# Max-Min Dispersion with Capacity and Cost for a Practical Location Problem

Isaac Lozano-Osorio[b], Anna Martínez-Gavara[a,*], Rafael Martí[a], Abraham Duarte[b]

[a] *Departament d'Estadística i Investigació Operativa, Universitat de València. C/Doctor Moliner, 50, 46100 Burjassot, València (Spain)*
[b] *Department of Computer Sciences, University Rey Juan Carlos. C/ Tulipán s/n. 28933 Móstoles, Madrid (Spain)*

## Abstract

Diversity and dispersion problems deal with selecting a subset of elements from a given set in such a way that their diversity is maximized. This study considers a practical location problem recently proposed in the context of max-min dispersion models. It is called the generalized dispersion problem, and it models realistic applications by introducing capacity and cost constraints. We propose two effective linear formulations for this problem, and develop a hybrid metaheuristic algorithm based on the variable neighborhood search methodology, to solve real instances. Extensive numerical computational experiments are performed to compare our hybrid metaheuristic with the state-of-art heuristic, and with integer linear programming formulations (ILP). Results on public benchmark instances show the superiority of our proposal with respect to the previous algorithms. Our extensive experimentation reveals that ILP models are able to optimally solve medium-size instances with the Gurobi optimizer, although metaheuristics outperform ILP both in running time and quality in large-size instances.

*Keywords:* metaheuristics, combinatorial optimization, diversity maximization, dispersion, variable neighborhood search.

## 1. Introduction

Discrete diversity maximization was introduced by Kuby (1988) in a seminal paper that originated an important family of optimization problems in the context of location (Martí et al., 2021b). Although optimizing diversity has been a key topic in Mathematics for many decades, it was mainly devoted to continuous models. However, in the last thirty years, the study of diversity has been applied to solve practical location problems in Operations Research and Computer Science. In this paper we target a realistic diversity problem arising in facility location called the Generalized Dispersion Problem.

In its simplest form, the problem of maximizing diversity or dispersion deals with selecting a subset of elements from a given set in such a way that the distance among the selected ele-

[⋆]This research was partially supported by the Spanish Ministry (MCIU/AEI/FEDER,UE) with grant ref. PGC2018-095322-B-C21 and PGC2018-095322-B-C22.

[*]Corresponding author.

*Email addresses:* `isaac.lozano@urjc.es` (Isaac Lozano-Osorio), `gavara@uv.es` (Anna Martínez-Gavara), `rmarti@uv.es` (Rafael Martí), `abraham.duarte@urjc.es` (Abraham Duarte) ments is maximized. Although we may think on the standard distance definition based on the Euclidean formula, many applications may require non-Euclidean geometries, such as those induced by affinities relationships expressing a relative degree of attraction between the elements, as arises in settings with a behavioral component. Typical examples are architectural space planning and analysis of social networks (Glover et al., 1998).

Over the last few years, different mathematical expressions have been proposed to capture the notion of diversity, dispersion, or even equity. The minimum and the sum of the distances among the selected elements (MaxMin and MaxSum models respectively) are probably the most well-known models to translate the term diversity into an integer quadratic expression. In discrete location problems, where points in the plane represent potential locations to set facilities over a given territory, MaxMin model solutions do not avoid to select points in the central region of the plane, while MaxSum model solutions fail do locate them (Parreño et al., 2021). The MaxMin model is therefore the model of choice when it comes to solve location problems, since its solutions are better suited to cover the territory with disperse points.

In this paper we consider the generalized dispersion problem (GDP), that is based on the MaxMin diversity model, and incorporates capacity and cost constraints. This model was introduced in a theoretical way by Rosenkrantz et al. (2000), and applied to solve a practical problem by Martínez-Gavara et al. (2021). In particular, the authors considered the location of the same-type of facilities, such a shop franchises, where we want to avoid their proximity. This is also the case of hospitals or schools, where we are providing a service and we do not want to be far away from the customers (or patients) and, at the same time, we want them not to be close to each other. The inclusion of capacity constraints translates into the number of customers (patients) that each facility may serve, and the cost constraints appear in a natural way when modeling a real situation. Our motivation to solve this model, comes from a realistic case proposed in Daskin (2011) in the context of locating branch offices in U.S.. In our computation experience in Section 5, we consider both randomly generated instances and this realistic case.

In spite of its practical significance, side constraints, such as capacity and cost, have been largely ignored in the discrete diversity literature. We have only found the two papers cited above considering these two types of constraints, while Martí et al. (2021b) found more than 50 papers in unconstrained discrete diversity (apart for the standard constraint to set the number of elements to be selected). In this paper we study this $\mathcal{NP}$-hard problem, which pose a challenge to optimization methods, and present an empirical comparison with existing methods.

In the last decades, algorithmic advances as well as hardware and software improvements have provided an excellent environment to create and develop solving methods to hard optimization problems. Modern exact and heuristic techniques are dramatically enhancing our ability to solve significant practical problems. A large number of intelligent algorithms based on artificial intelligence (Glover et al., 2021), social behavior Li et al. (2021) or bio-inspired learning strategies (Feng et al., 2021; Li et al., 2020; Wang et al., 2019) have been extensively researched in the past few decades. In this paper, we consider the variable neighborhood search (VNS) metaheuristic (Hansen & Mladenović, 2005), which has exhibited remarkable performance in many hard optimization problems due to a systematic change in the neighborhood exploration. A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms (Sörensen & Glover, 2013). On the other hand, a heuristic is an algorithm based on context dependent strategies to solve an optimization problem. We adapted the VNS methodology to solve the MaxMin dispersion problem with capacity and cost constraints (namely, the GDP.)

Considering that this problem has been already addressed in a previous paper, we itemize now the main contributions of this work with respect to previous developments:

(i) We propose an effective formulation for the GDP and adapt the method in Sayah & Irnich (2017) to solve medium-size instances to optimality,

(ii) We propose heuristic methods, based on the variable neighborhood metaheuristic, to solve large-size and realistic GDP instances. In particular, we consider different constructive methods, and three local search operators (swap, add, and drop),

(iii) We study efficient search strategies to overcome the lack of information given by the "flat landscape" of a MaxMin objective, where many solutions share the same objective value. Specifically, we propose an extended definition of improving move, and

(iv) We perform numerical experiments that first disclose the best strategies for our methods, and then compare it with the previous proposals by means of statistical tests. We finish our experimentation solving the practical case that triggered our interest in this problem.

The rest of this paper is organized as follows. We first propose mathematical models for the diversity problems considered here in Section 2. Previous work is described in Section 3 to clearly established the context of the GDP and the associated state-of-the-art methods. Section 4 provides a detailed description of our metaheuristics for the generalized dispersion problem, and it constitutes the core of our contributions. Then, Section 5 describes our experimental study and the paper concludes in Section 6 with the lessons learnt in this study.

## 2. Mathematical models

We define in this section the elements of this model in terms of facility location problems (from now on, we will use element or site indistinctly). Given a set of $n$ potential facilities $V$ connected by edges (links) in $E$, the problem consists in finding a subset $P$ of $V$ satisfying capacity and cost constraints, so that the minimum distance among the sites in $P$ is maximized. Let $B$ be the minimum required capacity (level of service) and let $K$ be the maximum budget allowed. Then, for each site $i \in V$, we define $c_i \geq 0$ and $a_i \geq 0$ as its capacity and cost, respectively. Let $d_{ij} \geq 0$ be the distance between sites $i$ and $j$. The mathematical programming model for GDP is based on the standard binary variables $x_i$ that take the value 1 if site $i$ is selected and 0 otherwise. Then, it can be stated as follows:

$$\begin{aligned} \text{(GDP)} \quad \max \quad & \min_{i,j \in V, i \neq j} \quad d_{ij} x_i x_j \\ \text{s.t.} \quad & \sum_{i=1}^{n} c_i x_i \geq B \\ & \sum_{i=1}^{n} a_i x_i \leq K \\ & x_i \in \{0,1\} \qquad\qquad \forall\, i \in V. \end{aligned} \tag{1}$$

Let $P = \{i \in V : x_i = 1\}$ be the set of selected elements. Note that, as opposite to the MaxMin dispersion problem, the cardinality of the subset $P$ is not fixed beforehand, and it depends on the capacity and cost of the selected elements. Let $f(P)$ be the objective function, which measures the minimum distance between the pairs of sites in $P$. In mathematical terms: $f(P) = \min_{i,j \in P, i \neq j} d_{ij}$, and the objective of GDP is to find the set $P^*$ maximizing $f(P)$ for all $P \subset V$ satisfying the capacity and cost constraints.

The main drawback of this model is the non-linearity of the objective function, which prevents the use of standard mixed integer programming solvers as, for example, `Gurobi` or `CPLEX`. However, this quadratic problem can be reformulated as the following integer linear programming model, as proposed by Kuby (1988) for the $p$-dispersion problem (see also Erkut (1990)), and adapted in Martínez-Gavara et al. (2021) for the GDP:

$$
\begin{aligned}
\max \quad & m \\
\text{s.t.} \quad & \sum_{i=1}^{n} c_i x_i \geq B \\
& \sum_{i=1}^{n} a_i x_i \leq K \\
& m \leq d_{ij} + M(2 - x_i - x_j) \qquad \forall\ 1 \leq i < j \leq n \\
& x_i \in \{0,1\} \qquad \forall\ i \in V.
\end{aligned}
\tag{2}
$$

The upper bound $M$ in (2) on the distances values guarantees that $m$ is the minimum distance among the selected sites. The model in (2) is equivalent to the model in (1), but it has the advantage that can be solved using an integer programming solver for small to medium size instances.

