec) Canada*

[b] *GERAD & Department of Computer and Software Engineering, Polytechnique Montréal, Montréal (Québec) Canada*

[c] *Canada Research Chair in Integrated Logistics, Université Laval, Québec (Québec) Canada*

[d] *GERAD & Escola de Ciências e Tecnologia, Universidade Federal do Rio Grande do Norte, Natal, RN, Brazil*

allyson.silva@cirrelt.ca
daniel.aloise@gerad.ca
leandro.coelho@cirrelt.ca
caroline.rocha@ect.ufrn.br

April 2019
Les Cahiers du GERAD
G–2019–29

Abstract: This paper studies the Dynamic Facility Location Problem with Modular Capacities (DFLPM). We propose a linear relaxation based heuristic (LRH) and an evolutionary heuristic that hybridizes a genetic algorithm with a variable neighborhood descent (GA+VND) to solve it. The problem is a generalization of several facility location problems and consists in determining locations and sizes of facilities to minimize location and demand allocation costs with decisions taken periodically over a planning horizon. The DFLPM is solved using two heuristics tailored for different network configurations and cost structures. Experiments are reported comparing them to a state-of-the-art mixed integer programming (MIP) formulation for the problem from the literature solved by CPLEX. For the existing benchmark instances, the solution generated by LRH improved by VND finds solutions within 0.03% of the optimal ones in less than half of the computation time of the state-of-the-art method from the literature. In order to yield a better representation of real-life scenarios, we introduce a new set of instances for which GA+VND proved to be an effective approach to solve the problem, outperforming both CPLEX and LRH methods.

Keywords: Location, modular capacity, hybrid metaheuristic, genetic algorithm, variable neighborhood descent

Acknowledgments: The authors thank Sanjay Jena for kindly providing all data and support needed to compare to his work. This work was partly supported by the Brazilian *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES) and by the Canadian Natural Sciences and Engineering Research Council under grants 2017–05617 and 2014–05764. This support is gratefully acknowledged.

1 Introduction

Facility location is an important strategic decision which impacts the whole supply chain. Facilities have to be installed to support the operation of a distribution network in order to meet customer demands. To efficiently determine where to locate facilities, decision makers should balance allocation and location costs, usually associated with transporting goods towards demand points and construction and operating costs of facilities, respectively.

Facility Location Problems (FLPs) cover a well-established body of research with a wide range of applications in many economic sectors (see Arabani and Farahani [4], Klose and Drexl [27] and Owen and Daskin [34] for reviews). The common decisions related to FLPs are *where* to locate facilities among the potential sites where they could be installed, and *how* to allocate demands to them. Furthermore, other properties can be considered, e.g., capacitated FLPs consider a limit on the maximum demand they can satisfy.

A variant of the problem allows capacity dimensioning at facilities [38, 40]. In this case, one also decides *what* is the capacity installed at an open facility. Capacity levels are commonly defined in discrete intervals, like in a staircase function [15]. Models that consider such feature represent capacities as modular structures [3]. A module is a block of capacity associated with an installation cost. The sum of the capacities of all modules installed on a site defines the total capacity of that facility. Modular facilities increase the flexibility of the network substantially when compared to centralized production in large facilities [8]. Dynamic, or multi-period, FLPs consider that costs or demands change over the planning horizon [10, 13], reviewing facility location decisions periodically in order to adapt the distribution network to the new parameters, improving demand satisfaction [22, 32] and decreasing costs [35]. Thus, for dynamic problems one also has to determine *when* to open, close or move a facility.

These features give rise to the Dynamic Facility Location Problem with Modular Capacities (DFLPM), as introduced by Jena et al. [23, 24]. The authors modelled the problem as a mixed-integer programming (MIP) formulation allowing several ways to adjust facility capacities within a planning horizon by opening and closing modules. They use a detailed cost structure based on a matrix describing the costs for capacity changes between the pairs of modules. The model was designed inspired by an application in the forestry industry [24].

Capacitated location problems and their variants belong to the class of NP-hard problems [12], which means there is no known polynomial time algorithm to solve them to optimality. Several heuristics have been developed to solve them. Recent and relevant ones are reviewed next. An et al. [2] present relax-and-round heuristics. Arya et al. [6] and Chudak and Williamson [11] adapt local search heuristics for different versions of such problem. The heuristic of Zhang et al. [44] allows to simulteneously exchange multiple facilities in order to improve solutions. Lagrangean heuristics are used to find bounds in Jena et al. [25] to a capacitated, dynamic and multi-commodity FLP, and in Rafie-Majd et al. [36] to a multi-level inventory-location-routing problem. Lee and Dong [31] combine Lagrangean relaxation and Benders decomposition to create a cross decomposition technique. Several metaheuristics were also proposed to solve FLPs. Arostegui et al. [5] implement and compare a tabu search, a simulated annealing and a genetic algorithm (GA) to solve different FLPs. Basu et al. [7] review other heuristics such as particle swarm optimization and scatter search. Greedy Randomized Adaptive Search Procedure (GRASP) [14] and variable neighborhood search (VNS) [42] are also proposed in literature. The number of published methods is an indication that the algorithm performance is situational, thus the choice of the most appropriate one depends on the characteristics of the problem. When one procedure does not provide efficient solutions for specific cases, authors exploit the hybridization of methods. Wollenweber [42] hybridizes a linear relaxation with a VNS with several local searches to improve the solution. Fernandes et al. [18] use a similar structure but with a GA to diversify the solutions.

In this paper, we consider the DFLPM as presented in Jena et al. [23] and develop a linear relaxation based heuristic (LRH) and an evolutionary algorithm as alternative methods to the state-of-the-art MIP to solve the problem. The first heuristic generates a feasible solution for the problem from the linear relaxation of the MIP formulation. The second algorithm hybridizes a GA structure with periodical applications of a variable neighborhood descent (VND) applied to solutions selected from the GA population. The neighborhoods explored within our VND are designed by extending successful local searches applied to simpler FLPs and constitute another contribution of this work. The performance of all local searches are tested by performing experiments with combinations of them for randomly generated solutions. We show that the use of all local searches defined in a VND framework is very effective for this problem. We solve benchmark instances using LRH and compare it to the state-of-the-art method from the literature. The instances are then adapted to better represent facility location costs in real-life scenarios. Finally, we show that our GA+VND is better than the other approaches for these new instances.

This paper is organized as follows. In Section 2 the problem is described and detailed by presenting its mathematical model. Section 3 introduces the linear relaxation heuristic and how a solution generated by the heuristics is evaluated. In Section 4 we describe the VND method used in this work. The hybrid evolutionary heuristic is described in Section 5. Section 6 presents the experiments to tune the algorithm and the comparison against the exact method from the literature. Finally, our conclusions are given in Section 7.

2 Problem description and mathematical model

The DFLPM is a discrete network location problem [16] derived from median problems. This means that demands exist and facilities should be located on a network of discrete nodes and arcs. The discrete points where facilities can be located are called *candidates* [37]. The problem can also be classified as a location-allocation problem, in which customers can be served by multiple facilities [33]. Facilities have capacity constraints and are located on a single stage of the network [27, 33]. Capacities can be dimensioned from a domain defined in discrete points, also called modules. Capacity expansions are made via a finite set of projects, and modules are represented as capacity blocks [23]. The problem is also a dynamic, or multi-period, location problem, where decisions are made in discrete periods [30, 34].

Let graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ represent the network where $\mathcal{V}$ is the node set and $\mathcal{A}$ is the arc set. The set $\mathcal{I} \subset \mathcal{V}$ represents the customers and $\mathcal{J} \subset \mathcal{V}$ represents the candidates. Directed arcs represent the links between candidate and customer nodes. Let L be the maximum number of modules that can be installed in $j \in \mathcal{J}$, and $\mathcal{L} = \{0, \ldots, L\}$ be the set of these modules. A facility j is said to be operating with module l when all modules up to l are installed in j. The set $\mathcal{T} = \{1, \ldots, T\}$ represents the periods of a planning horizon when location decisions are made. Finally, consider that the beginning of the time interval represented by the period $t + 1$ corresponds to the end of the time interval represented by the period t.

The problem is composed of the following parameters. Customer i has a demand d_i^t in period t. The unit cost to serve customer i from facility j operating with module l in period t is represented by c_{ijl}^t. A facility j operating with module l has capacity u_{jl}. The parameter $e_{jl_1l_2}^t$ represents the cost matrix of facility j operating with module l_1 in period $t-1$ changing to module l_2 in period t. We consider that the functions defining costs e and capacities u are monotonically increasing in l. A facility operating with module $l = 0$ has capacity $u_{j0} = 0$ for all candidates j. Finally, l^j represents the module installed at j at the beginning of the planning horizon.

Two types of decisions have to be made: location and allocation decisions. Location decisions are represented by binary decision variables y. Let $y_{jl_1l_2}^t$ be equal to one if facility j changes its module from l_1 to l_2 and operates with l_2 in period t. Note that when $l_1 = l_2$ and $y_{jl_1l_2}^t = 1$, the capacity has not changed in t. Yet, operation costs can occur if $l > 0$. Capacity expansions occur when $y_{jl_1l_2}^t = 1$ for $l_1 < l_2$, and capacity reductions when $l_1 > l_2$. Allocation decisions are represented by x_{ijl}^t, where x represents the fraction of demand of customer i in period t served by facility j with module l. The resulting MIP formulation, as modelled by Jena et al. [23], is defined as follows.

$$\min \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}}\sum_{l\in\mathcal{L}}\sum_{t\in\mathcal{T}} c^t_{ijl} d^t_i x^t_{ijl} + \sum_{j\in\mathcal{J}}\sum_{l_1\in\mathcal{L}}\sum_{l_2\in\mathcal{L}}\sum_{t\in\mathcal{T}} e^t_{jl_1l_2} y^t_{jl_1l_2} \tag{1}$$

subject to

$$\sum_{j\in\mathcal{J}}\sum_{l\in\mathcal{L}} x^t_{ijl} = 1, \quad \forall i \in \mathcal{I}, t \in \mathcal{T} \tag{2}$$

$$\sum_{i\in\mathcal{I}} d^t_i x^t_{ijl} \leq \sum_{l_1\in\mathcal{L}} u_{jl} y^t_{jl_1l}, \quad \forall j \in \mathcal{J}, l \in \mathcal{L}, t \in \mathcal{T} \tag{3}$$

$$\sum_{l_1\in\mathcal{L}} y^{t-1}_{jl_1l} = \sum_{l_2\in\mathcal{L}} y^t_{jll_2}, \quad \forall j \in \mathcal{J}, l \in \mathcal{L}, t \in \mathcal{T} \setminus \{1\} \tag{4}$$

$$\sum_{l_2\in\mathcal{L}} y^1_{jl^jl_2} = 1, \quad \forall j \in \mathcal{J} \tag{5}$$

$$x^t_{ijl} \geq 0, \quad \forall i \in \mathcal{I}, j \in \mathcal{J}, l \in \mathcal{L}, t \in \mathcal{T} \tag{6}$$

$$y^t_{jl_1l_2} \in \mathbb{B}, \quad \forall j \in \mathcal{J}, l_1 \in \mathcal{L}, l_2 \in \mathcal{L}, t \in \mathcal{T}. \tag{7}$$

The objective function (1) minimizes the total costs composed of demand allocation and facility location costs. Constraints (2) ensure that all demands are served in each time interval. Constraints (3) guarantee that demands allocated to the facilities do not violate capacity constraints. Constraints (4) link the variables of module changes in subsequent periods ensuring that at each period a decision is made regarding the next state of the facility. Constraints (5) ensure that one module is chosen to operate at the beginning of the planning horizon. Constraints (6) define the transportation variables domain and (7) the module change variables domain. For a detailed analysis of this formulation, as well as effective valid inequalities that can be used with it, the interested reader is referred to Jena et al. [23].

