Heuristics for the Capacitated Dispersion Problem

JUANJO PEIRÓ
Departament d'Estadística i Investigació Operativa,
Universitat de València, Spain
Juanjo.Peiro@uv.es

IRIS JIMÉNEZ
Departamento de Matemática,
Facultad de Ciencias Naturales, Exactas y Tecnología
Universidad de Panamá, Panama
irismarina71@gmail.com

JOSÉ LAGUARDIA
Departamento de Ciencias Exactas,
Facultad de Ciencias y Tecnología
Universidad Tecnológica de Panamá, Panama
Jose.Laguardia@utp.ac.pa

RAFAEL MARTÍ
Departament d'Estadística i Investigació Operativa,
Universitat de València, Spain
Rafael.Marti@uv.es

ABSTRACT

In this paper, we investigate the adaptation of the Greedy Randomized Adaptive Search Procedure (GRASP) and Variable Neighborhood Descent (VND) methodologies to the Capacitated Dispersion Problem (CDP). Dispersion and diversity problems arise in the placement of undesirable facilities, workforce management and social media, among others. Maximizing diversity deals with selecting a subset of elements from a given set in such a way that the distance among the selected elements is maximized. We target here a realistic variant with capacity constraints for which a heuristic with a performance guarantee was previously introduced. In particular, we propose a hybridization of GRASP and VND implementing within the Strategic Oscillation framework. To evaluate the performance of our heuristic, we perform extensive experimentation to first set key search parameters, and then compare the final method with the previous heuristic. Additionally, we propose a mathematical model to obtain optimal solutions for small size instances, and compare our solutions with the well-known Local-Solver software.

KeyWords: Diversity maximization, dispersion, Metaheuristics, Combinatorial Optimization.
Version: March, 2020

1. INTRODUCTION

The Capacitated Dispersion Problem (CDP) is an NP-Hard problem that belongs to the family of dispersion or diversity problems (Sandoya et al. 2018). When dealing with dispersion, the operations research literature has focused on maximizing diversity while neglecting, for the most part, the introduction of constraints. Several models have been proposed to deal with dispersion problems (Prokopyev et al. 2009). All of them require a diversity measure, typically based on a distance function. The most studied model is known as the maximum diversity problem (MDP, Silva et al. 2004, Silva et al. 2007, Duarte and Martí 2007), in which the sum of the distances between the selected elements is maximized. A very popular alternative is the max-min diversity problem (MMDP, Resende et al. 2010), in which the minimum distance between the selected elements is maximized. There are many variants of diversity models, see for example Martínez-Gavara et al. (2017a, 2017b).

The literature on diversity and dispersion problems is vast, starting with Glover et al. (1998). We refer the reader to a recent book chapter by Sandoya et al. (2018), which summarizes the previous heuristics and formulations for this problem. Assuming that the term dispersion may have different interpretations, not always properly defined, different mathematical models regarding dispersion may result in different types of solutions. Martí and Sandoya (2013) reviewed five diversity models and pointed out that the MMDP reflects in a better way the idea of dispersion. To illustrate this point, Figure 1 shows the optimal solutions of the MDP (left) and MMDP (right) respectively of an instance with 30 elements from which we select 10. It is clear that both solutions have a very different structure, being the 10 points in the MMDP solution (the one in the right part of Figure 1) better distributed considering that in the Max-Sum solution (shown in the left part of the figure) we can find two points that are very close from each other.