Sayah & Irnich (2017) proposed a compact formulation for the $p$-dispersion problem based on the fact that the optimal objective function distance is one of the distance values in the input data. Let $D^0 < D^1 < \ldots < D^{r_{max}}$ be the different non-zero values in the distance matrix. Formulation (3) considers the original variables, $x_i$ and new $r_{max}$ binary variables, $z_r$, that take the value 1 if the set of selected sites has at least a minimum distance of $D^r$, and 0 otherwise. Then, we adapt the new formulation of Sayah & Irnich (2017) for GDP by adding capacity and cost constraints. We obtain the following integer linear programming model:

$$
\begin{aligned}
\max \quad & D^0 + \sum_{r=1}^{r_{max}} (D^r - D^{r-1}) z_r \\
\text{s.t.} \quad & \sum_{i=1}^{n} c_i x_i \geq B \\
& \sum_{i=1}^{n} a_i x_i \leq K \\
& z_r \leq z_{r-1} \qquad r = 1, \ldots, r_{max} \\
& x_i + x_j + z_r \leq 2 \qquad \forall i,j \in V : d_{ij} = D^{r-1} \\
& x_i, z_r \in \{0,1\} \qquad i \in V,\ r = 1, \ldots, r_{max}.
\end{aligned}
\tag{3}
$$

Formulations (2) and (3) are both compact since the number of variables and constraints in both cases do not exceed $n^2$. Specifically, model (2) consists of $n$ variables and $n(n-1)/2 + 3$

constraints, and model (3) has $n + r_{max}$ variables and $r_{max} + |E(D^{r_{max}})| + 2$ constraints, where $E(D) = \{(i,j) \in E : d_{ij} < D\}$. In Section 5, we perform an empirical analysis to compare both formulations, and to study the maximum instance size in order to find the optimal solution.

It is important to note that the model considered in this paper is built from a Max-Min objective function, which differs from the Max-Sum objective function also applied to diversity maximization. This has important implications in the design of both exact and heuristic methods. In particular, we adapted the Sayah & Irnich (2017) formulation because it also considers the Max-Min objective function, and at the same time it is a compact formulation, which means that only requires to be submitted to the solver once. An alternative is the Sayyady & Fathi (2016) formulation, also proposed for the Max-Min objective, but that requires to solve a sequence of ILPs, which is why we considered the former. On the other hand, we did not consider in our study the linearization by He et al. (2012) since it was proposed for Max-Sum problems.

## 3. Previous heuristic algorithms

In this section, we review the state-of-the-art algorithms to solve the GDP. As far as we know, there are two previous papers for this problem. The first one is due to Rosenkrantz et al. (2000), where the authors proposed simple heuristics with performance guarantee, and the second one due to Martínez-Gavara et al. (2021), who applied complex metaheuristics to obtain high quality solutions. Detailed descriptions of both methods follow.

Rosenkrantz et al. (2000) proposed a binary search based method to solve a variant of the GDP called the capacitated dispersion problem (CDP). This problem involves distance and total storage capacity in the same way that the GDP, but does not include the storage cost constraints. In particular, it is a greedy algorithm that performs a binary search over the non-zero sorted distances, in order to find a set of sites satisfying the capacity constraint with minimum inter-site distance as large as possible. Martínez-Gavara et al. (2021) adapted this algorithm to the GDP, and called it `TI_Ad`. To do so, the authors consider the ratio between the storage capacity and cost of each site to sort a preference list in which sites are included in the solution. `TI_Ad` basically selects the elements from this list in decreasing order, and checks both, capacity and cost constraint, to validate the feasibility of the selected set of sites. The method stops when the capacity level is reached.

Martínez-Gavara et al. (2021) applied two metaheuristics to obtain high quality solutions to the GDP: a greedy randomized adaptive search procedure, `GRASP`, and a long-term tabu search, `TS`. Specifically, GRASP is a multi-start methodology that in each iteration builds a solution from scratch, and improves it by applying a local search post-processing to obtain a local-optimum. The GRASP methodology is based on the statistical sampling of the solution space (Feo & Resende, 1995; Festa & Resende, 2016). A short description of the main features of the GRASP algorithm proposed in Martínez-Gavara et al. (2021) follows.

The construction phase starts by creating a candidate list (CL), which consists of all unassigned sites that can be inserted in the solution $X$ without exceeding the upper budget $K$. Then, the next site to be added to the solution is selected at random from a list of good candidates, called the restricted candidate list (RCL). To obtain RCL, the evaluation of each site in CL is guided by a greedy function that assesses the contribution of each site $i$ in CL regarding the objective function. Martínez-Gavara et al. (2021) proposed the greedy function

$\tilde{g}$ to collect in a single expression the three elements of this problem, distance, cost, and capacity:

$$\tilde{g}(i) = \beta_d \frac{\tilde{d}_i}{\max\limits_{j \in CL} d_j} + \beta_c \frac{c_i}{\max\limits_{j \in CL} c_j} + \beta_k \frac{(\max\limits_{j \in CL} a_j - a_i)}{\max\limits_{j \in CL} a_j} \tag{4}$$

where $\tilde{d}_i = \sum_{j \in X} d_{ij}$, $c_i$ is the capacity of site $i$, and $a_i$ is its cost. The weights of these three factors, $\beta_d, \beta_c$, and $\beta_k$, are all set to 1/3 according to the experimentation described by the authors.

Once a feasible solution $X$ is constructed, the algorithm explores its neighborhood to obtain the corresponding local optimum. This neighborhood is based on swap moves. Specifically, let $d^*$ be the objective function value of solution $X$, and let `pivotal_list` be the set of all sites in $X$ with minimum distance equal to $d^*$. Then, at each iteration, the algorithm evaluates the exchange between a randomly selected site $i$ in the `pivotal_list` with a site $j \in V \setminus X$, with distance to the selected sites larger than $d^*$. We do not include in this computation the site $i$ that we are removing from the solution. The swap movement is applied if it is feasible and it improves the current solution. The local search based on this neighborhood performs iterations while the solution improves.

To complement the GRASP algorithm described above, the authors propose a long-term tabu search for GDP. This is a memory-based methodology that explores efficiently the solution space (Glover, 1989; Glover & Laguna, 1998). The proposed algorithm, `TS`, starts by constructing an initial solution in the same way that the `GRASP` constructive phase, but without applying the randomized feature, thus obtaining a greedy heuristic to construct an initial solution $X$. After that, the algorithm explores the same neighborhood around the current solution as in the improvement method explained above. In this methodology, the algorithm always performs a movement even if it does not improve the solution. After executing an exchange between a site $i \in$ `pivotal_list` and a site $j \in V \setminus X$, the tabu structure records as tabu-active the site $i$, i.e., the selected site that leaves the solution. The tabu status of a site $i$ remains active for a specific number of iteration controlled by a search parameter, and during these iterations, it cannot be selected for inclusion in the solution. Finally, the long term phase in `TS` diversifies the search by exploring unvisited areas of the solution space. This is done by avoiding frequently visited sites and favoring sites which provide high quality objective function values.

The computational experiments performed in Martínez-Gavara et al. (2021) showed that, as expected, the `TI_Ad` heuristic algorithm achieves relatively good solutions for small size instances with very short running times (less than 1 second), but it cannot compete with `GRASP` and `TS` in terms of the quality of the solutions. Furthermore, `GRASP` emerges as the winner with small running times. We can conclude from this revision that `GRASP` is the best method published for the GDP, and therefore, we will compare our heuristics with it in Section 5 to establish its quality relative to the state-of-the-art methods.

## 4. New heuristic methods

As mentioned in the introduction, we can find nowadays a large number of metaheuristic technologies that can be applied to any combinatorial optimization problem to solve it efficiently. They range from bioinspired methods, such as the well-known genetic algorithms, to methods based on artificial intelligence, such as tabu search. In this paper we consider the Variable Neighborhood Search (VNS), a metaheuristic based on exploring several neighborhoods during the solution search to overcome the limitations of local optimality. VNS was introduced in Mladenović & Hansen (1997) and, since then, this methodology has continuously evolved resulting in several extensions and variants. See Hansen et al. (2016) for a thorough analysis.

In this work, we propose a Multi-Start Basic VNS (MS-BVNS) to deal with the Generalized Dispersion Problem (GDP). BVNS combines deterministic and random changes of neighborhood structures in order to find a balance between diversification and intensification. Incorporating BVNS in a multi-start scheme allows us to perform several independent iterations to further improve the diversification ability of the procedure. Algorithm 1 shows the associated pseudo-code.

**Algorithm 1:** $\texttt{MS} - \texttt{BVNS}(it_{\max}, k_{\max})$

```
1  $X_{best} \leftarrow \emptyset$
2  for $t \leftarrow 1$ to $it_{\max}$ do
3      $X \leftarrow Construct()$
4      $X \leftarrow Improve(X)$
5      $k \leftarrow 1$
6      while $k \leq k_{\max}$ do
7          $X' \leftarrow Shake(X, k)$
8          $X'' \leftarrow Improve(X')$
9          $k \leftarrow NeighborhoodChange(X, X'', k)$
10     end
11     $UpdateBest(X, X_{best})$
12 end
13 return $X_{best}$
```

MS-BVNS algorithm takes as input parameters the number of restarts $it_{\max}$ and the largest neighborhood to be explored $k_{\max}$. We assume that the relevant information about an instance is available in all methods. Specifically, an instance contains the distances between sites $d_{ij}$ (with $i, j \in V$), the capacity $\{c_i\}$ and cost $\{a_i\}$ of each site $i$ (with $i \in V$), the maximum storage capacity $B$, and the maximum budget $K$.