3 Linear relaxation heuristic

A lower bound for the DFLPM can be obtained by relaxing integrality constraints (7) for variable y in the MIP formulation, transforming it into an easier to solve linear programming model. The solution obtained by the relaxed problem can either be a feasible solution for the original problem or an unfeasible solution in case there exist any non-integer value for y. In the first case, the solution is a guaranteed optimum for the original problem. If the second case happens, the relaxed solution can be used to build a feasible solution for the problem using the following linear relaxation heuristic (LRH).

A typical way to create a feasible solution from the relaxed problem solution in FLPs is using a relax-round approach [2]. When the location variables indicate if a facility is open or closed, the relaxed variables with non-integer values in the linear relaxation solution can be either rounded up to open such facility or rounded down to close it, for example, using the round to the nearest integer approach. On the DFLPM, location variables represent a more complex decision structure. If variables y are rounded to the nearest integer, the solution may violate the constraint that guarantees that at least one decision is made regarding module changes in subsequent time intervals.

Consider an example of a relaxed solution obtained for the module decisions through the planning horizon for a given facility j as shown in Figure 1(a). The arcs in the graph represent the $y^t_{jl_1l_2}$ variables with the value obtained in the relaxed problem solution, where l_1 is the origin node and l_2 the destination node. Each level on the graph represents a period ($t = 1$ to 3) and each node label can be understood as the number of modules that are located in the facility during the time interval represented by that period if an arc connected to it is equal to one. In this example, if a round to the nearest integer approach is used, all variables y are rounded to zero since they are all lower than 0.5. This would violate constraints (5) for facility j since no decision on the first period is made.

A very simple way to prevent all variables being rounded to zero is by guaranteeing that one of them is rounded to one, for example, selecting the highest y on that period. In Figure 1(a) this method results

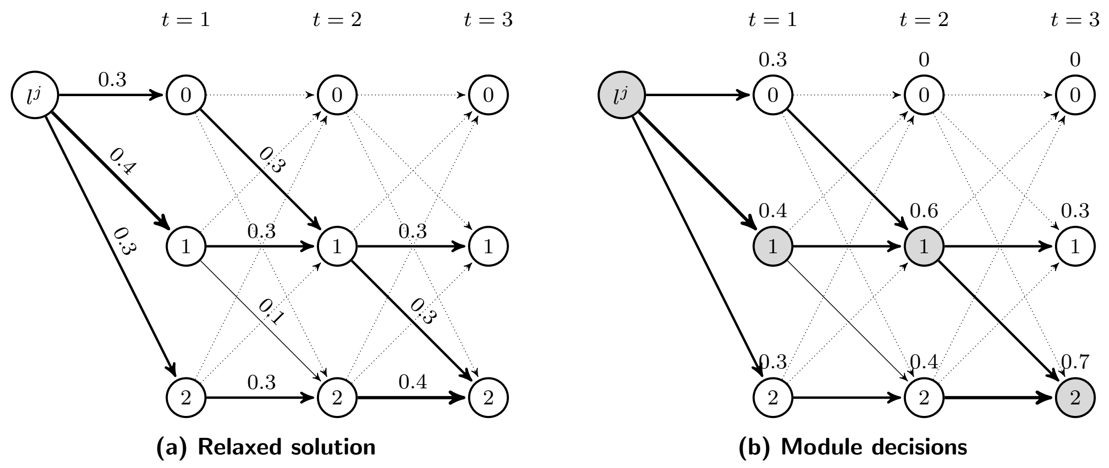

Figure 1: Example of a relaxed solution and the module decisions taken using the LRH

in the choice of the arc $y^1_{jl^j1}$ for $t = 1$, one of the arcs y^2_{j01}, y^2_{j11} and y^2_{j22} for $t = 2$, and the arc y^3_{j22} for $t = 3$. Although constraints (5) are satisfied using this approach, constraints (4) cannot be met regardless on the choice for $t = 2$ since it is impossible to link decisions on subsequent periods using only these arcs. To fix that, we can use the destination nodes of the arcs selected as being the module decision for each time interval and then connecting the proper arcs to them following constraints (4) and (5). So, in the example of Figure 1(a), considering that arcs $y^1_{jl^j1}$, y^2_{j01} and y^3_{j22} are selected, their destination nodes are 1, 1 and 2 for each time interval. Given that these are the module decisions made at each period, we have to activate arcs $y^1_{jl^j1}$, y^2_{j11} and y^3_{j12} to meet the module decision constraints in each period since the destination node of the activated arc in period $t-1$ has to be the origin node of the activated arc in period t. Note that for $t = 3$ the activated arc y^3_{j12} is different than the arc with highest value y^3_{j22} even though this arc was used to select the module for this time interval.

The selection of the destination node of the best arc approach may be undesirable in certain cases. When the module decisions are split between several arcs, the highest arc value may be too low and several arcs may have equal values in such a way that the greedy decision of selecting the highest arc would have several candidates as in $t = 2$ in the example of Figure 1(a). We circumvent this situation by suggesting another approach in which instead of looking only for single arc values, we consider the set of arcs connected to destination nodes. In Figure 1(b), we show the decisions made when considering the sum of the arcs connected to each node. The sum values are shown above the nodes. For each period the chosen node is the one with the highest sum highlighted in the figure. Then, we activate the proper arcs to connect them. We use this selection of the node with the highest sum approach in the LRH to make the decisions on facility location and capacity dimensioning for the DFLPM.

Formally, a solution generated by the LRH is represented by a $J \times T$ matrix $\mathbf{L}$, where $L_{j,t}$ represents the module opened at facility j in period t. Thus $L_{j,t}$ is given as

$$L_{j,t} = \operatorname*{arg\,max}_{l_2 \in \mathcal{L}} \sum_{l_1 \in \mathcal{L}} y^t_{jl_1l_2}. \tag{8}$$

Ignoring the module installed at the beginning of the planning horizon, this matrix can be visualized as being composed by the chosen nodes on the graphs representing the module decisions for each facility, as shown in Figure 1(b).

The approach described can still generate an unfeasible solution due to violation of demand constraints. A simple way to verify the feasibility of a solution is to check whether

$$\sum_{j \in \mathcal{J}} u_{jL_{j,t}} \geq \sum_{i \in \mathcal{I}} d_{it}, \quad \forall t \in \mathcal{T}, \tag{9}$$

is true. If enough capacity is available to meet all demands in each time interval, we can obtain feasible allocations for all customers. Otherwise, more capacity is required in the time intervals with demands that could not be satisfied. In such periods, we open one module in the facility whose next module has the highest sum on Equation (8). Ties are decided randomly. The process is repeated until a feasible solution is obtained. If all modules are open in all facilities and the solution is still unfeasible, we conclude that no feasible solution for the original problem exists.

From the MIP formulation, we note that a solution is evaluated in two parts, i.e., the facility locations and the demand allocations. LRH is designed to determine only facility locations and sizes. Location costs can be calculated straightforwardly from the solution matrix generated by the heuristic. When the facility locations and sizes are known, demand allocations are obtained optimally by solving a minimum-cost flow problem (MCF) [1] for each period, represented by a column in $\mathbf{L}$. The MCF problem can be solved by specialized algorithms with complexity $O(|\mathcal{V}||\mathcal{A}|\log|\mathcal{V}|\log(|\mathcal{V}|C))$, where $|\mathcal{V}|$ is the number of nodes in the network, $|\mathcal{A}|$ is the number of arcs and C is the maximum absolute value of an arc cost [26, 39]. The sum of location costs and each allocation cost is the objective function value of the solution generated by LRH.

4 Local search heuristics

Local search heuristics aim to improve an initial solution by performing a sequence of small changes in its structure and moving it towards a solution with a better objective function each time until a local optimum is found. The neighborhood of a solution x is described as a set of neighbor solutions defined by a neighborhood function $\mathcal{N}$. Thus, every solution in $\mathcal{N}(x)$ is said to be a neighbor of x. The local optimum x^* is such that $x^* \leq x, \forall x \in \mathcal{N}(x^*)$. Several local search heuristics were developed and applied to FLPs. The most effective ones are based on: opening facilities; closing facilities; and swapping open with closed facilities. Studies that use such local searches can be found in Arya et al. [6], Chudak and Williamson [11], Korupolu et al. [28], Zhang et al. [44], among others.

A local search can be created for the DFLPM from operations adapted from the classical local searches used in simpler FLPs. We define two basic operations based on the representation of how long a module stays open at a facility. In dynamic location problems it is common that a module stays open at a facility for a long time span due to the high costs involved in constructing new modules. We represent a module opened at facility j from period t_i to t_f by $S^j_{t_i t_f}$. So, for example, if facility j has only one module opened between periods 1 to 5, the configuration of this facility can be translated to one element S^j_{15} active. In the case that two modules are open between the same periods, then we have two elements S^j_{15} active to represent this facility, and so on. The set representation is an alternative way to represent a solution for the DFLPM. It consists of a set containing all elements active for all facilities in the solution. Consider the following solution x in the matrix representation form as introduced in Section 3:

$$x = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 3 & 2 & 3 & 2 \end{bmatrix}$$

The matrix representation can be converted to the set representation following the next steps. Each line of the matrix represents the configuration of one facility through the planning horizon. On facility 1 (line 1), since there is no open module, no element is active. On facility 2 there is only one module open from periods one to three. Then, the element S^2_{13} is active. Similarly, at facility 3 the element active is S^3_{34} representing the only module opened in that facility. On facility 4 we have four different elements active. The first element S^4_{14} represents module 1, which is open in all four periods. Another element S^4_{14} is active to represent module 2. Module 3 is open in periods 1 and 3. Since there is no continuity on the time it remains open, two elements are active to represent both periods. They are S^4_{11} and S^4_{33}. Hence, solution x can assume the set representation form as $x = \{S^2_{13}, S^3_{34}, S^4_{14}, S^4_{14}, S^4_{11}, S^4_{33}\}$.