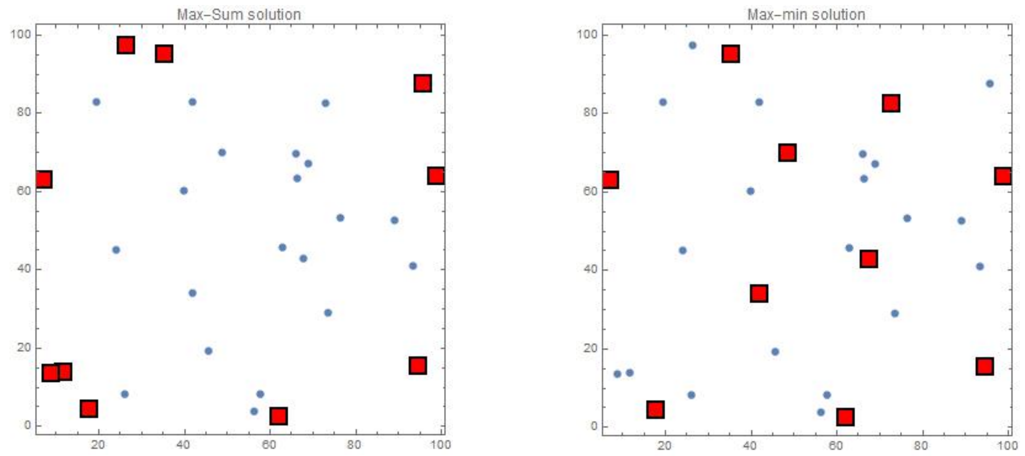

Figure 1. Example to illustrate MDP and MMDP solutions.

In this paper, we consider the dispersion variant known as CDP that was introduced in Rosenkrantz et al. (2000), in which a capacity constraint is added to the MMDP model. The motivation for this constrained model comes from its practical applications in facility location. For example, the location of undesirable or hazardous facilities, such as waste sites or nuclear plants, requires their dispersion while satisfying a certain total demand. Another example can be found in the context of retail franchises, where shops should not be located close to each other. These facilities/shops have a capacity to provide a service in systems that require an overall demand. As stated by the authors in Rosenkrantz et al. (2000), "these practical aspects add a new dimension to the conventional dispersion problem". Classical models, such as the MDP or MMDP, indirectly address the problem requirements by considering a pre-fixed number of facilities (i.e., the number of points to be selected is an input to the problem). However, more realistic approaches should be considered, as in the case of CDP, in which the capacity of each facility depends on its location.

Rosenkrantz et al. (2000) proposed a heuristic with performance guarantee to solve this NP-hard problem. The authors proved that on instances with inter-objects distances satisfying the triangle inequality, their heuristic has a performance guarantee of 2. Although no empirical results or experiments are reported, the theoretical study also concludes that their heuristic running time is $O(n^2 \log n)$. In this paper we approach this problem from a practical perspective, aiming at complementing their analysis. In particular, we propose heuristic algorithms based on state-of-the-art metaheuristic methodologies such as GRASP and VND in which we do not have a performance guarantee, but experimental results show that they statistically perform very well when solving a large set of instances.

Given a graph $G = (V, E)$, where V is the set of n nodes and E is the set of edges, let d_{ij} be the inter-elements distance between any two elements i and $j \in V$. Let c_i be the capacity of node $i \in V$, and B the total capacity required. The CDP can be easily formulated with the set M of selected elements, and with binary variables x_i that take value 1 if element i is selected, and 0 otherwise, as:

$$\text{(F1) Maximize} \quad f(M) = \min_{i,j \in M} d_{ij} \tag{1}$$

$$\text{subject to:} \quad \sum_{i=1}^{n} c_i x_i \geq B \tag{2}$$

$$x_i \in \{0,1\} \qquad i = 1, \dots, n \tag{3}$$

Where $M = \{i \in V : x_i = 1\}$ is the set of selected elements. Symmetrically, let $U = V \backslash M$ be the set of unselected elements. Then, the Capacitated Dispersion Problem (*CDP*) consists of selecting a set $M \subseteq V$ of elements such that the smallest distance between each pair of them is maximized, while the sum of their capacities is at least B. The objective is to maximize its minimum inter-distance value, $f(M)$, by a judicious selection of M.