The method first initializes the best solution found, $X_{best}$, as an empty set in step 1 of the algorithm. $X_{best}$ will be updated during the search to contain the best solution found so far. At each iteration, $t$, a solution $X$ is generated by considering one of the constructive procedures presented in Section 4.1 (step 3). The solution is locally improved with any of the local search methods described in Section 4.3 (step 4). Then, starting from the first predefined neighborhood (step 5), our VNS method iterates until it reaches the maximum considered neighborhood $k_{\max}$ (steps 6-10). For each neighborhood search step, the incumbent solution is perturbed with the shake method described in Section 4.4 (step 7), generating a solution $X'$ in the neighborhood under exploration. The shake process is designed to escape from local optima. The local search method is then responsible for finding a local optimum $X''$ in the current neighborhood with respect to the perturbed solution $X'$ (step 8). Finally, the neighborhood change method selects the next neighborhood to be explored (step 9). In particular, if $X''$ outperforms $X$ in terms of objective function value, then it is updated (i.e., $X \leftarrow X''$), and the search starts again from the first neighborhood (i.e., $k \leftarrow 1$). Otherwise, the search continues in the next neighborhood (i.e., $k \leftarrow k+1$). The algorithm repeats this strategy for $t_{max}$ iterations, updating the best solution found so far (step 11). The best solution found is finally returned in step 13.

## *4.1. Greedy construction methods*

VNS methods typically consider a random approach to construct an initial feasible solution. However, as it has been recently determined in the associated literature, an initial solution constructed with a more elaborated procedure usually converges faster (Sánchez-Oro et al., 2017; Duarte et al., 2012; Martí et al., 2021) . In this paper we propose two different greedy constructive procedures for the GDP, named `C1` and `C2`, which are able to find high quality solutions in very short computing times. Both constructive methods follow the same scheme but varying the greedy function used to select the next site to be included in the solution. `C1` and `C2` build a solution by iteratively adding one site at a time. Initially, the first site $i \in V$ to be included in the partial solution $X$ is chosen at random from $V$. Then, the selection of the best site is done according to the corresponding greedy function, breaking ties at random.

In particular, `C1` selects the best site according to the greedy function, $g_1$, which depends on distance, capacity, and cost. In mathematical terms, for each site $i \in V \setminus X$, the greedy function is defined as:

$$g_1(i) = d_i \, \frac{\hat{c}_i}{\hat{a}_i} \tag{5}$$

where $d_i := \min_{j \in X} d_{ij}$, $\hat{c}_i := c_i / \max_{j \in V} c_j$, and $\hat{a}_i := a_i / \max_{j \in V} a_j$. The rationale behind the greedy function $g_1$ compared with the previous greedy function proposed in Martínez-Gavara et al. (2021), is threefold: the first motivation is the inclusion of the objective value in the definition of $g_1$, i.e., the minimum value, $d_i$, instead of the sum, $\tilde{d}_i$, although it comes at the cost of evaluating $d_i$ for all sites. The second point is the use of the ratio $\hat{c}_i/\hat{a}_i$, which favours the selection of those sites with large capacities and low cost values. Finally, the third advantage of our proposal is the lack of dependency on extra parameters.

We propose an alternative constructive method `C2` only based on distances to evaluate the incremental contribution of including a specific site. In this procedure, capacity and cost are only considered as constraints. More precisely, given a partial solution $X$, for each site $i \in V \setminus X$, the greedy function in `C2` is defined as $g_2(i) = d_i$. We will compare `C1` and `C2` in our computational testing reported in Section 5.

## *4.2. Auxiliary constructive method*

Note that the GDP is a highly constrained problem due to the inclusion of a minimum capacity and maximum cost, which results in a reduced feasible set of solutions. In fact, we have experimentally found that the two constructive methods proposed above are not able to find feasible solutions in some of the hardest instances. In order to provide good starting points to the local search algorithm, we consider an auxiliary constructive method to build a feasible solution when `C1` or `C2` fail. We denote this method as `Caux` and it only considers the capacity and cost, ignoring distances among selected sites. In particular, `Caux` sorts all the elements in descending order of the ratio between capacity and cost. This function, denoted as $g_{aux}$, is computed as follows:

$$g_{aux}(i) = \frac{c_i}{a_i} \tag{6}$$

for all $i \in V$. Notice that this evaluation and sorting is computed offline only once. It is worth mentioning that the quality of a solution is basically determined by the distances among elements. Therefore, we do not expect to produce high quality solutions with this method but, as stated before, its main goal is to provide initial feasible solutions.

This approach is not new. In fact, Rosenkrantz et al. (2000) introduced a similar strategy for their heuristic to solve the Capacitated Dispersion Problem, in which only the capacity constraint is considered. Similarly, Martínez-Gavara et al. (2021) described a straightforward adaptation of their heuristic to the GPD, where the ratio between capacity and cost is computed. In this paper, we propose a more elaborated strategy, based on performing multiple iterations over the list of ordered sites, starting in each one from a different site until identify a feasible solution.

**Algorithm 2:** `Caux`$(S)$

```
1  $X \leftarrow \emptyset$
2  $cost, cap \leftarrow 0$
3  while $|S| > 0$ do
4      $i \leftarrow first(S)$
5      $X \leftarrow \{i\}$
6      $cost \leftarrow cost + a_i$
7      $cap \leftarrow cap + c_i$
8      $S \leftarrow S \setminus \{i\}$
9      for $j \in S$ do
10         if $(cost + a_j \leq K)$ then
11             $X \leftarrow \{j\}$
12             $cost \leftarrow cost + a_j$
13             $cap \leftarrow cap + c_j$
14         end
15         if $(cost \leq K)$ and $(cap \geq B)$ then
16             return $X$
17         end
18     end
19     $X \leftarrow \emptyset$
20     $cost, cap \leftarrow 0$
21 end
22 return $X$
```

Our method, detailed in Algorithm 2, receives an array of sites $S$ as the input. Without loss of generality, we assume that $S$ is sorted in decreasing order according to Equation (6). After the initialization of the solution $X$ and the associated cost and capacity (steps 1-2), `Caux` scans with a while-loop all the sites in the order given by $g_{aux}$ (steps 3 to 21). Each iteration starts by selecting the element $i$ with the largest ratio (step 4), adding it to the solution under construction (step 5), and updating the corresponding cost and capacity (step 6-7). Then, this element $i$ is removed from the list of sites $S$ for the next iterations (step 8). Then, a for-loop traverses the remaining elements in $S$ (steps 9 to 17), including to the solution those sites that satisfy the cost constraint (steps 10 to 14), and skipping those ones that produce unfeasible solutions. When `Caux` finds a feasible solution (step 15), it is returned (step 16); otherwise, the algorithm performs a new iteration and initializes again $X$, *cost*, and *cap* (step 19-20). Note that, the element $i$ considered in the current iteration is discarded and it is not longer considered in the construction process. Finally, if the algorithm is not able to find a feasible solution, an empty solution is returned in step 22.

## *4.3. Local Search procedure*

Local search methods typically perform the intensification stage of Basic VNS. Their main objective is to find a local optimum with respect to a predefined neighborhood. In general, local search algorithms implement either a first-improvement or a best-improvement selection strategy when scanning the neighborhood. Iterations performed in the former are usually more efficient than those in the best-improvement one, since the first-improvement strategy only evaluates part of the neighborhood and directly applies the first move that results in an improved solution, discarding the examination of the rest of the neighborhood (thus saving the associated computing time). On the other hand, the improvement obtained in the first-improvement strategy is typically smaller than the one achieved by the best-improvement strategy, in which the entire neighborhood is explored, resulting generally in more iterations to obtain the local optimum.

The best improvement strategy is usually more adequate to perform efficient catching and updating mechanisms, which allows the search to explore effectively the neighborhood (Hoos & Stützle, 2005). Hansen & Mladenović (2009) perform an empirical study on the well-known Traveling Salesman Problem to compare first and best improvement strategies. The authors conclude that the first improvement approach is better and faster if initial solutions are constructed at random. However, if the search starts from a greedy solution (as we do in this paper), it is recommended to follow the best improvement strategy. Thus, considering the aforementioned reasons, the local search methods proposed in this paper are based on the best improvement strategy.

Attending to the solution representation described in this paper, we propose three different move operators: *swap*, *add*, and *drop*. Given a solution $X$, the first one consists in replacing an element $i \in X$ with an element $j \in V \setminus X$, producing a new solution $X' \leftarrow swap(X, i, j)$. The neighborhood associated to solution $X$ is formally defined as follows:

$$N_{swap}(X) = \{X' \subseteq V : \; X' = X \setminus \{i\} \cup \{j\}, \, i \in X, \, j \in V \setminus X\} \tag{7}$$

The first local search method proposed in this paper, named as `LS1`, scans $N_{swap}(X)$ as described in Algorithm 3. `LS1` receives a feasible solution and then systematically explores $N_{swap}$ with two for-loops (steps 1 and 2). For the sake of clarity, we denote in this pseudo-code the objective function as $f$ (to be consistent with the definition given in Section 2). Additionally, we assume that we have specific functions to obtain the capacity (`getCap`) and cost (`getCost`) values of a given solution, to check its feasibility.