We now describe four types of operations performed on a solution. From an initial solution x, we can generate a neighbor solution x' by doing the set operations of union $(\mathrm{Op}(\cup))$ and minus $(\mathrm{Op}(\setminus))$ defined as

Op($\cup$): $x' = x \cup \{S^j_{t_i t_f}\}$: Open one module in facility j from period t_i to t_f in x;

Op($\setminus$): $x' = x \setminus \{S^j_{t_i t_f}\}$: Close one module in facility j from period t_i to t_f in x.

Note that when adding modules to a facility, the elements in the set can change. So, for example, an operation $\{S^j_{13}\} \cup \{S^j_{24}\}$ has to be transformed to $\{S^j_{14}, S^j_{23}\}$ to fit the set representation definition. Similarly, an operation $\{S^j_{24}\} \setminus \{S^j_{33}\}$ change the elements on this facility to $\{S^j_{22}, S^j_{44}\}$ since one module is closed in period 3.

The movement of modules between facilities is represented by a closing followed by an opening of these modules in an origin and a destination facility, respectively. Since this is a recurring operation, we define the movement of modules by the operator Op($\Rightarrow$) as

Op($\Rightarrow$): $x' = x \Rightarrow S^{j^o \Rightarrow j^d}_{t_i t_f} = (x \setminus \{S^{j^o}_{t_i t_f}\}) \cup \{S^{j^d}_{t_i t_f}\}$: Move one module from j^o to j^d from period t_i to t_f in x.

An operation of splitting one element into two complementary elements is also defined. This operation does not modify an existing solution and, therefore, must be used with other operations to generate a new solution. The split operator Op($|$) is defined as

Op($|$): $S^j_{t_i t_f} | c = \{S^j_{t_i c}\} \cup \{S^j_{(c+1) t_f}\}$: Split $S^j_{t_i t_f}$ at time c resulting in $S^j_{t_i c}$ and $S^j_{(c+1) t_f}$.

A countless number of neighborhoods can be defined using different combinations of these four operators. The challenge is to define efficient ones, leading to good local optima without requiring too much processing time. From preliminary experiments we create three local search algorithms for the DFLPM. Table 1 summarizes what types of operations are performed for each one of them.

Table 1: Operations used in the local searches

Local search	*Operations*			
	$\cup$	$\setminus$	$\Rightarrow$	$\vert$
1	✓	✓		
2	✓	✓	✓	✓
3			✓	✓

4.1 Hashing

A local search can be greatly speed up by caching solutions for MCF problems [29]. Given the facilities configuration for a time interval, its MCF solution is always the same since optimal allocations are independent of configurations from other intervals. This is an important characteristic of the DFLPM because it allows the evaluation of new solutions from an initial one without the need to recalculate the allocation costs for all periods individually, but only for those whose facility configurations have changed.

In our algorithm, we save solutions obtained in every run of the MCF algorithm in a hash table. Hash tables are data structures that associate a key to a value. In the table created, the key indicates all facilities configuration and the value is the allocation cost obtained from the MCF. Computationally, the search for an element in a hash table is quickly performed in $O(\log n)$, where n is the number of keys contained in the table. Thus, when the facility configurations repeat during the execution of the algorithm, the optimal allocation cost is easily retrieved from the table. This process is significantly faster than solving the MCF.

4.2 Local search 1: Solutions at a distance of one element

The first local search (LS1) performs all possible moves of opening and closing one module for all combinations of duration from the current solution. The module opening neighborhood is defined as

$$\mathcal{N}^{\cup}(x) = x \cup \{S^j_{t_i t_f}\}, \forall j \in J, t_i = \{0, \dots, T\}, t_f = \{t_i, \dots, T\}.$$

Similarly, the module closing neighborhood is defined as

$$\mathcal{N}^{\setminus}(x) = x \setminus \{S^j_{t_i t_f}\}, \forall j \in J, t_i = \{0, \ldots, T\}, t_f = \{t_i, \ldots, T\}.$$

Thus, we define LS1 neighborhood as

$$\mathcal{N}_1(x) = \mathcal{N}^{\cup}(x) + \mathcal{N}^{\setminus}(x).$$

LS1 procedure is described in Algorithm 1. The algorithm starts the search in a randomly defined facility to avoid bias. The neighborhood $\mathcal{N}^{\setminus}(x)$ is always searched first for a given facility (lines 3–10). Since we use a first improvement strategy, the exploration order matters and we observe that closing modules is a preferable neighborhood to be explored before the opening modules neighborhood, especially because it quickly improves bad initial solutions. If no improvement is found in these neighborhoods, the search continues on another facility. When all facilities have been explored and no improving solution is found, the local optimum is reached.

Algorithm 1 Local search 1

```
1: $x$: initial solution;
2: for all $j \in \mathcal{J}$ do
3:     for $t_i = 1$ to $T$ do
4:         for $t_f = t_i$ to $T$ do
5:             $x' \leftarrow x \setminus \{S^j_{t_i t_f}\}$;
6:             if $f(x') < f(x)$ then
7:                 return $x'$;
8:             end if
9:         end for
10:    end for
11:    for $t_i = 1$ to $T$ do
12:        for $t_f = t_i$ to $T$ do
13:            $x' \leftarrow x \cup \{S^j_{t_i t_f}\}$;
14:            if $f(x') < f(x)$ then
15:                return $x'$;
16:            end if
17:        end for
18:    end for
19: end for
20: return $x$;
```

LS1 neighborhood contains at most $J(T^2 + T)$ solutions. In terms of the MCF algorithm, due to caching the maximum number of calls is $2JT$, i.e., once for each module opened or closed in a facility and a period, whenever possible. That means LS1 explores a neighborhood of size $O(JT^2)$ with a running time of only $O(JT)$.

4.3 Local search 2: Move one module between facilities and slightly change a split element of the set

The second local search (LS2) explores a neighborhood where modules are moved between facilities and a change in their opening and closing periods is performed. Algorithm 2 shows the steps of LS2. Let an element $S^{j^o}_{t_i t_f}$ being contained in the current solution (line 2). LS2 splits it by all possible means, i.e., $S^{j^o}_{t_i t_f}|c, \forall c = \{t_i, \ldots, t_f - 1\}$. For example, from $S^{j^o}_{14}$ we generate the splits $S^{j^o}_{11}$ and $S^{j^o}_{24}$ for $c = 1$, $S^{j^o}_{12}$ and $S^{j^o}_{34}$ for $c = 2$, and $S^{j^o}_{13}$ and $S^{j^o}_{44}$ for $c = 3$. All split elements and the original one constitute the set of elements that are moved by LS2. In the example, the set $\mathcal{T} = \{S^{j^o}_{11}, S^{j^o}_{12}, S^{j^o}_{13}, S^{j^o}_{14}, S^{j^o}_{24}, S^{j^o}_{34}, S^{j^o}_{44}\}$ is generated (line 3). Then, for each element in $\mathcal{T}$, LS2 performs $S^{j^o \Rightarrow j^d}_{t'_i t'_f}$, i.e., moves this element from facility j^o to j^d from periods t'_i to t'_f (line 6). For each element moved, the new solution is evaluated and compared to the current solution (lines 7–8). If the new solution is not better, LS2 performs its last step. This step acts as a very small neighborhood search, containing only eight solutions. They are: early opening (open $S^{j^d}_{(t_i-1)(t_i-1)}$), late opening (close $S^{j^d}_{t_i t_i}$), early closing (close $S^{j^d}_{t_f t_f}$), late closing (open $S^{j^d}_{(t_f+1)(t_f+1)}$), and all four combinations

Algorithm 2 Local search 2

```
1: $\mathcal{S}$: set of active elements in the initial solution $x$;
2: for all $S^{j^o}_{t_i t_f} \in \mathcal{S}$ do
3:     Generate set $\mathcal{T} = \{S^{j^o}_{t_i t_f}\} \cup \{S^{j^o}_{t_i t_f} | c, \forall c = \{t_i, \ldots, t_f - 1\}\}$;
4:     for all $j^d \in \mathcal{J} \setminus \{j^o\}$ do
5:         for all $S^{j^o}_{t'_i t'_f} \in \mathcal{T}$ do
6:             $x' \leftarrow x \Rightarrow S^{j^o \Rightarrow j^d}_{t'_i t'_f}$;
7:             if $f(x') < f(x)$ then
8:                 return $x'$;
9:             else
10:                $x'' \leftarrow x' \cup \{S^{j^d}_{(t'_i-1)(t'_i-1)}\}$, if $f(x'') < f(x)$ then return $x''$;
11:                $x'' \leftarrow x' \setminus \{S^{j^d}_{t'_i t'_i}\}$, if $f(x'') < f(x)$ then return $x''$;
12:                $x'' \leftarrow x' \setminus \{S^{j^d}_{t'_f t'_f}\}$, if $f(x'') < f(x)$ then return $x''$;
13:                $x'' \leftarrow x' \cup \{S^{j^d}_{(t'_f+1)(t'_f+1)}\}$, if $f(x'') < f(x)$ then return $x''$;
14:                $x'' \leftarrow x' \cup \{S^{j^d}_{(t'_i-1)(t'_i-1)}\} \setminus \{S^{j^d}_{t'_i t'_i}\}$, if $f(x'') < f(x)$ then return $x''$;
15:                $x'' \leftarrow x' \cup \{S^{j^d}_{(t'_i-1)(t'_i-1)}\} \setminus \{S^{j^d}_{t'_f t'_f}\}$, if $f(x'') < f(x)$ then return $x''$;
16:                $x'' \leftarrow x' \cup \{S^{j^d}_{(t'_f+1)(t'_f+1)}\} \setminus \{S^{j^d}_{t'_i t'_i}\}$, if $f(x'') < f(x)$ then return $x''$;
17:                $x'' \leftarrow x' \cup \{S^{j^d}_{(t'_f+1)(t'_f+1)}\} \setminus \{S^{j^d}_{t'_i t'_i}\}$, if $f(x'') < f(x)$ then return $x''$;
18:            end if
19:        end for
20:    end for
21: end for
22: return $x$;
```

of opening or closing both extremities simultaneously (lines 10–17). LS2 ends when no improving solution is obtained for all possible moves.

LS2 neighborhood size depends on the length and the number of existing elements in the original solution set of elements and the number of destination facilities, which is usually $J - 1$. Equal elements in a same origin facility generate the same solutions when moved. Similarly, splits that result in the same sets are also redundant. So, what matters is the number of distinct elements that splits are able to create. In the worst case scenario it is $O(T^2)$ for every origin facility J. Therefore, also considering the number of destinations, this neighborhood has size $O(J^2T^2)$ in the worst case. As in LS1, the number of calls to the MCF algorithm is reduced due to the repetition of facilities configurations. It is required only once per move of a module between two facilities. So, to evaluate the $O(T^2)$ distinct elements, it is enough to solve T MCFs. In conclusion, LS2 solves $O(J^2T)$ MCFs to explore $O(J^2T^2)$ neighbor solutions. Since LS2 searches into a neighborhood where modules are moved between facilities, and its complexity is not influenced by the maximum number of modules L in a facility, it is a good local search for instances with large values of L.