The contributions of our work can be summarized as follows:

- Implementation of the previous heuristic for CDP (Rosenkrantz et al. 2000)
- Development and implementation of GRASP and VND heuristics for the CDP
- Creation of a hybrid approach combining GRASP and VND within Strategic Oscillation.
- Development of an efficient mathematical formulation
- Implementations of CDP models for Cplex and LocalSolver
- Comparison of new and existing methods on general instances for the CDP

As mentioned above, one of the objectives of this paper is to investigate a Strategic Oscillation (SO) proposal. It basically alternates between constructive and destructive phases as a basis for creating a competitive method. SO was proposed in the context of the tabu search methodology

(Glover and Laguna, 1997), and we propose to integrate it with GRASP and VND to create a hybrid efficient method to obtain high-quality solutions for the Capacitated Dispersion Problem.

2. MATHEMATICAL FORMULATION

The Capacitated Dispersion Problem can be easily formulated, as shown in the Introduction, in mathematical terms as a quadratic binary problem. Note however that the objective function (1) is nonlinear, which makes it difficult to solve. As a matter of fact, it can be modeled as the product of two variables and the computation of a minimum value, and each of this two characteristics makes the model nonlinear. To linearize it, we employ a standard artifact. For each $(i,j) \in E$, we introduce binary variables y_{ij} that take the value 1 if and only if x_i and x_j simultaneously take the value 1. In other words, y_{ij} takes the value of the product of both variables (but we avoid the use of this product to obtain a linear formulation). We accomplish it with constraints (5) - (7) below. With these three families of constraints we overcome the problem of multiplying variables.

To deal with the computation of a minimum value, we introduce constraints (8) where an upper bound D on the distances values permits to model it. In particular, when $y_{ij} = 1$ the expression $m \leq d_{ij}y_{ij} + D(1 - y_{ij})$ results in $m \leq d_{ij}$, and considering that we compute it for each pair of selected elements $(i, j \in M)$, m takes the minimum value of their distances. On the other hand, when $y_{ij} = 0$, expression (8) results in $m \leq D(1 - y_{ij})$, which can be simplified as $m \leq D$. Since D is an upper bound on the distances values (i.e., it is an arbitrary value larger than all the distances in G), then these constraints are not active when $y_{ij} = 0$. The complete formulation follows:

$$\text{(F2) Max} \quad m \qquad (4)$$

subject to:

$$\textstyle\sum_{i=1}^{n} c_i x_i \geq B \qquad (2)$$

$$y_{ij} \leq x_i \qquad 1 \leq i < j \leq n \qquad (5)$$

$$y_{ij} \leq x_j \qquad 1 \leq i < j \leq n \qquad (6)$$

$$x_i + x_j \leq y_{ij} + 1 \qquad 1 \leq i < j \leq n \qquad (7)$$

$$m \leq d_{ij}y_{ij} + D(1 - y_{ij}) \qquad 1 \leq i < j \leq n \qquad (8)$$

$$y_{ij} \in \{0,1\} \qquad 1 \leq i < j \leq n \qquad (9)$$

$$x_i \in \{0,1\} \qquad 1 \leq i \leq n \qquad (3)$$

Formulation F2 is equivalent to F1, showed in the previous section, but it only contains linear expressions. In our computational experiments in Section 5, we implement this formulation in Cplex to obtain the optimal solution for small and medium size instances.

To illustrate how are the optimal solutions obtained with this formulation, we consider the small example with 50 points shown in Figure 2 with capacity values ranging from 1 to 1000 for each point.