The GDP is a highly constrained optimization problem. Therefore, `LS1` is not expected to perform a large number of steps. This is why we propose the *add* and *drop* moves, performing a nested exploration of the three neighborhoods, to overcome the difficulties of optimizing such a constrained problem. Given a solution $X$, the add-move consists in selecting an element $j \in V \setminus X$ and inserting it in $X$, producing a new solution $X'$. For the sake of simplicity, we denote this move as $X' \leftarrow add(X, j)$. Thus, given the solution $X$, its neighborhood $N_{add}(X)$ is defined as follows:

$$N_{add}(X) = \{X' \subseteq V : \; X' = X \cup \{j\}, \, j \in V \setminus X\} \tag{8}$$

**Algorithm 3:** `LS1`(X)

```
1 for $i \in V$ do
2     for $j \in V \setminus X$ do
3         $X' \leftarrow swap(X, i, j)$
4         if $(f(X') > f(X))$ and $(getCap(X') \geq B)$ and $(getCost(X') \leq K)$ then
5             $X \leftarrow X'$
6     end
7 end
8 return $X$
```

The next neighborhood is defined with the drop-move. It consists in removing an element $i \in X$, producing a new solution $X'$, (i.e., $X' \leftarrow drop(X, i)$). Then, the associated neighborhood of solution $X$ is:

$$N_{drop}(X) = \{X' \subseteq V : \; X' = X \setminus \{i\}, \, i \in X\} \tag{9}$$

Our second local search `LS2` iteratively applies these two moves, add and drop, to the solution, until both constraints are satisfied. This can be viewed as a repair mechanism, that applies an oscillation pattern (adding and dropping sites iteratively), which generates solutions around the feasibility frontier defined by the problem constraints in the solution space.

The pseudo-code of the local search `LS2` is shown in Algorithm 4. It has the same structure than the one reported above for `LS1`, but including steps from 4 to 8. Specifically, the swap move (step 3) might produce an infeasible solution if it does not satisfy capacity and/or cost constraints (step 4). In particular, if after a *swap* move, $X'$ is unfeasible, since its capacity is below $B$, the *ADDS* procedure (step 5) performs as many *add* moves as possible trying to get a feasible solution. More precisely, for an unfeasible solution $X'$, *ADDS* scans sites in $V \setminus X'$, discarding those ones that their inclusion would exceed the budget $K$ in the cost constraint, and incorporating the site that produces the largest improvement in the objective function.

The procedure *DROPS* (step 7) considers the situation when the *swap* move produces a solution that satisfies the capacity constraint. As in the aforementioned case, it is performed as much drop moves as possible trying to obtain a better and feasible solution. Specifically, for a solution $X'$, *DROPS* scans sites in $V \setminus X'$, discarding those ones that their inclusion would violate the capacity constraint, and finally incorporating the site that produces the largest improvement in the objective function.

**Algorithm 4:** `LS2`(X)

```
1  for $i \in V$ do
2      for $j \in V \setminus X$ do
3          $X' \leftarrow swap(X, i, j)$
4          if $(getCap(X') < B)$ and $(getCost(X') \leq K)$ then
5              $X' \leftarrow ADDS(X')$
6          else
7              $X' \leftarrow DROPS(X')$
8          end
9          if $(f(X') > f(X))$ and $(getCap(X') \geq B)$ and $(getCost(X') \leq K)$ then
10             $X \leftarrow X'$
11     end
12 end
13 return $X$
```

The objective of the GDP consists in maximizing a minimum value. Consequently, there may be many different solutions with the same objective function value. In other words, the solution space presents a "flat landscape" (Della Croce et al., 2009) where most of the moves that can be performed to modify a solution, have a null objective function value, and cannot be classified as improving moves. To overcome this lack of information to guide a local search, we extend the meaning of improving move. In particular, when two solutions have the same objective function, we use an alternative evaluation function to break ties. Given a solution $X$ we define $f_{alt}(X)$ as the sum of distances of elements $i \in X$. In mathematical terms:

$$f_{alt}(X) = \sum_{i,j \in X, i \neq j} d_{ij}. \tag{10}$$

We show in Algorithm 5 the pseudo-code of the local search method `LS3` based on this alternative evaluation function. It has the same structure than `LS1` and `LS2`, but including steps from 11 to 14. Specifically, in step 11 if the feasible solution $X'$ in the neighborhood of $X$ has the same objective function value, we evaluate the alternative objective function (step 12). In case that we find an improvement according to this new function, the incumbent solution is updated (step 13); otherwise, the local search method performs a new iteration.

**Algorithm 5:** LS3(X)

```
for $i \in V$ do
    for $j \in V \setminus X$ do
        $X' \leftarrow swap(X, i, j)$
        if $(getCap(X') < B)$ and $(getCost(X') \leq K)$ then
            $X' \leftarrow ADDS(X')$
        else
            $X' \leftarrow DROPS(X')$
        end
        if $(f(X') > f(X))$ and $(getCap(X') \geq B)$ and $(getCost(X') \leq K)$ then
            $X \leftarrow X'$
        else if $(f(X') = f(X))$ and $(getCap(X') \geq B)$ and $(getCost(X') \leq K)$ then
            if $f_{alt}(X') > f_{alt}(X)$ then
                $X \leftarrow X'$
            end
    end
end
return $X$
```

### *4.4. Shake procedure*

The perturbation mechanism in VNS is usually called the Shake procedure. The goal of this method is to diversify the search by generating a neighbor solution relatively different to the current one. We propose a method that modifies the structure of the solution according to a parameter $k$. Its value ranges from 1 to $k_{max}$, which is an input parameter of the complete procedure (see Algorithm 1).

According to the moves defined in Section 4.3, the Shake procedure could be implemented as a sequence of *swap* moves or, alternatively, as *drop* followed by *add* moves (symmetrically, *add* and *drop* moves). Considering that we are considering here a highly constrained optimization problem, we have discarded *swap* moves, since the number of feasible moves (those that produce a feasible solution) is relatively low, and thus not adequate for a diversification method.

We propose a shake method based on dropping a percentage $k$ of the elements in the solution. As it is customary in the BVNS methodology, these elements are selected at random. Then, to recover feasibility, we add to the partial solution obtained when these elements are removed, as many elements as necessary with the *add* moves described above. Recent works have studied more advanced shake techniques balancing the diversification and intensification of the search. This strategy has been referred to as intensified shake (see Duarte et al. (2014) and Sánchez-Oro et al. (2014) for further details).

## 5. Computational experiments

In this section we described the numerical experiments carried out to analyze the performance of the mathematical models and the metaheuristic procedures presented in the previous sections. In particular, we consider two mathematical models adapted for GDP and solved with `Gurobi`: the linear model introduced by Kuby (2), named as `Standard Model`, and the new model (3), named as `Compact Model`; the previous algorithm `GRASP` described in Section 3; and finally, our metaheuristic based on a multi-start basic VNS, `MS-BVNS`.

After introducing the experimental setup, which includes the description of the instances used in our study, we divide the experimentation into three parts. In the first one, we empirically compare the two models presented in Section 2. In the second part, we set the values of the search parameters of our `MS-BVNS` algorithm. Finally, in the third part we perform the comparative study among heuristic algorithms.

### *5.1. Experimental setup*

`GRASP` and `MS-BVNS` are implemented in Java. The experiments are conducted on a computer with a 2.8 GHz Intel Core i7 processor with 16 GB of RAM. The executable files and the individual results for future comparisons are available on: `https://grafo.etsii.urjc.es/GDP`.

Martínez-Gavara et al. (2021) proposed a benchmark set of 200 instances for the GDP. The authors generated them from a subset of 50 MaxMin instances, with different features of size and type, selected from the MDPLIB (Martí & Duarte, 2010). A GDP instance is obtained from each of these instances by adding a capacity $c_i$ and cost $a_i$ for each element $i$. In particular, these values are randomly generated with a uniform distribution between 1 and 1000, and $c_i/2$ and $2\,c_i$, respectively. Then, the minimum required capacity $B$ is computed as the total capacity multiplied by a factor $\varphi_b$ of 0.2 or 0.3, as well as the maximum budget $K$, which is calculated as the sum of all costs values multiplied by a factor $\varphi_k$ of 0.5 or 0.6.

Computational experiments in Martínez-Gavara et al. (2021) suggest that their instances are relatively easy to solve. We performed a preliminary experiment to confirm this point. In particular, we test how the cost constraints reduce the search space, thus making the search exploration more complicated. To do that, we compare the optimal solutions obtained with the mathematical model (2) with and without the cost constraint. So, we run `Gurobi` in the benchmark set with a time limit of 1 hour per instance, and the optimal solution values obtained in both models are the same. Therefore, we confirm that the cost constraints are not active in the optimal solutions, thus making this set of instances somehow easy to solve, so they are not suitable to compare solving methods for this problem as they stand. To make them more challenging, and adequate for comparison of solving methods, we recompute the maximum budget $K$ as the total cost multiplied by a factor $\varphi_k$ of 0.2 and 0.3 instead of 0.5 or 0.6. We repeat the same empirical analysis than before, and we observe that the optimal solutions obtained between both models are different, so this proves that the recomputed $K$ values affect the solution.

To complement the study above and disclose the influence of the $\varphi$ factors in the solution, we graphically analyze the differences between the optimal solutions obtained with the mathematical model (2), and those reached without the cost constraint for GKD-b instances ($n = 50$ and $n = 150$). Figure 1 shows the percentage of the relative difference of the two solution values computed as:

$$\%\ rel = \frac{|z_g - z_c|}{z_c} \cdot 100, \tag{11}$$

where $z_g$ is the optimal solution value obtained with model (2) and $z_c$ is the optimal one without considering the cost constraint in the optimization model. The box-plots in Figure 1 are grouped by the different capacity and cost factor values: $(\varphi_b, \varphi_k) = (0.2, 0.2)$, $(0.2, 0.3)$, $(0.3, 0.2)$, and $(0.3, 0.3)$. As can be observed, instances with $\varphi_b = 0.3$ and $\varphi_k = 0.2$ produce large differences between both solutions. In particular, this combination of factor values generates the most restrictive set of instances. Furthermore, we can see how the percentage ratio between the two solutions is always larger than zero, except for $(\varphi_b, \varphi_k) = (0.3, 0.3)$, meaning that in those instances the structure of both solutions are different. In this last case, the pairwise Wilcoxon statistical test with a $p$-value equal to 0.01427 ($< 0.05$) provides evidence to support that, for $(\varphi_b, \varphi_k) = (0.3, 0.3)$, both solutions are not likely to be the same. So, we set 0.2 and 0.3 as appropriate factor values.