4.4 Local search 3: Split and move multiple modules from the same facility

The third local search (LS3) moves multiple modules simultaneously from one facility to another following a scheme with coordinated split operations. LS3 has split, move and return phases. Algorithm 3 details the steps of LS3. It begins selecting an origin facility j^o that has a module open and identifies all its modules using the set representation in S^{j^o} (line 3). Then, for a destination facility j^d LS3 starts the selection of modules to be moved from j^o to j^d using the following rules.

Consider the example in Figure 2 to illustrate LS3 iterations. An origin j^o has the elements $\mathcal{S}^{j^o} = \{S^{j^o}_{14}$, $S^{j^o}_{11}$, $S^{j^o}_{44}\}$ activated, resulting in the matrix representation [2 1 1 2] as shown in the figure. A destination facility j^d is chosen and has the configuration [1 1 0 0]. LS3 split phase sets a time interval starting at time c_1 and ending at time c_2 (lines 5–6). On the first split, the time interval is set to $c_1 = 1$ and $c_2 = 1$, i.e., only modules in period 1 are moved. In Figure 2 the dashed lines represent the split operations done for the elements of j^o. In Algorithm 3 the modules from the splits belong to the set $\mathcal{S}^{j^o}_{c_1c_2}$ (line 7). In its move phase, LS3 applies the move operator for all elements in this set (line 9), moving all modules within the time interval from the origin facility to the destination. Then, the return phase (lines 13–19) iteratively brings the modules back to their original positions. The number of return iterations is equal to the maximum number of modules moved in a period within the time interval. In the example of Figure 2, the maximum number of modules moved is two, so only two iterations are necessary to return the solution to its original setting. The example demonstrates all iterations performed for $c_1 = 1$. Following Algorithm 3, the next iteration would be for $c_1 = 2$ and $c_2 = 2$. After all combinations of splits are checked, the local search changes the destination facility and the search scheme is repeated. LS3 stops when all combinations of origin and destination facilities are explored or any improving solution is found.

$$
\begin{array}{llcccc}
 & & \text{Split} & \text{Move} & \text{Return 1} & \text{Return 2} \\
c_1 = 1, c_2 = 1 & \begin{array}{c} j^o \\ j^d \end{array} & \begin{array}{|c|ccc} 2 & 1 & 1 & 2 \\ 1 & 1 & 0 & 0 \end{array} & \begin{array}{|c|ccc} 0 & 1 & 1 & 2 \\ 3 & 1 & 0 & 0 \end{array} & \begin{array}{|c|ccc} 1 & 1 & 1 & 2 \\ 2 & 1 & 0 & 0 \end{array} & \begin{array}{|c|ccc} 2 & 1 & 1 & 2 \\ 1 & 1 & 0 & 0 \end{array} \\
c_1 = 1, c_2 = 2 & \begin{array}{c} j^o \\ j^d \end{array} & \begin{array}{|cc|cc} 2 & 1 & 1 & 2 \\ 1 & 1 & 0 & 0 \end{array} & \begin{array}{|cc|cc} 0 & 0 & 1 & 2 \\ 3 & 2 & 0 & 0 \end{array} & \begin{array}{|cc|cc} 1 & 1 & 1 & 2 \\ 2 & 1 & 0 & 0 \end{array} & \begin{array}{|cc|cc} 2 & 1 & 1 & 2 \\ 1 & 1 & 0 & 0 \end{array} \\
c_1 = 1, c_2 = 3 & \begin{array}{c} j^o \\ j^d \end{array} & \begin{array}{|ccc|c} 2 & 1 & 1 & 2 \\ 1 & 1 & 0 & 0 \end{array} & \begin{array}{|ccc|c} 0 & 0 & 0 & 2 \\ 3 & 2 & 1 & 0 \end{array} & \begin{array}{|ccc|c} 1 & 1 & 1 & 2 \\ 2 & 1 & 0 & 0 \end{array} & \begin{array}{|ccc|c} 2 & 1 & 1 & 2 \\ 1 & 1 & 0 & 0 \end{array} \\
c_1 = 1, c_2 = 4 & \begin{array}{c} j^o \\ j^d \end{array} & \begin{array}{|cccc|} 2 & 1 & 1 & 2 \\ 1 & 1 & 0 & 0 \end{array} & \begin{array}{|cccc|} 0 & 0 & 0 & 0 \\ 3 & 2 & 1 & 2 \end{array} & \begin{array}{|cccc|} 1 & 1 & 1 & 1 \\ 2 & 1 & 0 & 1 \end{array} & \begin{array}{|cccc|} 2 & 1 & 1 & 2 \\ 1 & 1 & 0 & 0 \end{array}
\end{array}
$$

Figure 2: Example of LS3 iterations

Algorithm 3 Local search 3

```
1: $\mathcal{S}$: set of active elements in the initial solution $x$;
2: for all $j^o \in \mathcal{J}$ do
3:     Create $\mathcal{S}^{j^o} \subset \mathcal{S}$ containing all elements $S^j_{t_i t_f}$ with $j = j^o$;
4:     for all $j^d \in \mathcal{J} \setminus \{j^o\}$ do
5:         for all $c_1 = 1$ to $T$ do
6:             for all $c_2 = c_1$ to $T$ do
7:                 Create set $\mathcal{S}^{j^o}_{c_1 c_2}$ containing all elements from $\mathcal{S}^{j^o}$ resulting from a split at $c_1 - 1$ and then at $c_2$;
8:                 $x' \leftarrow x$;
9:                 $x' \leftarrow x' \Rightarrow S^{j^o \Rightarrow j^d}_{c'_1 c'_2}, \quad \forall S^{j^o}_{c'_1 c'_2} \in \mathcal{S}^{j^o}_{c_1 c_2}$, where $c'_1$ and $c'_2$ represent the initial and final period of each element and
                   are bounded by $c_1$ and $c_2$;
10:                if $f(x') < f(x)$ then
11:                    return $x'$;
12:                end if
13:                while $\mathcal{S}^{j^o}_{c_1 c_2} \neq \emptyset$ do
14:                    $x' \leftarrow x' \Rightarrow \{S^{j^d \Rightarrow j^o}_{c_1 c_2}\}$
15:                    if $f(x') < f(x)$ then
16:                        return $x'$;
17:                    end if
18:                    $\mathcal{S}^{j^o}_{c_1 c_2} \leftarrow \mathcal{S}^{j^o}_{c_1 c_2} \setminus \{S^{j^o}_{c_1 c_2}\}$;
19:                end while
20:            end for
21:        end for
22:    end for
23: end for
24: return $x$;
```

The number of iterations on LS3 in the reverting step is $O(L)$, while the two splits on the split step can generate $O(T^2)$ new elements for each element moved from an origin. Considering yet each pair of facilities origin-destination, the neighborhood size of LS3 is $O(J^2 T^2 L)$. Like for the other local searches, the number of times the MCF is solved is reduced by a factor of T due to caching solutions in the hash table with MCF solutions.

4.5 Variable neighborhood descent

The variable neighborhood descent (VND) is a framework that systematically explores distinct neighborhoods by changing local search algorithms [20]. Hansen and Mladenović [20] state that a local optimum in a neighborhood is not necessarily a local optimum in another, such that the use of several local searches, even with simple neighborhood structures, can lead to high quality solutions. A VND structure is shown in Algorithm 4. There, N neighborhoods are defined and their exploration order is known *a priori* (line 2). Improving solutions replace the current one by exhaustion in each neighborhood. When the first improving solution replaces the current one in lines 4–6, we call *first improvement* strategy. When the whole neighborhood is explored before replacing current solution by the best found one, the strategy is called *best improvement*. The search returns to the first neighborhood when an improving solution is found after the exhaustive search of any other (line 8). Otherwise, it continues the search for the next neighborhood (line 10). The structure presented in Algorithm 4 with first improvement strategy is used in the VND of this work along with the three neighborhoods introduced before.

Algorithm 4 VND basic structure

```
1: Initial solution: $x$;
2: Neighborhoods $\mathcal{N}_n(x)$, $n = \{1, \dots, N\}$;
3: for $n = 1$ to $N$ do
4:     while there is a $x' \in \mathcal{N}_n(x) | f(x') < f(x)$ do
5:         $x \leftarrow x'$;
6:     end while
7:     if $f(x) < f(x_i)$ then
8:         $n = 1$;
9:     else
10:        $n = n + 1$;
11:    end if
12: end for
13: return $x$.
```

5 Hybrid evolutionary heuristic

Metaheuristics are frameworks that sequentially use stochastically chosen heuristics. Metaheuristics often perform better than simple heuristics [43] by alternating diversification and intensification phases during the search for new solutions. In the former, one is interested in finding "different" solutions, i.e., solutions located in unexplored regions of the solutions space. Then, the use of intensification heuristics, such as local searches, helps reach local optima in such a region.

Metaheuristics are classified by Gendreau and Potvin [19] according to the number of solutions considered at a time. Single-solution metaheuristics, such as Simulated Annealing, Tabu Search and Variable Neighborhood Search, consider one solution at a time and have a single search trajectory. In population metaheuristics, such as evolutionary algorithms, multiple solutions evolve concurrently. GA is an example of an evolutionary algorithm.

Compared to other metaheuristics, the convergence in pure GA is slow [5]. A common intensification strategy is the hybridization with local searches applied to solutions in the population. The evolutionary metaheuristic developed in this study uses the structure of a GA as a diversification strategy on the search to generate feasible solutions. Periodically, an intensification phase is applied to one of these solutions by using sequential local search heuristics organized in a VND framework. The resulting algorithm is the GA+VND: a population-based iterated local search heuristic with a VND local search phase.

5.1 Genetic algorithm

GAs are nature-inspired metaheuristics based on the mechanisms of natural selection and evolution. These concepts were popularized by Holland [21] and details of its structure are found in Whitley [41].

Algorithm 5 introduces the pseudocode of a general GA. *Chromosomes*, or individuals, representing solutions for the problem form a *population* $\mathcal{P}$. The *fitness* of each chromosome is evaluated according to the solution's quality, usually measured by its objective function value (line 9). Individuals evolve through successive iterations called *generations*. At every generation, individuals from the current population are combined to generate new individuals forming an *offspring*. The new chromosomes are formed by a *crossover* operator between parents selected by a *selection* operator (lines 4–5). The new solution should have characteristics, or *genes*, inherited from its parents. Furthermore, an offspring can have its genes slightly modified by a *mutation* operator (lines 6–8). A new generation is formed selecting the best chromosomes among the existing ones and the offspring created by crossover using an *acceptance* operator. After successive generations, the solutions tend to converge to better results. The stopping criterion of the algorithm is usually related to a maximum number of generations or search time.