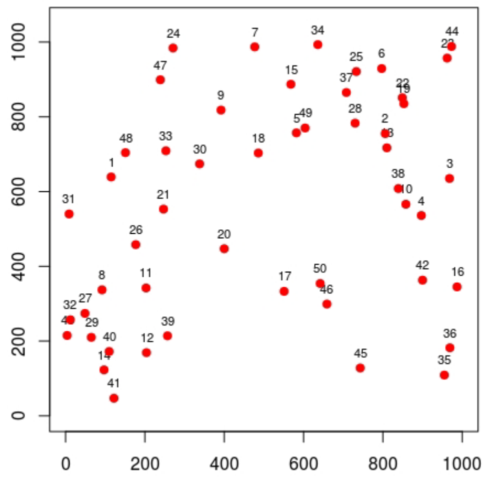

Figure 2. Example to illustrate CDP solutions

We solve the CDP problem of the example in Figure 2 with Cplex, running with the integer linear formulation F2. In particular, we consider a capacity limit $B = 6275$. In less than 2 seconds, we obtain the optimal solution $M = \{1, 4, 23, 24, 27, 36, 46, 49\}$ with an objective function value of $m = 331.29$ and a capacity value of 6411. Figure 3 shows the selected points in M.

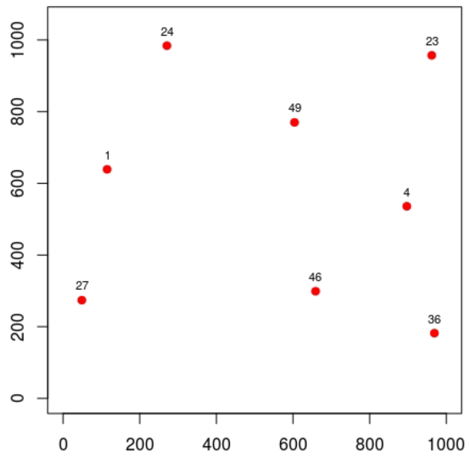

Figure 3. Cplex optimal CDP solution.

Unfortunately, Cplex cannot solve large instances with this formulation. As it is well-known, the branch and bound code implemented in Cplex to deal with integer variables explores a huge number of solutions when the problem size is large, leading to impractical running times. We therefore resort to heuristic methods to target large instances.

In the next section, we describe the previous heuristic method, and apply it to solve this problem to illustrate its performance in a small example with optimal solution known.

3. THE PREVIOUS HEURISTIC METHOD

As mentioned in the Introduction, Rosenkrantz et al. (2000) proposed the only previous heuristic known for this problem. This heuristic, called T1, has a performance guarantee of 2. This means that the solution of the method is within a distance of the optimal solution. In particular, for any instance, the optimal value divided by the value of the solution obtained with the heuristic is lower than 2.

Figure 4 shows a pseudo-code of this previous method. It basically calls the routine *Greedy_Try* with two parameters, α and B, which tries to select a set of nodes, called sites, in a greedy fashion to satisfy the distance constraint α and capacity constrain B. In this code, $CAP\ (V')$ denotes the sum of the capacities of the nodes in V'.

```
Heuristic T1

1. Sort the sites in non-increasing capacity order, and create a list Site_List.
2. Sort the inter-site distances in non-increasing order. Eliminate
   duplicates distances. Let the resulting sorted list of distances be
   sorted in the array $D$ such that $D[1] > D[2] > \dots > D[t]$.
3. Carry out a binary search over the array $D$ to find the index $i$ such
   that for $\alpha = D[i]$, the call Greedy_Try(α, B) returns "success" and
   for $\alpha' = D[\,i - 1\,]$, the call Greedy_Try(α', B) returns "failure".
4. Output the intersite distance α found in Step 3 and stop.

Procedure Greedy_Try(α, B)

1. Let L := Site_List and $V' = \emptyset$.
2. While L is not empty do
      a. Add the first node $v$ from $L$ to $V'$.
      b. Remove from $L$ all sites (including v) whose intersite distance to v is
         strictly less than α.
3. If $CAP\ (V') \geq B$ then return "success" else return "failure".
```

Figure 4. Previous heuristic T1.

We implement the algorithm T1 shown above to perform an empirical comparison with our heuristics proposed in the next section. The results of our experimentation are shown in Section 5. We illustrate now how the method works, as the authors did in Rosenkrantz et al. (2000) with some examples.