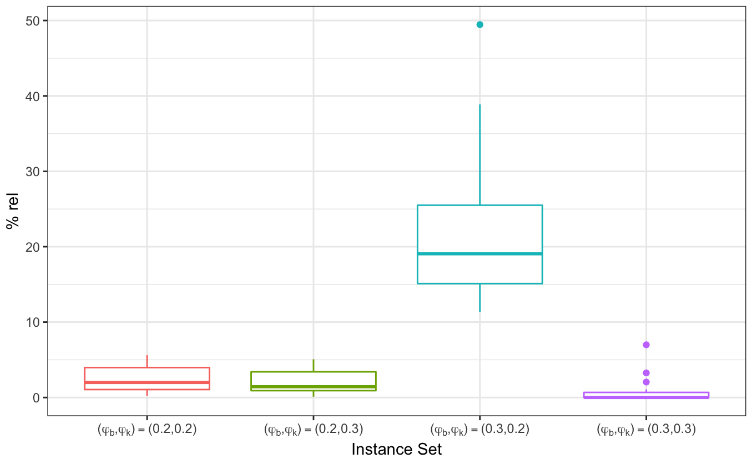


Figure 1: Boxplot in the analysis of the parameters $\varphi_b$ and $\varphi_k$.

Additionally, we have generated a new set of large instances. In particular, we consider 10 new instances in the Euclidean set (GKD-d) with $n = 2000$. The capacity and cost values are generated as described above, thus for each original instance, 4 instances are created. The set of instances we use is available online in the MDPLIB 2.0 (Martí et al., 2021a), and we detail them in Table 1.

Table 1: Benchmark set.

| **Type** | **#inst.** | **Sites ($n$)** | **Capacity factor ($\varphi_b$)** | **Cost factor ($\varphi_k$)** |
|---|---|---|---|---|
| | | 120 instances solved with `Gurobi` | | |
| GKD-b | 80 | $50, 150$ | $0.2, 0.3$ | $0.2, 0.3$ |
| SOM | 40 | 50 | $0.2, 0.3$ | $0.2, 0.3$ |
| | | 80 instances in the heuristic comparison and `Gurobi` | | |
| GKD-c | 40 | 500 | $0.2, 0.3$ | $0.2, 0.3$ |
| MDG-b | 40 | 500 | $0.2, 0.3$ | $0.2, 0.3$ |
| | | 40 instances in the heuristic comparison | | |
| GKD-d | 40 | 2000 | $0.2, 0.3$ | $0.2, 0.3$ |

We study the following statistics to evaluate the performance of the models and algorithms in the computational experiments:

- $\overline{m}$: the average of the minimum distance value in the best solution found;

- *% dev*: the average percent deviation with respect to the best solution found in the experiment. This measure shows how far the heuristic is to the best value. Smaller percent deviations therefore mean better results. This percentage value is computed as

$$\frac{m_a - m_{best}}{m_{best}} \cdot 100, \qquad (12)$$

  where $m_a$ is the objective function value of the solution with the heuristic algorithm $a$ and $m_{best}$ is the best solution found on a given instance in the experiment;

- *#best*: the number of best solutions found;

- *#opt*: the number of optimal solutions found;

- *% gap*: the average percent deviation of the objective function value of the heuristic method with respect to the optimal solution.

- *time*(*s*): average total time in seconds to execute the method.

## *5.2. Preliminary experiments*

In this section, a series of preliminary experiments were conducted to determine the values for the critical search parameters, and to compare the different designs in our heuristics. From the 240 instances in our benchmark, we employed 44 representative instances (with size less than or equal to 500) in the parameter tuning to prevent the over training of the algorithms. We called this subset of instances the *training set*, in contrast to the *testing set*, which contains the entire benchmark.

In the first preliminary experiment, and in order to provide a baseline for comparison, we add a constructive procedure in which the solutions are built with the random selection of the elements (sites). We will refer to it as `C0`. As in the constructive methods `C1` and `C2`, the auxiliary constructive method (`Caux`) is applied if it generates infeasible solutions. Recall that `Caux` is only applied if the construction procedure is not able to find at least one feasible solution. The first experiment is devoted to analyze the effectiveness of this auxiliary construction procedure. The 44 instances in the *training set* were solved with the constructive methods applied for 100 iterations. Table 2 presents, on average, the percentage of feasible solutions that each constructive procedure is able to find. Results show that `C1` obtains a remarkable 98% of feasible solutions in this set of instances. In fact, in just one instance the algorithm requires the application of `Caux`. Constructive procedures `C0` and `C2` present on the other hand a moderate performance, since they are on average capable to obtain around a 50% of feasible solutions. Table 2 reveals that `C0` and `C2` are not able to find feasible solutions in the set of instances with factor values $(\varphi_b, \varphi_k) = (0.3, 0.2)$. In this set, `Caux` has to be applied to obtain an initial feasible solution. It is clear then that `Caux` is necessary to generate a starting point of the algorithm in those cases where the constructive procedure fails to obtain a feasible solution. For the sake of simplicity, and for now on, we will use `C0`, `C1`, and `C2` also to refer the constructive procedure coupled with `Caux`, whenever there is no ambiguity among them.

In our second preliminary experiment, we compare our two constructive procedures with `C0` in the training set of instances. Table 4 summarizes the statistics for each instance set, grouped by the different capacity and cost factor values $(\varphi_b, \varphi_k)$ (11 instances of each type). Results indicate that on average `C2` obtains the best percent deviation value (10.51%), and

Table 2: Average percent of feasible solutions in 100 constructions.

| $\varphi_b$ | $\varphi_k$ | C0 | C1 | C2 |
|---|---|---|---|---|
| 0.2 | 0.2 | 57 % | 100 % | 60 % |
|  | 0.3 | 100 % | 100 % | 100 % |
| 0.3 | 0.2 | 0 % | 91 % | 0 % |
|  | 0.3 | 57 % | 100 % | 50 % |
| summary |  | 53 % | 98 % | 52 % |

0.00 means less than 0.001

it is able to identify 35 (out of 44) best solutions in the experiment. However, C1 obtains the best solutions in the experiment for $(\varphi_b, \varphi_k) = (0.3, 0.2)$. Note that this combination of factor values produces the most restrictive instances, i.e., the solution space is relatively small, thus making them hard to solve. In this case, what emerges from Table 4 is that a greedy function based on the different characteristics of each site: distance, capacity, and cost, is probably the best option to create a constructive method for these hard instances.

Table 3: Comparison of the construction procedures in 100 constructions (in the *training set*).

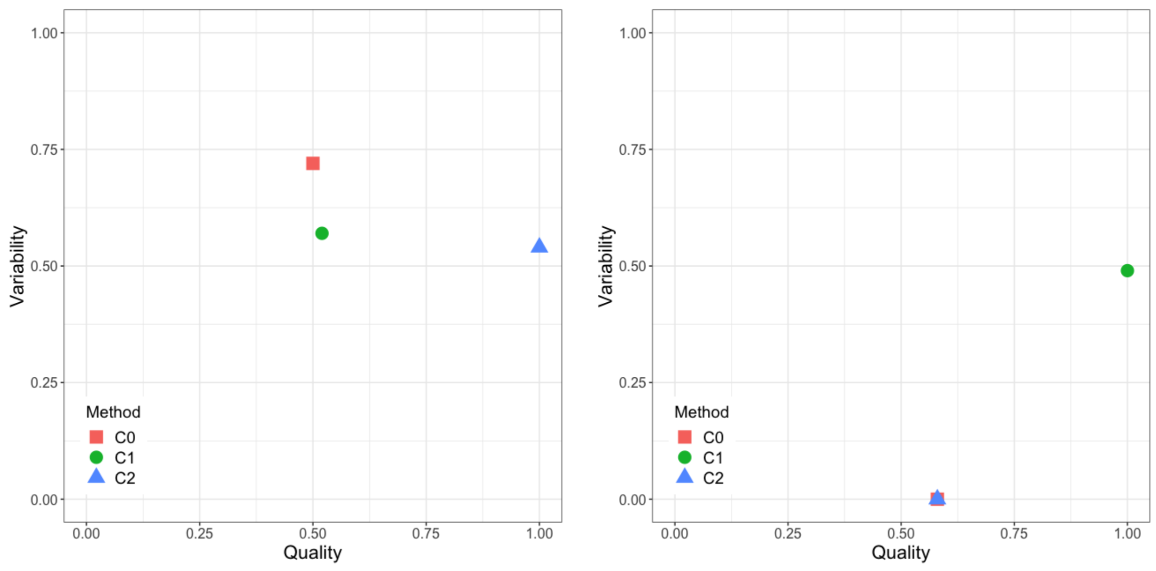
|  |  | C0 |  |  | C1 |  |  | C2 |  |  |
|---|---|---|---|---|---|---|---|---|---|---|
| $\varphi_b$ | $\varphi_k$ | % *dev* | #*best* | *time*(*s*) | % *dev* | #*best* | *time*(*s*) | % *dev* | #*best* | *time*(*s*) |
| 0.2 | 0.2 | 46.15 | 0 | 0.09 | 45.68 | 0 | 1.67 | 0.00 | 11 | 0.74 |
|  | 0.3 | 47.70 | 0 | 0.09 | 45.74 | 0 | 1.66 | 0.00 | 11 | 0.76 |
| 0.3 | 0.2 | 42.02 | 2 | 0.09 | 0.00 | 11 | 3.57 | 42.02 | 2 | 0.78 |
|  | 0.3 | 57.64 | 0 | 0.12 | 51.24 | 1 | 3.42 | 0.00 | 11 | 1.50 |
| summary |  | 48.38 | 2 | 0.10 | 35.67 | 12 | 2.58 | **10.51** | **35** | **0.94** |

0.00 means less than 0.001

Table 4: Comparison of the construction procedures in 100 constructions (in the *training set*).