Algorithm 5 General GA structure

```
1: $\mathcal{P}_0$: Initial population created by constructive heuristics in generation $g = 0$;
2: while stopping criteria are not satisfied do
3:     repeat
4:         Selection: Selects parents in $\mathcal{P}_g$;
5:         Crossover: Generates offspring using selected parents;
6:         if mutation condition satisfied then
7:             Mutation: Modifies offspring generated;
8:         end if
9:         Fitness Evaluation: Calculate the fitness of offspring;
10:        Acceptance: Accepts or rejects offspring to $\mathcal{P}_{g+1}$;
11:    until sufficient offspring created
12:    $g \leftarrow g + 1$;
13: end while
14: return the best individual found.
```

5.1.1 Chromosome Representation

In Sections 3 and 4 we presented two ways to represent a solution for the DFLPM. The matrix representation shows the number of modules opened for each facility and each period. The set representation shows the initial and final periods that each module is opened in a given facility. While the set representation is better fit to perform the operations in the local searches presented in Section 4, the matrix representation helps in the understanding and the implementation of the GA operators for this problem. The exception is the mutation operator, which is based on the operations performed on the local searches. In the set representation, a gene is associated to an element in the set, while in the matrix representation a gene is each value of the matrix representing the number of modules opened in each facility at each period.

5.1.2 Initial Population

The population in a generation of GA has a fixed size of P chromosomes. The first P chromosomes created form the initial population of the evolutionary process. These chromosomes are created by heuristic methods. LRH is capable of generating only one solution and therefore cannot be used to generate the whole population. Therefore, we propose two other heuristics to generate feasible solutions. Both heuristics use the set representation of the chromosome to open new modules starting from an empty solution. Constructive Heuristic 1 (CH1) randomly selects one facility j and performs $x \cup \{S^j_{1T}\}$ until a feasible chromosome is reached. Constructive Heuristic 2 (CH2) randomly selects all three parameters (j, t_i and t_f) and performs $x \cup \{S^j_{t_i t_f}\}$ until feasibility is achieved in all periods. It is easy to note that the domain of possible individuals formed by CH1 is a subset of that of the CH2 domain. CH1 creates a solution without any changing in the number of modules in a facility avoiding construction costs in the planning horizon. However, CH2 generates more diversified solutions, which are useful for the crossing over of solutions. CH2 is also used in other parts of the GA+VND algorithm when it is necessary to create new random solutions.

5.1.3 GA Operators

As an evolutionary algorithm, the GA+VND performs crossovers between existing individuals to generate new solutions. The evolutionary process is done in four steps: selection, crossover, mutation and acceptance.

Selection

The selection operator combines family selection and individual selection methods [17]. At each generation, the population is divided into classes based on fitness: A (high), B (middle) and C (low). The number of individuals of each class is controlled by parameters α and β. As in Ericsson et al. [17], the next generation is formed by the automatic promotion of all αP class A individuals, by $(\beta - \alpha)P$ individuals generated by the crossover and the mutation operators and accepted by the acceptance operator, and by $(1-\beta)P$ new random individuals generated by CH2. The proportion of each family size in relation to the population size P is shown in Figure 3. The crossover input is formed by two parents selected from the current population, one from class A and another from any class. At every generation all individuals are sorted by their fitness to determine to which class they belong to.

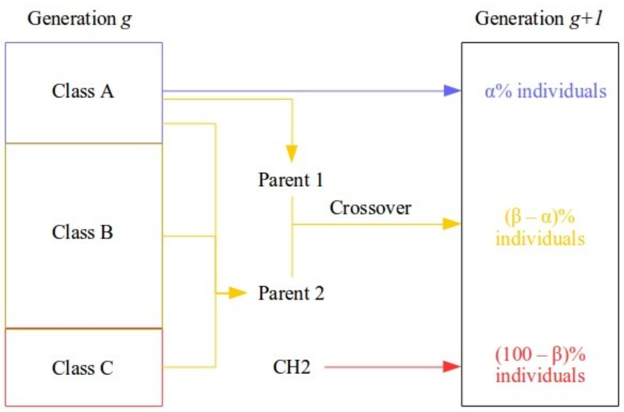

Figure 3: Selection operator and family sizes

Crossover

The crossover operator is performed between the two individuals selected to generate an offspring. As in Fernandes et al. [18], the operator randomly determines for each facility which parental configuration is passed to the new solution. Figure 4 shows the selection process of the genes when facilities 1 and 4 are inherited from parent 1 and facilities 2 and 3 from parent 2.

$$\begin{array}{ccccc} \text{Parent 1} & & \text{Offspring} & & \text{Parent 2} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 2 & 2 & 3 & 2 \end{bmatrix} & \begin{matrix} \rightarrow \\ \\ \\ \rightarrow \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 2 \\ 0 & 1 & 1 & 0 \\ 2 & 2 & 3 & 2 \end{bmatrix} & \begin{matrix} \\ \leftarrow \\ \leftarrow \\ \end{matrix} & \begin{bmatrix} 0 & 0 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 0 & 1 & 1 & 0 \\ 2 & 2 & 1 & 1 \end{bmatrix} \end{array}$$

Figure 4: Crossover Operator

The tendency is that, after some generations, the population becomes homogeneous and stabilizes in a very tight class A. On the one hand, this reduces the time spent in the evaluation process of new individuals thanks to repetitions of facilities configurations. On the other hand, a homogeneous population reduces the probability to escape from local optima. However, the GA+VND has some diversification tools to keep a reasonable level of heterogeneity in the population. One of them is the generation of random individuals that substitutes class C at every generation. Others, as the mutation operator, are explained in the following.

Mutation

In biology, mutation is a process that randomly modifies the genes of an organism. In our GA+VND, the mutation operator combines some of the neighborhoods explored by the local searches to modify a chromosome generated in the crossover. A chromosome has a probability M to be mutated. The operator is designed to move open modules to new locations following these steps:

1. *Draw origin facility*: Select a random origin facility that has open modules, i.e., the set representation of elements in the origin $S^{j^o} \neq \emptyset$;
2. *Draw modules to be moved*: Select the number of elements (genes) in S^{j^o} to be moved with 50% probability of moving only one element and 50% probability of moving more than one element. The number of elements in the latter case is determined randomly in the interval $[2, |S^{j^o}|]$. We draw from S^{j^o} the corresponding number of elements selected and define $\mathcal{M} \subset S^{j^o}$ as the subset containing the selected elements to be moved;
3. *Split decision*: Either all elements in $\mathcal{M}$ are completely or partially closed in j^o, both decisions with equal probability. If partially, a split operator for a random period c is applied to all elements in $\mathcal{M}$. Since the split of an element results in two elements, one ending in c and another beginning in $c+1$, the mutation operator decides if the elements moved are those with final period below c or initial period above $c+1$. The set $\mathcal{M}$ keeps only the elements to be closed in j^o;
4. *Close modules in origin*: Close in j^o the modules in $\mathcal{M}$;
5. *Draw destination facility*: Select a random destination facility $j^d \neq j^o$;
6. *Single element change decision*: Select the element in $\mathcal{M}$ that has the highest time span, i.e., $\max(t_f - t_i)$. As in the first four operations of the last step of LS2, we now decide if we change this element by opening the module one period earlier $(t_i - 1)$ or one period later $(t_i + 1)$, or closing the module one period earlier $(t_f - 1)$ or one period later $(t_f + 1)$. All four options together with the option to keep the time span as is have equal probabilities of being chosen;
7. *Open modules in destination*: Open in j^d the modules in $\mathcal{M}$, considering the single element time span change decision.

The use of predefined criteria instead of random moves allows the mutation to perform potentially good modifications generating new individuals with a reasonable chance of being accepted by the acceptance operator presented next.

Acceptance

The acceptance operator determines whether an offspring created survives for the next generation. The individual is accepted if it meets all the following criteria:

- is feasible;
- has location costs no more than 10% higher than the worst current class A individual;
- and is different than any individual in the current generation or those already accepted to the next one.

The order of verification is important since the evaluation cost increases for the proposed order. The first criterion avoids that infeasible solutions are part of the evolutionary process. The second discards potentially bad solutions before their fitness evaluation. The third prevents identical individuals being part of the same population, which reduces diversity.

The GA operators repeat until $(\beta - \alpha)P$ individuals are accepted. When the process is finished, class A and B individuals are sorted to define the new generation classes. To save time, class C individuals do not have their fitness evaluated, since it is unlikely that CH2 generates class A solutions.

5.2 Elitism via variable neighborhood descent

The elitism is the intensification phase of the hybrid evolutionary heuristic. It consists in applying the VND in one chromosome of the population to improve it to the local optima of the neighborhoods designed. The elitism is applied:

- to the best individual generated in the initial population;
- to a new best solution generated by the GA operators;
- to a random individual that was not in an elitism before from class A or B at every G generations;
- and to a random individual, except the best, after the population mutation phase described next.

The population mutation is a diversification strategy that changes all individuals, except the best one found so far, using the mutation operator at every R generations without improvement in the best solution. The objective is to modify the population, usually slightly worsening the fitness of all individuals, improving the chance of new chromosomes entering class A especially at a point of near convergence of solutions in this class. To increase diversification level over time, the number of mutations performed in each chromosome increases by one every time the population mutation is invoked. The GA+VND stops when no improving solution is found after N cycles.

Algorithm 6 shows how the GA of Algorithm 5 is modified to the application of elitism for the situations where it is applied (lines 3, 10, 12 and 21). Three stopping criteria exist. The time limit, the maximum number of cycles, or the memory limit, whichever happens first (line 4). The last condition is especially important since the use of a hash table to save solutions from the FCM algorithm can consume too much memory for very large instances.

6 Computational experiments

Computational experiments were performed on a single Intel Xeon X5650 processor with a 2.67 GHz clock and 24 GB of RAM. LRH and GA+VND were implemented in C++ and compiled by gcc 5.4.0. The MIP model, and its linear relaxation, were solved using the CPLEX optimization studio version 12.5. The minimum-cost flow problem was implemented and optimized using the CPLEX network optimizer. We enhanced its performance by using a warm start from the previous solution calculated on the network for the same period. The running time was measured using the CPU time obtained by the *time* library of the C language.

We used the benchmarking instances provided by Jena et al. [23] to their ER-GMC formulation. The instances are grouped by size regarding the combinations *facilities/customers* (10/20, 10/50, 50/50, 50/100, 50/250, 100/250, 100/500 and 100/1000) and maximum number of modules L by facility (3, 5 and 10). The number of periods is fixed at ten for all instances. Each group contains 12 instances according to the network topology (three scenarios), the demand distribution (one stable and one unstable scenarios) and the cost proportion (a regular and a 500% raised allocation costs scenarios). A total of 288 original instances were tested with graphs varying from 30 nodes and 200 arcs for the smallest (*10/20*) to 1100 nodes and 100,000 arcs for the biggest (*100/1000*) in each period of the planning horizon.