We first consider the example in Figure 5 based on Euclidean distances. It has four nodes, namely 1, 2, 3, and 4, with capacity 2 in the corner of a square of two units, and node 5 in the center with capacity 1. We apply algorithm T1 to solve the CDP with a capacity limit $B = 5$. In step 1 of T1 we create Site_list = (1,2,3,4,5), and in step 2, we obtain D=(2.82, 2 , 1.41). Then, we call Greedy_Try(α =2.82, B = 5), and perform the following steps:

1. v = 1 ; L = (4) ; V' = (1)
2. v = 4 ; L = Ø; V' = (1, 4)

3. Failure

Then, we call Greedy_Try(α =2, B = 5) again an perform these steps:

1. v = 1 ; L = (2,3,4) ; V' = (1)
2. v = 2 ; L = (3,4); V' = (1, 2)
3. v = 3 ; L = (4); V' = (1, 2,3)
4. Success

Obtaining the solution V' = (1,2,3) with α =2

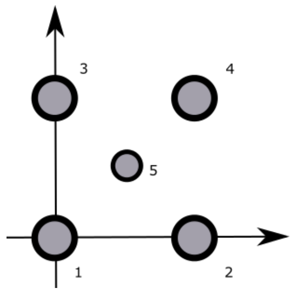

Figure 5. Small instance

We now solve the CDP problem of the example in Figure 2 with T1. As we did with Cplex in the previous section, we consider a capacity limit $B = 6275$. We obtain the solution $M = \{1, 4, 8, 15, 23, 24, 36, 46\}$ with an objective function value of $m = 302.9$ and a capacity value of 6514. Figure 6 shows the selected points in M.

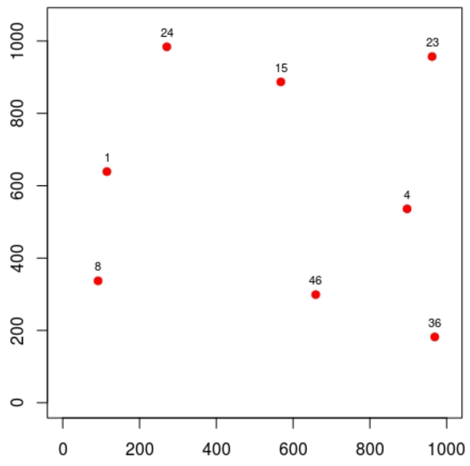

Figure 6. Heuristic solution of example in Figure 2.

If we compare the optimal solution obtained with Cplex in the previous section, with a value of $m = 331.29$, with the heuristic solution above, with a value of $m = 302.9$, we can conclude that there is room for improvement in the design of a heuristic method. This is basically the motivation of our work: the design of a heuristic algorithm to obtain high-quality solution in short computational times.

4. NEW HEURISTIC METHODS

In this section we propose three new methods to obtain good solutions for the CDP. The first one is based on the GRASP methodology. It is complemented with a VND method as a local search optimizer, described in the second subsection. Finally, both methods are integrated on a strategic oscillation scheme for improved outcomes. The following subsections describe in detail the three methods.

4.1 GRASP - Greedy Randomized Adaptive Search Procedure

The GRASP methodology was proposed by Feo and Resende (1995). Each GRASP iteration consists of constructing a trial solution and then applying an improvement method to find a local optimum. The construction phase is iterative, greedy, randomized and adaptive. It is iterative because the initial solution is built considering one element at a time. It is greedy because the addition of each element is guided by a greedy function. It is randomized because a random selection takes place and the information provided by the greedy function is used in combination with this random element.