Since the previous greedy function (4) and the greedy evaluation of the C1 procedure (5) depend on three factors (distance, capacity, and cost), we study the differences between both evaluation functions in terms of % *dev* and #*best* in the instances of the training set with size 500. The results obtained in this experiment reveal that C1 outperforms the previous evaluation function with an average percentage deviation of 5.40% (2.94% in the subset $(\varphi_b, \varphi_k) = (0.3, 0.2)$), in contrast to 47.30% (83.50%) for the previous greedy function. In fact, the previous constructive method is not able to obtain feasible solutions in 3 instances out of 5 in the class with $(\varphi_b, \varphi_k) = (0.3, 0.2)$. We do not reproduce here the table for this experiment for the sake of simplicity, but we conclude that our new evaluation C1 based on the product of the three factors, performs better than the previous one based on their sum.

We now compare the three constructive methods, C0, C1, and C2, with the auxiliary constructive procedure, in quality and variability considering that the role of the construction method in the entire solving algorithm is to provide not only good solutions, but also scattered in the search space in a way that they are good initial points for the local search method. We compute the variability as the number of different solutions in 100 independent constructions. The quality is defined in the range $[0, 1]$ by $1 - (\%\ dev/100)$, where an average value of 1 means that this method captures the best solutions in the experiment, and an average value



a) $(\varphi_b, \varphi_k) \neq (0.3, 0.2)$ b) $(\varphi_b, \varphi_k) = (0.3, 0.2)$

Figure 2: Comparison of quality and variability of the constructive methods

close to 0 indicates low quality solutions. The values of variability have been normalized to fall between 0 and 1 as well. Figure 2 shows the average results in two types of instances; depending whether $(\varphi_b, \varphi_k) = (0.3, 0.2)$ or not. C2 presents the best trade-off between quality and variability when $(\varphi_b, \varphi_k) \neq (0.3, 0.2)$. However, in the instance set with a small solution space, C1 attains the largest quality and variability compared with the other two methods. The variability shown in Figure 2-b reveals that both constructive methods, C0 and C2, are not able to find feasible solutions in this class of instances $((\varphi_b, \varphi_k) \neq (0.3, 0.2))$, so the solution is uniquely determined by the auxiliary construction, producing poor-quality results. We therefore cannot conclude at this stage that one method is better than the other, and perform an additional experiment to evaluate them when coupled with the improvement phase.

The next experiment is performed to test the ability of the constructive procedures C1 and C2 to produce local optima once the local search has been applied. To do that, the constructive procedures are coupled with the basic local search based on swaps, LS1. Results in Table 5 show a similar performance of C1 and C2 as in the previous experiments. C2 coupled with LS1 is able to reduce the average of the percent deviation of C1+LS1 in almost 15% in the instance class $(\varphi_b, \varphi_k) \neq (0.3, 0.2)$, i.e., the percentage deviation is diminished from 42.02% in Table 4 to 27.75% in Table 5. The results on the entire training set indicate that C2+LS1 obtains better outcomes than C1+LS1. In fact, more than 80% of best solutions are due to C2+LS1 with a percent deviation of 8.16%. So, we can conclude that, on average, C2 is the best method to construct an initial solution.

In the next experiment of this section, we analyze the influence of the different local search procedures described in Section 4.3. For the sake of simplicity, we do not reproduce the summary table here. As expected, results show that C2+LS3 clearly obtains the best outcomes in the experiment. In particular, C2+LS3 obtains 36 out of 44 best solutions, in contrast to the 10 and 23 of C2+LS1 and C2+LS2, respectively. In terms of % *dev*, C2+LS3 also exhibits the best value compared with those obtained by C2+LS1 and C2+LS2 (0.36 % vs

Table 5: Performance comparison of the construction procedures with `LS1` in 100 constructions.

| | | `C1+LS1` | | | `C2+LS1` | | |
|---|---|---|---|---|---|---|---|
| $\varphi_b$ | $\varphi_k$ | *% dev* | *#best* | *time(s)* | *% dev* | *#best* | *time(s)* |
| 0.2 | 0.2 | 37.30 | 0 | 3.51 | 0.00 | 11 | 5.52 |
| | 0.3 | 34.03 | 1 | 3.65 | 0.00 | 11 | 7.27 |
| 0.3 | 0.2 | 0.00 | 11 | 11.03 | 27.75 | 5 | 8.67 |
| | 0.3 | 39.68 | 2 | 8.73 | 4.89 | 9 | 10.96 |
| summary | | 27.75 | 14 | **6.73** | **8.16** | **36** | 8.10 |

0.00 means less than 0.001

10.69 % and 9.04 %, respectively). For this reason, the local search used in the final `MS-BVNS` algorithm is `LS3`.

The last experiments are devoted to fine-tune the search parameters involved in the `MS-BVNS` algorithm, $k_{max}$ and $it_{max}$. In a VNS framework, the diversification of the search process is controlled by the search parameter $k_{max}$, which indicates the percentage of selected elements to be dropped from the solution. For the sake of simplicity, we do not show the results of these experiments in which we evaluated the average objective function values ($\overline{m}$) for different $k_{\max}$ and $it_{\max}$ values. As expected, large values of $k_{max}$ generate better quality solutions, with the drawback of larger CPU times. Similarly, the objective function value improves significantly with a CPU time less than 60 seconds for $it_{\max} \leq 5$ and it does not for $it_{\max} > 8$. We therefore select $k_{max} = 0.5$ and $it_{\max} = 8$ for their good trade-off between solution quality and running time.

In the final preliminar experiment, we undertake to evaluate the relative contribution of each element of the algorithm to the final solution in one iteration of the variable neighborhood algorithm. In particular, Figure 3 represents the evaluation of the value of the best solution found in a given iteration on a 500 element instance when the different elements of our algorithm are applied. It shows the value of the current solution obtained with the constructive method `C2`, the auxiliary constructive method (`Caux`), the local search (`Local Search`), and the variable neighborhood search (`VNS`). In this way, we may observe the evolution of the incumbent solution in the search process. Additionally, we depict with a square red symbol the infeasible solutions generated during the search, and with a blue circle symbol the feasible ones. Figure 3 shows that the constructive method generates an infeasible solution. This is to be expected considering the difficulty to achieve feasibility in this problem due to its two constraints. Then, the repair method is able to improve the solution value in the process of making it feasible. The local search on the other hand, presents an oscillation pattern between feasible and not feasible solutions that turns out to be very effective on allocating good feasible solutions. In the last stage of the search, the `VNS` marginally improves the output solution of the local search, alternating between feasible and infeasible solutions. Note that, in some iterations, this oscillation produces solutions with the same objective function value but it is able to repair the not feasible ones, thus obtaining good feasible solutions. Although this diagram only represents a single instance, we repeat this experiment with other challenging instances in our test-bed obtaining a similar profile.

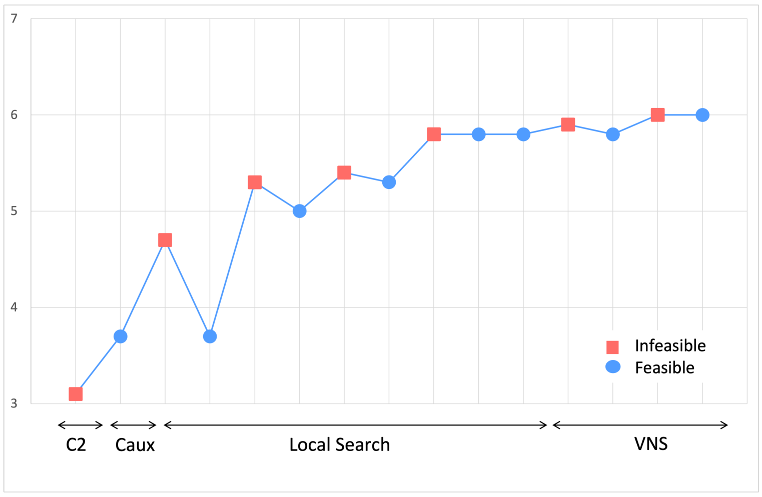


Figure 3: Contribution of search methods.

## *5.3. Mathematical models*

This section is devoted to the comparison of the mathematical models described in Section 2. Specifically, we consider model (2), named `Standard Model`, and our adaptation (3) of the model by Sayah and Irnich to GDP, named `Compact Model`. In the first experiment, we compare both models in the instances with size 50 and 150, on sets GKD-b and SOM. We do not reproduce here the table of this experiment because both formulations are able to optimally solve instances with size $n = 50$ in less than 1 second, and the GKD-b instances of size 150 in less than 30 seconds. In this case, the results of both models are identical.