On the original instances, we observed that module sizes vary according to the number of customers on the instance while location costs are fixed to the number of modules installed on a potential site. We believe this is not a good representation of real cases since module sizes available to construction usually remain the same regardless of the number of customers a company has. An alternative is to consider that module sizes increase but construction costs also increase proportionally maintaining their ratio cost per capacity unit, which is reduced not by the instance size but by economies of scale when locating multiple modules on the same site. So, we adapted Jena et al. [23] instances to consider these cost structures. From the original 288 instances, we considered only those with the regular cost proportion. Then, these 144 instances were modified in two distinct ways. The first fixes the size of each module so that build one module in any instance will add the same capacity no matter how many customers the instance has. So, for example, in the original instances when $L = 3$ the module size is 150 for instances with 20 customers, 300 for 50 customers, etc. Now, we fixed to 150 independent on the number of customers. The second modification changes only the construction costs, keeping original capacities as they are. Construction costs are increased proportionally to keep marginal costs of building one module fix for all instance sizes. Considering the same example for $L = 3$, when module size is 150, the construction cost of the first module is 150,000 (subsequent modules are less expensive due to economies of scale). However, when module size is 300 instead of using the same cost,

Algorithm 6 GA+VND for DFLPM

```
1: Create initial population $\mathcal{P}_0 = \{x_0^1, \ldots, x_0^P\}$ using $LRH$ and/or $CH1$ and $CH2$;
2: Evaluate and sort individuals in $\mathcal{P}_0$;
3: Apply *Elitism* for best individual $x_0^1$;
4: Set $n = 1$;
5: **while** stopping condition not met **do**
6:    **for** $g = 1$ **to** $R$ **do**
7:       Apply *GA Operators*;
8:       Sort $\mathcal{P}_g$;
9:       **if** $x_g^1 \neq x_{g-1}^1$ **then**
10:          Apply *Elitism* for $x_g^1$;
11:      **else if** $g \% G = 0$ **then**
12:          Apply *Elitism* for $x_g^p$ where $p = RAND\{1, \beta P\}$;
13:      **end if**
14:      **if** $f(x_g^1) < f(x_{g-1}^1)$ **then**
15:          $g = 1$;
16:      **else**
17:          $g = g + 1$;
18:      **end if**
19:   **end for**
20:   Apply *Mutation* $n$ times to the whole population except the best individual creating a new $\mathcal{P}_0$;
21:   Apply *Elitism* for $x_0^p$ where $p = RAND\{2, P\}$;
22:   Evaluate and sort $\mathcal{P}_0$;
23:   $n = n + 1$;
24: **end while**
25: **return** the best individual found;
```

we raise proportionally to the module capacity, going to 300,000 now. These two modifications result in two instance sets with 144 instances each.

Finally, two new maximum number of modules scenarios were created for the new instances. They are for L equal 15 and 30. Module sizes and installation costs were calculated in the same way as for L equals to 3, 5 and 10. The implication is that each module will have proportionally lower capacity when L is larger.

6.1 Order of neighborhoods exploration

The first set of experiments aims to determine the best sequence of neighborhoods to explore in the VND. Each experiment consists in exploring different neighborhoods starting at the same random solutions to verify the effectiveness of local searches when applied individually and in combination with other ones. One hundred feasible solutions were generated using CH1 and CH2 with equal proportion for smaller instances (10/20, 10/50 and 50/50) and $L = \{3, 5, 10\}$. On the first round of experiments, we test the use of each neighborhood individually, as a simple exhaustive neighborhood search. Table 2 presents the results for these experiments. The first column refers to the instance group and the others to the LSs. The relative gap between a solution x and the best solution x^* obtained by the MIP in Jena et al. [23] is calculated as $gap(x) = \dfrac{\bar{f}(x) - f(x^*)}{\bar{f}(x)}$. Results are shown for the best solution generated by each local search and the average solution of all 100 local optima.

Table 2: Results for the application of single neighborhoods

Inst	*LS1*			*LS2*			*LS3*		
	Best gap (%)	*Avg gap (%)*	*Time (s)*	*Best gap (%)*	*Avg gap (%)*	*Time (s)*	*Best gap (%)*	*Avg gap (%)*	*Time (s)*
10/20	1.29	4.63	1.5	0.17	1.40	9.8	4.02	20.11	8.1
10/50	0.33	1.71	2.5	0.13	0.81	21.6	2.53	13.18	14.7
50/50	2.19	5.97	25.8	0.39	1.47	999.4	5.81	22.33	638.0
Avg	*1.27*	*4.10*	*9.9*	*0.23*	*1.23*	*343.6*	*4.12*	*18.54*	*220.3*

We note from Table 2 that LS1 is the fastest and LS2 is the most effective searches. This is an expected behavior because LS1 has the smallest time complexity as discussed before, since it does not move modules between facilities. This leads to local optima with a nearly optimal number of open modules but located at non-optimal facilities. LS3 is the opposite since it only modifies positions of existing modules without closing modules with too much idle capacity, common in solutions generated by the CHs. LS2 is the middle ground that performs both types of operations. Thus, it is more effective but at an additional computational cost.

A new round of experiments is then performed to assess the performance of these neighborhoods in pairs using the VND structure. Table 3 shows these results, that confirm the analysis on the previous round. When LS1 is performed first it better prepares the solution so that the other heuristics reach good local optima faster. This is evident when we compare the experiments with the pairs $LS1 \rightarrow LS2$ and $LS1 \rightarrow LS3$ with their inverse order. Both local searches achieve similar results, but when LS1 is searched first the running time is around a quarter of that of the other way around. The average gap of using LS2 instead of LS3 with LS1 is better but running time is higher, which makes the results regarding which is the best neighborhood inconclusive.

Table 3: Results for the application of pairs of neighborhoods in a VND

Inst	$LS1 \rightarrow LS2$			$LS2 \rightarrow LS1$			$LS1 \rightarrow LS3$		
	Best gap (%)	*Avg gap (%)*	*Time (s)*	*Best gap (%)*	*Avg gap (%)*	*Time (s)*	*Best gap (%)*	*Avg gap (%)*	*Time (s)*
10/20	0.01	0.21	2.8	0.01	0.20	10.4	0.01	0.40	2.8
10/50	0.00	0.04	3.6	0.00	0.03	22.2	0.00	0.12	3.6
50/50	0.01	0.33	218.0	0.01	0.29	969.2	0.02	0.70	165.9
Avg	*0.01*	*0.20*	*74.8*	*0.01*	*0.18*	*333.9*	*0.01*	*0.41*	*57.4*

Inst	$LS3 \rightarrow LS1$			$LS2 \rightarrow LS3$			$LS3 \rightarrow LS2$		
	Best gap (%)	*Avg gap (%)*	*Time (s)*	*Best gap (%)*	*Avg gap (%)*	*Time (s)*	*Best gap (%)*	*Avg gap (%)*	*Time (s)*
10/20	0.01	0.34	9.4	0.15	1.24	9.9	0.23	2.09	13.1
10/50	0.00	0.09	16.4	0.12	0.77	21.8	0.14	1.47	24.8
50/50	0.03	0.67	654.3	0.36	1.41	875.6	0.23	1.66	859.2
Avg	*0.01*	*0.37*	*226.7*	*0.21*	*1.14*	*302.4*	*0.20*	*1.74*	*299.0*

A third round of experiments is performed to identify whether the exploration of the three neighborhoods is more effective than using only two. Table 4 shows the results of the application of the VND with the three LSs. We fixed LS1 to always be performed first.

Table 4: Results for the application of all neighborhoods in a VND

Inst	$LS1 \rightarrow LS2 \rightarrow LS3$			$LS1 \rightarrow LS3 \rightarrow LS2$		
	Best gap (%)	*Avg gap (%)*	*Time (s)*	*Best gap (%)*	*Avg gap (%)*	*Time (s)*
10/20	0.01	0.14	3.0	0.00	0.13	3.2
10/50	0.00	0.04	3.8	0.00	0.04	3.9
50/50	0.01	0.32	191.3	0.00	0.35	162.2
Avg	*0.01*	*0.17*	*66.0*	*0.00*	*0.17*	*56.4*

Comparing the best pairs from Table 3 and the results in Table 4 we note that the addition of the third neighborhood improves solution quality and speeds up the search in both scenarios. When compared to the exploration of single neighborhoods, the multiple neighborhood approach is much more efficient. Since both orders provided almost identical solutions, we decided to switch priorities between LS2 and LS3 randomly at every VND call as a search strategy.

6.2 Linear relaxation heuristic comparison with the exact method from the literature for the benchmark instances of Jena et al. [23]

Jena et al. [23] showed that the integrality gap of the linear relaxation problem using the MIP model described in Section 2 provides a strong lower bound when applied to the benchmark instances. We tested the capacity of LRH to generate near-optimal solutions from the relaxed solutions improved by VND. Table 5 shows for each group of instances the number of instances with proven optimality within a time limit of three hours using Jena et al. [23] MIP model (column *#Opt* in *ER-GMC*), the average time to prove optimality (column *Time(s)* in *ER-GMC*), the average lower bound gap of the solution found by the linear relaxation described in Jena et al. [23], where the linear relaxation gap of a solution is calculated as $\frac{f(x) - f'(x)}{f(x)}$ for the linear relaxation solution $f'(x)$ and the optimal solution $f(x)$ (column *LR Gap*), and the average gap to the optimal solution and time for the solution found by LRH improved by VND (columns in *LRH*). Instances without an optimal solution are removed from these statistics.

Table 5: Results for the comparison of the proposed linear relaxation heuristic with the MIP model of Jena et al. [23] for the benchmark instances solved to optimality

L	*Inst*	*ER-GMC*		*LR*	*LRH*	
		#Opt	Time (s)	Gap (%)	Gap (%)	Time (s)
3	10/20	12	0.3	0.45	0.02	0.1
	10/50	12	0.4	0.17	0.00	0.1
	50/50	12	3.1	0.12	0.10	0.7
	50/100	12	2.9	0.00	0.00	1.0
	50/250	12	6.6	0.00	0.00	2.1
	100/250	12	17.9	0.01	0.01	8.3
	100/500	12	51.6	0.02	0.02	35.0
	100/1000	12	88.2	0.00	0.01	81.9
5	10/20	12	0.9	0.59	0.06	0.2
	10/50	12	2.4	0.39	0.08	0.3
	50/50	12	512.9	0.36	0.06	3.0
	50/100	12	9.3	0.06	0.02	3.1
	50/250	12	14.6	0.00	0.00	4.7
	100/250	12	42.0	0.01	0.01	16.7
	100/500	12	93.0	0.02	0.02	48.0
	100/1000	12	164.3	0.00	0.01	109.4
10	10/20	12	27.2	0.65	0.06	1.7
	10/50	12	45.0	0.47	0.03	2.4
	50/50	5	306.4	0.64	0.09	7.5
	50/100	9	194.1	0.32	0.03	13.0
	50/250	12	402.5	0.16	0.01	34.9
	100/250	11	591.9	0.11	0.03	128.0
	100/500	12	391.2	0.04	0.01	170.2
	100/1000	12	451.4	0.00	0.01	206.0
Avg			*136.2*	*0.18*	*0.03*	*37.3*

We see that linear relaxation gap tends to decrease when instance sizes increase. The heuristic results follow a similar pattern. The longest gaps are observed on smaller instances (10/20, 10/50 and 50/50). A possible explanation lies on instance's structure. The ratio module capacity/total demand is approximately constant in all instances. As a consequence, the minimum number of modules that have to be open to satisfy feasibility is around the same. When location costs have a greater impact than allocation costs on objective function, one implication on the optimum solution is that the number of modules opened tends to be reduced, since construction of excess capacity is too costly. On the other hand, if construction costs are low, more facilities can be opened closer to customers to reduce allocation costs. Mathematically, the reduction of location costs turns the problem easier to be solved. One example is when all location costs are zero. The dynamic location problem for T periods would be reduced to T min-cost flow problems where all facilities could be opened since no penalties are incurred, turning into a relatively easy problem to solve with existing methods. In fact, we observed that the number of modules opened in smaller instances is smaller than in bigger instances at optimal solutions. Also, the proportion of location costs on total costs decrease from around 60–70% on 10/20 instances to 20–30% on 100/1000 instances. This is even more noticeable when we analyze separately the two types of instances according to allocation cost proportion. The average time to solve the regular scenario instances is 4.5 times higher than when allocation costs are raised 500%. From the 11 instances not solved to optimality within three hours, nine are in the first scenario, and the two that are in the second are derived from instances not solved in the first one as well.

LRH solutions are on average only 0.03% from optimal and obtained 3.7 times faster than what CPLEX takes to prove optimality. Considering only the worst instances (10/20, 10/50 and 50/50), the average gap increases to 0.05%. In Section 6.1 we showed that it is possible to obtain an average gap of 0.00% on the same instances when initializing several random solutions and improving them with VND. However, the neighborhood exploration is costly in terms of time. So, a random approach is undesirable when instance size grows. We consider that on the benchmark instances from Jena et al. [23] the multiple application of VND on randomly generated solutions is enough when instances are small. For bigger instances, LRH could solve them efficiently since the observed gap is on average only 0.01% and it takes less than half of CPLEX time. The evolutionary heuristic, however, is more useful for the modified instances with the new construction cost structures.

6.3 Parameter setting of the evolutionary algorithm

The proposed evolutionary algorithm requires the definition of several parameters. The size of each class of individuals was chosen as in Buriol et al. [9], being $\alpha = 25\%$ and $\beta = 95\%$. The number of individuals generated by each constructive heuristic at the beginning of the algorithm was split in half for each heuristic. We also set the mutation rate M at a fixed value equal to 20%. Another four parameters were set empirically through the results of experiments using a grid search method of parameterization. They are: the population size (P), the interval of generations to apply the elitism process (G), the cycle size (number of generations) to mutate population (R), and the total number of cycles (N).

The experiments were conducted on the following combinations of settings: (i) $P = \{50, 200, 500\}$; (ii) $G = \{5, 10, 20\}$; (iii) $R = \{5, 20, 50\}$; and (iv) $N = \{0, 1, 5, 10\}$. Table 6 reports average results of dominant combinations of parameters in terms of quality and time for 10 distinct runs of the GA+VND regarding all modified instances from sets 10/20, 10/50 and 50/50 and $L = \{3, 5, 10\}$. The first four columns refer to the respective parameters (i), (ii), (iii) and (iv). The average gap in the fifth column was determined as before. The running time in the sixth column is the average for all instances.

Table 6: Dominant set of parameter combinations for the GA+VND

P	G	R	N	Gap (%)	Time (s)
50	5	20	10	0.00	15.9
50	5	50	1	0.01	9.2
50	10	20	5	0.02	8.2
50	5	50	0	0.03	6.5
50	20	20	5	0.04	6.3
50	5	20	0	0.06	4.6
50	10	50	0	0.07	4.2
50	20	20	1	0.10	3.5
50	10	20	0	0.12	3.0
50	20	50	0	0.13	2.8
50	20	20	0	0.17	2.3
50	20	5	0	0.25	1.6

Any combination of parameters from Table 6 could be chosen since we cannot claim that one clearly outperforms another. In common, they all have the same population size. As we want to focus on solution quality, we choose the combination that generates the best gap, which is: population composed by 50 chromosomes each generation; elitism (VND) is applied at every 5 generations to a random individual from class A or B; the entire population except the best chromosome is mutated at every 20 generations without improvement in the best solution; and 10 population mutations are done at maximum before algorithm stops. This was not the sole combination that resulted in a gap of 0.00%, but it was the one with the lowest running time. For the next experiments, we keep the settings with the best performance observed.

6.4 Heuristics comparison against the exact method for the new instances

These experiments compare the efficiency of the evolutionary algorithm GA+VND and the relaxation-based heuristic LRH to the state-of-the-art method ER-GMC of Jena et al. [23] to solve the DFLPM and, thus, determine the advantages of using the heuristic methods. The comparison is made for the instances where a feasible solution could be found in all three methods. Although GA+VND is capable of finding feasible solutions in all instances easily, for some of them either the linear relaxation or the exact approach was not able to and a comparison was not possible.

Table 7 shows the results of experiments for 10 distinct runs of the GA+VND, compared to the LRH improved by VND, and the ER-GMC formulation for all modified instances. Results are divided into instances with optimality proven within three hours (column *Optimal*) and instances that a feasible solution is found but optimality is not guaranteed (column *Feasible*). Column *No solution* shows the number of instances that a feasible solution was not found by LRH or ER-GMC. We highlight that in all these instances the GA+VND finds feasible solutions. However, as there is no solution from either the ER-GMC or the LRH to compare to, we cannot provide gaps for them. Some rows are omitted from the table since all 12 instances are in *No solution* column. All gaps shown are relative to the best solution provided by ER-GMC.

Several conclusions can be obtained from Table 7. We highlight the following:

- GA+VND is better than LRH in most scenarios. On average it provides better results in less time either when solutions have optimality proven by ER-GMC or when only a feasible solution without optimality proof is known. In the former, the quality of solutions is better for the GA+VND in under half of the runtime of LRH; in the latter, GA+VND presents solutions 0.43% better than the state-of-the-art MIP in less than 15% of the runtime;
- On the *Optimal* instances the average gap of GA+VND is considerable (0.11%), mostly because the heuristic has some difficulty on instances when the number of modules L equals to 15 and 30. Without these instances, the gap falls to less than 0.01%;
- On instances with unproven optimality by ER-GMC, GA+VND finds on average better solutions than the exact method (gap -0.43%) spending only one eighth of the maximum time for CPLEX. The gain is even higher when considering instances with no solution provided by ER-GMC that GA+VND is capable of finding a feasible solution;
- Time complexity in terms of L is much smaller for GA+VND than ER-GMC. For $L = 3$ and 5, considering instances with optimality proven, ER-GMC time is comparable to GA+VND. For a bigger L the time differences between both methods are evident, favorably to GA+VND.

The modified instances greatly increase the impact of module construction costs on total costs. We observed that this makes them harder to solve using an exact approach. For instances with a bigger number of modules per facility the MIP modeled in CPLEX with default settings is a useless tool for most instances with as little as 50 facilities. In most of them, the memory required by CPLEX is too large to store its branch-and-bound tree in such way that we run out of memory before a feasible solution could be found. The large number of variables for a bigger L makes a linear relaxation approach also impossible to be used due to the time required to finish the linear relaxation problem optimization. It is important to note that in some runs, specifically when the instance size grows, the heuristic stopped due to maximum memory limitation. On these particular experiments this did not represent a problem since the available memory was enough to the heuristic generate satisfactory solutions before stopping. In case of bigger instances or in environments with limited memory, some memory management should be required on the hash table that stores FCM results. Finally, although we cannot guarantee solution quality provided by GA+VND on bigger instances, its good performance on the instances that could be compared in these experiments is a good indicator that the proposed heuristic is a reliable tool to deal with this problem.

Table 7: Results for the comparison with the state-of-the-art method of [23] (ER-GMC) with the linear relaxation heuristic (LRH) and the evolutionary heuristic (GA+VND)

		Optimal						*Feasible*						*No solution*
L	*Inst*	*#Inst*	*GA+VND*		*LRH*		*ER-GMC*	*#Inst*	*GA+VND*		*LRH*		*ER-GMC*	*#Inst*
			Gap (%)	Time (s)	Gap (%)	Time (s)	Time (s)		Gap (%)	Time (s)	Gap (%)	Time (s)	Time (s)	
	10/20	12	0.00	0.9	0.04	0.1	0.4	-	-	-	-	-	-	-
	10/50	12	0.00	1.5	0.00	0.4	6.3	-	-	-	-	-	-	-
	50/50	10	0.01	29.2	0.08	2.5	1185.4	2	0.02	36.0	0.42	8.6	10800.2	-
	50/100	7	0.01	53.5	0.07	4.5	255.1	5	0.01	189.7	0.18	24.1	10800.3	-
	50/250	6	0.00	140.7	0.00	15.7	15.2	6	-0.05	1391.0	0.17	274.0	10800.7	-
3	100/250	6	0.01	400.6	0.00	101.8	69.1	6	-0.28	6397.6	-0.10	1439.0	10801.3	-
	100/500	6	0.00	1152.4	0.00	473.8	283.5	4	-0.89	4987.9	-0.74	6733.2	10803.2	2
	100/1000	6	0.03	2075.5	0.04	2890.2	1582.8	-	-	-	-	-	-	6
	10/20	12	0.00	1.0	0.08	0.3	2.1	-	-	-	-	-	-	-
	10/50	12	0.00	2.1	0.03	1.1	24.7	-	-	-	-	-	-	-
	50/50	7	0.00	30.1	0.18	9.3	620.7	5	0.01	62.7	0.21	32.6	10800.2	-
	50/100	6	0.00	41.6	0.09	21.3	37.2	6	0.00	344.5	0.30	181.7	10800.5	-
	50/250	6	0.01	141.6	0.00	116.5	114.4	6	-0.23	2323.9	-0.05	1861.8	10801.3	-
5	100/250	6	0.01	391.6	0.05	365.8	497.5	4	-2.62	6193.3	-2.39	7216.2	10802.6	2
	100/500	6	0.00	1037.1	0.00	1245.0	902.8	-	-	-	-	-	-	6
	100/1000	4	0.08	2172.4	0.10	2759.0	2856.8	-	-	-	-	-	-	8
	10/20	12	0.01	1.7	0.11	2.9	40.6	-	-	-	-	-	-	-
	10/50	12	0.01	4.0	0.21	9.8	227.8	-	-	-	-	-	-	-
	50/50	4	0.00	63.8	0.12	218.3	2983.7	8	-0.13	130.8	0.33	444.8	10800.6	-
	50/100	5	0.00	93.9	0.24	534.8	1895.2	7	-0.38	447.4	-0.03	1796.3	10801.2	-
	50/250	4	0.01	274.4	0.13	758.2	3418.4	5	-3.06	1200.4	-3.16	4886.3	10803.4	3
10	100/250	1	0.00	1282.9	0.04	8133.8	8582.6	2	-0.02	1708.7	0.20	3790.1	10808.3	9
	100/500	1	0.02	1979.8	0.00	6649.4	8722.2	-	-	-	-	-	-	11
	10/20	12	0.25	2.2	0.25	8.5	354.5	-	-	-	-	-	-	-
	10/50	11	0.06	6.2	0.31	29.1	1049.6	1	-0.03	9.4	0.27	29.7	10800.2	-
15	50/50	4	0.16	51.3	0.32	1137.5	2835.3	8	-0.02	150.1	0.44	1210.5	10801.4	-
	50/100	5	0.09	158.9	0.03	2310.4	5617.4	6	-0.68	285.3	-0.61	3170.1	10802.6	1
	50/250	1	0.00	370.2	0.00	2021.5	7245.2	4	-0.80	685.4	-0.75	5122.4	10807.2	7
	10/20	12	1.00	1.9	0.89	24.2	22.6	-	-	-	-	-	-	-
	10/50	12	0.32	6.0	0.75	144.0	262.8	-	-	-	-	-	-	-
30	50/50	8	0.30	52.9	0.53	3542.4	4543.1	4	1.17	87.4	3.31	4760.7	10805.0	-
	50/100	1	0.19	283.5	0.33	8559.5	8673.5	1	0.43	187.8	-0.02	9340.5	10809.5	10
	Avg		*0.11*	*215.6*	*0.19*	*528.4*	*948.7*		*-0.43*	*1428.9*	*-0.12*	*2271.0*	*10802.0*	