In the next experiment, we run each model for a time limit of 3600 seconds on each medium size instance ($n = 500$). Results in Table 6 show that the `Compact Model` is able to optimally solve 32 out of the 40 GKD-c instances (*#opt*), and to obtain on average a 87.5% of best solutions in this set. However, in the case of MDG-b set, the `Standard Model` achieves the maximum number of best solutions (40 out of 40). There exist significant differences between both models with a p−value ($5.762 \cdot 10^{-6}$) less than 0.01 in the Wilcoxon signed rank test. Note that both models are able to optimally solve the most restrictive instances, those with $\varphi_b = 0.3$ and $\varphi_k = 0.2$. The computing time is on average less than 200 seconds per instance in the GKD-c set, and 600 seconds in the case of MDG-b instances. `Gurobi` is a powerful mathematical optimization solver that uses an advanced pioneering branch-and-cut algorithm. In fact, the developers of this solver have introduced new kinds of cuts that are very efficient in this type of problems. This could explain that even with a small number of feasible solutions in these restrictive sets, `Gurobi` is able to solve both linear programming models. It must be noted that this is not necessarily true for heuristics, for which a reduced search space with a low number of feasible solutions, may create a difficult problem, for which finding feasible solutions can be a difficult task.

In conclusion, our empirical analysis states that both models can optimally solve small size instances ($n \leq 150$) in a very short time (less than 30 seconds). Additionally, both models solve medium size instances ($n = 500$) with `Gurobi` within a reasonable execution time (less than 3600 seconds), where the `Compact Model` is able to certify the optimality of some solutions in this set. Considering the good performance of the models on small and medium size instances, we run an additional experiment with large instances ($n = 2000$). As

Table 6: Comparison of the mathematical models ($n = 500$).

| | | | `Standard Model` | | | | `Compact Model` | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instances | $\varphi_b$ | $\varphi_k$ | *%dev* | *#best* | *#opt* | *time(s)* | *%dev* | *#best* | *#opt* | *time(s)* |
| GKD-c | 0.2 | 0.2 | 2.13 | 1 | 0 | 3600 | 0.00 | 10 | 9 | 1871 |
| | | 0.3 | 2.34 | 1 | 0 | 3600 | 0.00 | 10 | 10 | 1171 |
| | 0.3 | 0.2 | 0.00 | 10 | 10 | 151 | 0.00 | 10 | 10 | 123 |
| | | 0.3 | 0.12 | 9 | 0 | 3600 | 2.77 | 5 | 3 | 3390 |
| | summary | | 1.15 | 21 | 10 | 2738 | **0.69** | **35** | **32** | **1639** |
| MDG-b | 0.2 | 0.2 | 0.00 | 10 | 0 | 3600 | 10.81 | 0 | 0 | 3600 |
| | | 0.3 | 0.00 | 10 | 0 | 3600 | 8.00 | 0 | 0 | 3600 |
| | 0.3 | 0.2 | 0.00 | 10 | 10 | 243 | 0.00 | 10 | 10 | 587 |
| | | 0.3 | 0.00 | 10 | 0 | 3600 | 10.92 | 0 | 0 | 3600 |
| | summary | | **0.00** | **40** | **10** | **2761** | 7.43 | 10 | **10** | 2847 |

0.00 means less than 0.001

expected, none of these two models is able to solve them within a time limit of 3600 seconds and they even encounter difficulties to identify a feasible solution. We therefore have to resort to metaheuristics to find good solutions in moderate running times for medium and large size instances.

## *5.4. Comparative testing*

The aim of this section is twofold. On one hand, the comparison of our `MS-BVNS` heuristic with the solutions obtained with the `Compact Model` with `Gurobi` for the medium size instances ($n = 500$). On the other hand, we compare our `MS-BVNS` algorithm with the previous heuristic approach, `GRASP`, in medium and large size instances ($n = 500$ and $n = 2000$). The last experiment in Section 5.2 shows that the best configuration for `MS-BVNS` is $it_{\max} = 8$. However, `GRASP` and the `Compact Model` with `Gurobi` are run with a time limit of 60 seconds and 3600 seconds, respectively. Therefore, for a fair of comparison, `MS-BVNS` is set up to stop with a similar time limit criterion.

In the first experiment, we test the ability of `MS-BVNS` to match the optimal values for those instances where we know them (or the best upper bound known). We consider our `MS-BVNS` approach for different time horizons: 60, 450, and 900 seconds. Figure 4 shows the percentage deviation value ($y$-axis) of the solutions obtained with `MS-BVNS` with respect to the optimal values (or bounds) in the different time limits ($x$-axis). We can observe that on average (black cross mark), the % *dev* decreases from 22% in 60 seconds to 13% in 900 seconds. It is clear that our heuristic has difficulties to match the optimal solution in the running times considered. However, using `MS-BVNS` heuristic has the advantage that it reaches good solutions in short running times.

Note that the linear relaxation of the formulation (where the binary definition for the $x$ variables is replaced with the constraints $0 \leq x \leq 1$) can be considered weak, since it provides a solution with many variables taken a fractional value equal to 0.5. In line with that, we have empirically found that its associated objective function value is far from the optimal solution value (with an average percentage deviation around 150%), and therefore we need to resort to the branch and bound (implemented in Gurobi in our case) to obtain the optimal value to assess the merit of the heuristics.

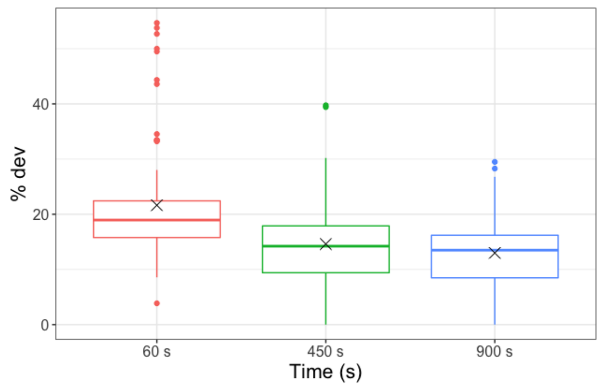


Figure 4: Boxplot in the comparison of `MS-BVNS` and `Compact Model` on 3600s ($n = 500$).

Table 7: Comparison of heuristic solutions in medium (80 inst.) and large instances (40 inst.) on 60 seconds.

| | | | `GRASP` | | | `TS` | | | `MS-BVNS` | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instance size | $\varphi_b$ | $\varphi_k$ | $\nu$ | *% dev* | *#best* | $\nu$ | *% dev* | *#best* | *% dev* | *#best* |
| 500 | 0.2 | 0.2 | 20 | 53.08 | 0 | 20 | 46.95 | 0 | 0.00 | 20 |
| | | 0.3 | 20 | 51.34 | 0 | 20 | 46.88 | 0 | 0.00 | 20 |
| | 0.3 | 0.2 | 15 | 46.13 | 1 | 15 | 48.13 | 1 | 2.87 | 18 |
| | | 0.3 | 20 | 54.00 | 0 | 20 | 54.25 | 0 | 0.00 | 20 |
| | summary | | 75 | 51.14 | 1 | 75 | 49.05 | 1 | **0.71** | **78** |
| 2000 | 0.2 | 0.2 | 10 | 81.22 | 0 | 10 | 2.40 | 8 | 6.68 | 3 |
| | | 0.3 | 10 | 81.26 | 0 | 10 | 1.74 | 8 | 5.67 | 2 |
| | 0.3 | 0.2 | 0 | - | - | 0 | - | 0 | 0.00 | 10 |
| | | 0.3 | 10 | 79.51 | 0 | 10 | 28.17 | 0 | 0.00 | 10 |
| | summary | | 30 | 80.67 | 0 | 30 | 10.77 | 16 | **3.09** | **25** |

0.00 means less than 0.001

In the last experiment, we consider the 80 medium size instances ($n = 500$) and the 40 large ones ($n = 2000$) to evaluate the performance of our heuristic `MS-BVNS` in comparison with the previous `GRASP` and `TS` (Martínez-Gavara et al., 2021). In large instances ($n = 2000$), the mathematical model is not able to obtain competitive results or even find feasible solutions, so that, we do not have optimal values to compare with. We run the three heuristics for 60 seconds on each instance and report the fraction of instances in each set in which they are able to find feasible solutions. Let $\nu$ be the number of feasible solutions found. We also report the average deviation with respect to the best solution found in the experiment. Additionally, we report the number of instances in which each method finds the best solution.

The results of this experiment clearly show the superiority of our heuristic that systematically obtains lower deviations and larger number of best solutions than the previous `GRASP` and `TS` methods. Additionally, results in Table 7 show that `GRASP` and `TS` are not able to obtain feasible solutions in the instance class $(\varphi_b, \varphi_k) = (0.3, 0.2)$ in 5 out of 20 instances of size 500, and in all the 10 instances of size 2000 of this class, as shown in the $\nu$ column. On the other hand, `TS` is able to obtain high quality solutions for the large size instances with $\varphi_b = 0.2$; while quality of `GRASP` solutions deteriorates as the instances size increases.

Wilcoxon statistical tests are applied to the individual results, and the $p$-values ($2.454 \cdot 10^{-14}$ and $1.863 \cdot 10^{-9}$) are in both cases less than the significance level 0.01, which indicates the superiority of `MS-BVNS` with respect to the previous heuristics.

To sum it up, the experimentation shows that `MS-BVNS` provides the best trade-off between solution quality and computational running times for medium and large size instances ($n \geq 500$).

## *5.5. A case study*

We consider in this section a real case based on the data set described in Daskin (2011) and Snyder & Daskin (2005), which consists of the capitals of the continental United States plus Washington, DC., from which we have to select some of them to provide a given service. In particular, we adapted it to obtain a realistic example for our facility location problem.

This is the case of a company that offers a specific service through their chain of facilities, and it wants to expand its business internationally by opening new branch offices in some of the 49 US cities considered, to launch new innovative product lines in this competitive market. In technical terms, each site $i$ has associated the cost $a_i$ of opening and operating a facility (branch office), together with its capacity of service $c_i$. This data is given in Daskin (2011), where the capacity of each facility is computed from the customer demands in each city, and it is proportional to their population. In particular, the associated capacity $c_i$ represents the number of customers that can be attended in the potential facility at city $i$. To complete the data, the company has a maximum budget $K$ of \$1150000 to open up markets in US, and it wants to provide service to a minimum of 5000 customers in all their new subsidiaries. The goal is to select the sites as disperse as possible all around the country in order to gain visibility for the new affiliates and open up new market opportunities in the US.