7 Conclusions

This paper presented some heuristics for the Dynamic Facility Location Problem with Modular Capacities (DFLPM). The first one is based on the linear relaxation of a mixed integer programming formulation for the problem with proven good results. The second is a hybrid evolutionary heuristic that uses the structure of a genetic algorithm with a periodic intensification phase that explores several neighborhoods in a variable neighborhood descent framework (GA+VND). The DFLPM is a relatively new problem in facility location literature that generalizes several facility location problems allowing facilities to have their capacities adjusted during a planning horizon using blocks of capacities, also called modules.

We adapted neighborhood exploration strategies typically applied to simpler FLPs to the DFLPM. Operations of adding new modules to facilities, closing existing modules and swapping modules between facilities can be used separately or in combinations to design effective heuristics, such as local searches or mutation operators, to improve solutions generated by other methods. For the GA+VND, three neighborhoods were defined and explored exhaustively until reaching a common local optimum. We have also shown that the same structure is very effective when improving the solution created based on the linear relaxation of the MIP formulation.

The heuristics were tested for benchmark instances from the literature. We have shown that for the existing instances, a simple approach as the linear relaxation heuristic followed by the proposed local searches is capable of finding near optimal solutions quite faster than a state-of-the-art exact approach. We also presented instances to better represent construction costs of modules in real cases. The adapted instances were proven to be hard to solve by the exact approach. The evolutionary heuristic presented was capable of solving them much faster and, in most scenarios, finding better solutions than the exact method and the alternative heuristic based on linear relaxation.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Upper Saddle River, 1993.

[2] H.-C. An, M. Singh, and O. Svensson. LP-based algorithms for capacitated facility location. SIAM Journal on Computing, 46(1):272–306, 2017.

[3] A. Antunes and D. Peeters. On solving complex multi-period location models using simulated annealing. European Journal of Operational Research, 130(1):190–201, 2001.

[4] A. B. Arabani and R. Z. Farahani. Facility location dynamics: An overview of classifications and applications. Computers & Industrial Engineering, 62(1):408–420, 2012.

[5] M. A. Arostegui, S. N. Kadipasaoglu, and B. M. Khumawala. An empirical comparison of tabu search, simulated annealing, and genetic algorithms for facilities location problems. International Journal of Production Economics, 103(2):742–754, 2006.

[6] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. SIAM Journal on Computing, 33(3):544–562, 2004.

[7] S. Basu, M. Sharma, and P. S. Ghosh. Metaheuristic applications on discrete facility location problems: a survey. Opsearch, 52(3):530–561, 2015.

[8] T. Becker, S. Lier, and B. Werners. Value of modular production concepts in future chemical industry production networks. European Journal of Operational Research, 276(3):957–970, 2019.

[9] L. S. Buriol, M. G. C. Resende, C. C. Ribeiro, and M. Thorup. A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. Networks, 46(1):36–56, 2005.

[10] P. Chardaire, A. Sutter, and M.-C. Costa. Solving the dynamic facility location problem. Networks, 28(2): 117–124, 1996.

[11] F. A. Chudak and D. P. Williamson. Improved approximation algorithms for capacitated facility location problems. Mathematical Programming, 102(2):207–222, 2005.

[12] G. Cornuéjols, R. Sridharan, and J.-M. Thizy. A comparison of heuristics and relaxations for the capacitated plant location problem. European Journal of Operational Research, 50(3):280–297, 1991.

[13] I. Correia and T. Melo. A multi-period facility location problem with modular capacity adjustments and flexible demand fulfillment. Computers & Industrial Engineering, 110:307–321, 2017.

[14] H. Delmaire, J. A. Díaz, E. Fernández, and M. Ortega. Reactive GRASP and tabu search based heuristics for the single source capacitated plant location problem. INFOR: Information Systems and Operational Research, 37(3):194–225, 1999.

[15] J. Dias, M. E. Captivo, and J. Clímaco. Dynamic location problems with discrete expansion and reduction sizes of available capacities. Investigação Operacional, 27(2):107–130, 2007.

[16] Z. Drezner and H. W. Hamacher. Facility Location: Applications and Theory. Springer-Verlag, Berlin Heidelberg, 2002.

[17] M. Ericsson, M. G. C. Resende, and P. M. Pardalos. A genetic algorithm for the weight setting problem in ospf routing. Journal of Combinatorial Optimization, 6(3):299–333, 2002.

[18] D. R. M. Fernandes, C. Rocha, D. Aloise, G. M. Ribeiro, E. M. Santos, and A. Silva. A simple and effective genetic algorithm for the two-stage capacitated facility location problem. Computers & Industrial Engineering, 75:200–208, 2014.

[19] M. Gendreau and J.-Y. Potvin. Metaheuristics in combinatorial optimization. Annals of Operations Research, 140(1):189–213, 2005.

[20] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. European Journal of Operational Research, 130(3):449–467, 2001.

[21] J. H. Holland. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. U Michigan Press, Oxford, 1975.

[22] H. Jang, K. Hwang, T. Lee, and T. Lee. Designing robust rollout plan for better rural perinatal care system in korea. European Journal of Operational Research, 274(2):730–742, 2019.

[23] S. D. Jena, J.-F. Cordeau, and B. Gendron. Dynamic facility location with generalized modular capacities. Transportation Science, 49(3):484–499, 2015.

[24] S. D. Jena, J.-F. Cordeau, and B. Gendron. Modeling and solving a logging camp location problem. Annals of Operations Research, 232(1):151–177, 2015.

[25] S. D. Jena, J.-F. Cordeau, and B. Gendron. Lagrangian heuristics for large-scale dynamic facility location with generalized modular capacities. INFORMS Journal on Computing, 29(3):388–404, 2017.

[26] D. Jungnickel. Graphs, Networks and Algorithms. Springer-Verlag, Berlin Heidelberg, 4th edition, 2013.

[27] A. Klose and A. Drexl. Facility location models for distribution system design. European Journal of Operational Research, 162(1):4–29, 2005.

[28] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facility location problems. Journal of Algorithms, 37(1):146–188, 2000.

[29] J. Kratica, D. Tošic, V. Filipović, and I. Ljubić. Solving the simple plant location problem by genetic algorithm. RAIRO-Operations Research, 35(1):127–142, 2001.

[30] G. Laporte, S. Nickel, and F. Saldanha da Gama. Location Science, volume 145. Springer International Publishing, Berlin, 2015.

[31] D.-H. Lee and M. Dong. A heuristic approach to logistics network design for end-of-lease computer products recovery. Transportation Research Part E: Logistics and Transportation Review, 44(3):455–474, 2008.

[32] T. Lee and H. Jang. An iterative method for simultaneously locating trauma centers and helicopters through the planning horizon. Operations Research for Health Care, 19:185–196, 2018.

[33] M. T. Melo, S. Nickel, and F. Saldanha da Gama. Facility location and supply chain management–a review. European Journal of Operational Research, 196(2):401–412, 2009.

[34] S. H. Owen and M. S. Daskin. Strategic facility location: A review. European Journal of Operational Research, 111(3):423–447, 1998.

[35] R. H. Pearce and M. Forbes. Disaggregated Benders decomposition and branch-and-cut for solving the budget-constrained dynamic uncapacitated facility location and network design problem. European Journal of Operational Research, 270(1):78–88, 2018.

[36] Z. Rafie-Majd, S. H. R. Pasandideh, and B. Naderi. Modelling and solving the integrated inventory-location-routing problem in a multi-period and multi-perishable product supply chain with uncertainty: Lagrangian relaxation algorithm. Computers & Chemical Engineering, 109:9–22, 2018.

[37] C. S. Revelle, H. A. Eiselt, and M. S. Daskin. A bibliography for some fundamental problem categories in discrete location science. European Journal of Operational Research, 184(3):817–848, 2008.

[38] A. Shulman. An algorithm for solving dynamic capacitated plant location problems with discrete expansion sizes. Operations Research, 39(3):423–436, 1991.

[39] R. E. Tarjan. Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm. Mathematical Programming, 78(2):169–177, 1997.

[40] V. Verter and M. C. Dincer. Facility location and capacity acquisition: an integrated approach. Naval Research Logistics, 42(8):1141–1160, 1995.

[41] D. Whitley. Next generation genetic algorithms: A user's guide and tutorial. In M. Gendreau and J.-Y. Potvin, editors, Handbook of Metaheuristics, pages 245–274. Springer, New York, 3rd edition, 2019.

[42] J. Wollenweber. A multi-stage facility location problem with staircase costs and splitting of commodities: model, heuristic approach and application. OR Spectrum, 30(4):655–673, 2008.

[43] X.-S. Yang. Nature-Inspired Metaheuristic Algorithms. Luniver press, Frome, 2nd edition, 2010.

[44] J. Zhang, B. Chen, and Y. Ye. A multiexchange local search algorithm for the capacitated facility location problem. Mathematics of Operations Research, 30(2):389–403, 2005.