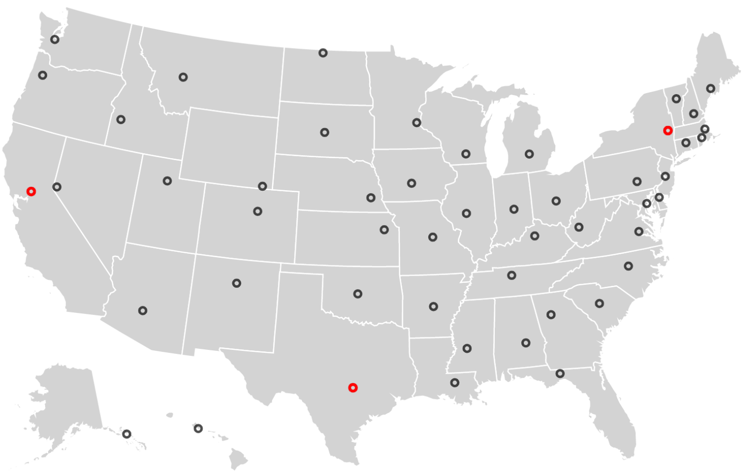


Figure 5: Potential locations (black circles) and selected facilities (red circles).

Figure 5 shows a map of US with the 49 potential sites (black circles) considered by the company to establish new branch offices. Our `MS-BVNS` algorithm provides the company with the best strategical decision to reinforce their international presence, given by opening new offices in Sacramento (CA), Albany (NY), and Austin (TX) (depicted with red circles in the figure). As shown in the figure, locating a branch office in Sacramento (CA) covers the demands of the Western United States' customers. Similarly, the office in Austin (TX) provides service to Central United States, and the one in Albany (NY) to those in the eastern part of the country. At the same time, the constraints in the model assure that the required level of service is provided without exceeding the budget. In conclusion, solving the model results in a selection of locations that better represents the company interests in both reaching a large target audience and satisfying its operational requirements.

## 6. Conclusions

In this paper we explore the adaptation of mathematical models and metaheuristic methodologies to a practical variant of the well-known maximum diversity problem. In particular, we target the case in which diversity maximization is subject to two side constraints; namely cost and capacity. This variant, known as the generalized dispersion problem (GDP), models many practical situations in location of facilities both public or private, such as hospitals or franchise chains.

In this paper, we discuss the adaptation of a previously proposed model for the unconstrained case to the GDP, and the development of a heuristic algorithm based on the VNS methodology. Our experimentation reveals that both methods are very effective to solve this problem, each one in a different way. The mathematical model is able to solve small instances to optimality in short running times; however, as it is usually the case with mathematical programming models, its performance significantly deteriorates when it targets large instances. On the other hand, our VNS obtains good solutions in large instances within short computational times, outperforming the previous heuristic proposed for this problem. In this way, our two proposals complement each other, in line with previous research in similar combinatorial optimization problems.

The use of intelligent optimization algorithms to solve this NP-hard problem constitutes a viable way to find the optimal or approximate solution within a reasonable time. Trajectory-based metaheuristics (Glover, 1989), population-based algorithms (Martí et al., 2021) or bio-inspired algorithms (Wang et al., 2009, 2018; Li et al., 2020) may be applied as well to solve this challenging problem. We hope that this paper triggers the interest of researchers to apply them.

**Acknowledgments.** This work has been partially supported by the Spanish Ministerio de Ciencia, Innovación y Universidades (MCIU/AEI/FEDER, UE) with grant ref. PGC2018-095322-B-C21 and PGC2018-095322-B-C22.

## References

Daskin, M. S. (2011). *Network and discrete location: models, algorithms, and applications*. John Wiley & Sons.

Della Croce, F., Grosso, A., & Locatelli, M. (2009). A heuristic approach for the max–min diversity problem based on max-clique. *Computers & Operations Research*, *36*, 2429–2433.

Duarte, A., Escudero, L. F., Martí, R., Mladenović, N., Pantrigo, J. J., & Sánchez-Oro, J. (2012). Variable neighborhood search for the vertex separation problem. *Computers & Operations Research*, *39*, 3247–3255.

Duarte, A., Pantrigo, J. J., Pardo, E. G., & Mladenović, N. (2014). Multi-objective variable neighborhood search: an application to combinatorial optimization problems. *Journal of Global Optimization*, *63*, 515–536.

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

Feng, Y., Deb, S., Wang, G.-G., & Alavi, A. H. (2021). Monarch butterfly optimization: A comprehensive review. *Expert Systems with Applications*, *168*, 114418.

Feo, T. A., & Resende, M. G. (1995). Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, *6*, 109–133.

Festa, P., & Resende, M. G. (2016). GRASP. In R. Martí, P. Panos, & M. G. Resende (Eds.), *Handbook of Heuristics* (pp. 465–488). Springer International Publishing volume 1-2.

Glover, F. (1989). Tabu Search—Part I. *ORSA Journal on Computing*, *1*, 190–206.

Glover, F., Campos, V., & Martí, R. (2021). Tabu search tutorial. a graph drawing application. *TOP*, (pp. 1–32).

Glover, F., Kuo, C.-C., & Dhir, K. S. (1998). Heuristic algorithms for the maximum diversity problem. *Journal of Information and Optimization Sciences*, *19*, 109–132.

Glover, F., & Laguna, M. (1998). Tabu Search. In *Handbook of Combinatorial Optimization* (pp. 2093–2229). Springer US.

Hansen, P., & Mladenović, N. (2005). Variable neighborhood search. In *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques* (pp. 211–238). Springer US.

Hansen, P., & Mladenović, N. (2009). Variable neighborhood search methods. In C. A. Floudas, & P. M. Pardalos (Eds.), *Encyclopedia of Optimization* (pp. 3975–3989). Boston, MA: Springer US.

Hansen, P., Mladenović, N., Todosijević, R., & Hanafi, S. (2016). Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, *5*, 423–454.

He, X., Chen, A., Chaovalitwongse, W. A., & Liu, H. X. (2012). An improved linearization technique for a class of quadratic 0-1 programming problems. *Optimization Letters*, *6*, 31–41.

Hoos, H. H., & Stützle, T. (2005). 4 - empirical analysis of sls algorithms. In H. H. Hoos, & T. Stützle (Eds.), *Stochastic Local Search* The Morgan Kaufmann Series in Artificial Intelligence (pp. 149–202). San Francisco: Morgan Kaufmann.

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

Li, J., Lei, H., Alavi, A. H., & Wang, G.-G. (2020). Elephant herding optimization: variants, hybrids, and applications. *Mathematics*, *8*, 1415.

Li, W., Wang, G.-G., & Gandomi, A. H. (2021). A survey of learning-based intelligent optimization algorithms. *Archives of Computational Methods in Engineering*, (pp. 1–19).

Martí, R., & Duarte, A. (2010). MDPLIB - maximum diversity problem library. `https://www.uv.es/rmarti/paper/mdp.html`.

Martí, R., Duarte, A., Martínez-Gavara, A., & Sánchez-Oro, J. (2021a). The MDPLIB 2.0 library of benchmark instances for diversity problems. `https://www.uv.es/rmarti/paper/mdp.html`.

Martí, R., Martínez-Gavara, A., Pérez-Peló, S., & Sánchez-Oro, J. (2021b). A review on discrete diversity and dispersion maximization from an OR perspective. *European Journal of Operational Research (to appear)*, .

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

Martínez-Gavara, A., Corberán, T., & Martí, R. (2021). GRASP and tabu search for the generalized dispersion problem. *Expert Systems with Applications*, (p. 114703).

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*, 1097 – 1100.

Parreño, F., Álvarez-Valdés, R., & Martí, R. (2021). Measuring diversity. A review and an empirical analysis. *European Journal of Operational Research*, *289*, 515–532.

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

Sánchez-Oro, J., López-Sánchez, A. D., & Colmenar, J. M. (2017). A general variable neighborhood search for solving the multi-objective open vehicle routing problem. *Journal of Heuristics*, *26*, 423–452.

Sánchez-Oro, J., Pantrigo, J. J., & Duarte, A. (2014). Combining intensification and diversification strategies in VNS. an application to the vertex separation problem. *Computers & Operations Research*, *52*, 209–219.

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

Sayyady, F., & Fathi, Y. (2016). An integer programming approach for solving the p-dispersion problem. *European Journal of Operational Research*, *253*, 216–225.

Snyder, L. V., & Daskin, M. S. (2005). Reliability models for facility location: the expected failure cost case. *Transportation Science*, *39*, 400–416.

Sörensen, K., & Glover, F. (2013). Metaheuristics. *Encyclopedia of operations research and management science*, *62*, 960–970.

Wang, G.-G., Deb, S., & Coelho, L. D. S. (2018). Earthworm optimisation algorithm: a bio-inspired metaheuristic algorithm for global optimisation problems. *International journal of bio-inspired computation*, *12*, 1–22.

Wang, G.-G., Gandomi, A. H., Alavi, A. H., & Gong, D. (2019). A comprehensive review of krill herd algorithm: variants, hybrids and applications. *Artificial Intelligence Review*, *51*, 119–148.

Wang, J., Zhou, Y., Yin, J., & Zhang, Y. (2009). Competitive Hopfield Network Combined With Estimation of Distribution for Maximum Diversity Problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *39*, 1048–1066.