Given the set V with n vertices or nodes, the construction procedure C1 performs consecutive steps to produce a solution. The set M, initially empty, represents the partial solution under construction. At each step, C1 selects a candidate element $i \in V \setminus M$ with a good evaluation, $eval(i)$. A straightforward evaluation for candidate elements in this problem is the distance to the elements already in the partial solution. Specifically, we can compute $eval(i)$ as the sum of the distances between element i and the selected elements. In mathematical terms:

$$eval(i) = \sum_{j \in M} d_{ij}$$

Then, C1 constructs the restricted candidate list, RCL, with all the candidate (unselected elements) with an evaluation within a fraction of the maximum evaluation. In mathematical terms:

$$RCL = \{i \in V \setminus M : \; eval(i) \geq eval_{min} + \alpha\,(eval_{max} - eval_{min})\} \qquad (10)$$

where $\alpha \in [0,1]$ is a search parameter that will be empirically adjusted. Then, the method randomly selects an element in RCL (see equation (10)), and adds it to the partial solution M. C1 performs steps as long as the capacity constraint (2) is not met. In other words, the method stops when the sum of the capacities of the elements in M is larger than or equal to B.

One could argue that C1 is blind in its selection process with respect to the capacity values of the nodes. To overcome this limitation, we propose C2 with a more elaborated evaluation function. In particular, it first computes for any element $i \in V \setminus M$, $d_i = \sum_{j \in M} d_{ij}$ and $d_{max} = \max_i d_i$ to adjust the contribution of the distance value to the range $[0,1]$. Similarly, it computes $c_{max} = \max_i c_i$ and then:

$$eval(i) = \beta\,\frac{d_i}{d_{max}} + (1-\beta)\frac{c_i}{c_{max}} \qquad (11)$$

and the relative weight of these two factors, distance and capacity, is adjusted with the $\beta \in [0,1]$ parameter. C2 performs steps in the same manner than C1, with the only difference of the evaluation function (11).

Glover et al. (1998) is probably the first study on diversity problems from a heuristic optimization perspective. The authors anticipate that since different versions of this problem may include additional constraints, as it is our case here, the objective is to design heuristics whose basic moves for transitioning from one solution to another are both simple and flexible, allowing these moves to be adapted to multiple settings. Especially attractive moves in this context are constructive and destructive processes that drive the search to approach and cross-feasibility boundaries from different directions. Such moves are also highly natural in the maximum diversity problem, where the goal is to determine an optimal composition for a set of selected elements. In line with these observations, we propose two "destructive" methods, called D1 and D2.

D1 is the counterpart of C1, in which in each iteration, instead of adding an element to the partial solution, we remove one element from this partial solution. In destructive methods we consider that initially all the elements are selected and we remove elements, one-by-one as long as the capacity constraint is satisfied. In particular, in each iteration we remove, or deselect, the element with a relative bad evaluation, as indicated by the following RCL:

$$RCL = \{i \in M : \ eval(i) \leq eval_{min} + \gamma \ (eval_{max} - eval_{min})\}, \qquad (12)$$

where $\gamma \in [0,1]$. Similarly, we propose D2 as the counterpart of C2. It employs the same evaluation function than C2, although in D2 it is meant to identify the elements with a bad evaluation. Then, the method removes from the solution one of them randomly selected. The selection is performed from a restricted candidate list (RCL) as it is customary in GRASP.

4.2 VND – Variable Neighborhood Descent

The Variable Neighborhood Search (VNS) methodology is based on a simple and effective idea: a systematic change of neighborhood within a local search algorithm (Mjirda et al. 2017; Hansen et al., 2017). Variable Neighborhood Descent (VND) is a variant of VNS that explores neighborhoods in a deterministic way. In particular, VND explores small neighborhoods until a local optimum is encountered. At that point, the search process switches to a different (typically larger) neighborhood that might allow further progress towards the global optimum. In this section we adapt the VND to the Capacitated Dispersion Problem. We follow the description given in Duarte et al. (2018).

We define $N_k(M)$ for $k = 1, 2, \ldots, k_{max}$ as the set of solutions that are obtained when we exchange k elements in solution M with k elements in $V \setminus M$. Exchanges in this context consist of replacing selected elements with unselected ones. VND is based on the fact that a local optimum is defined with respect to a neighborhood relation, such that if a candidate solution M is locally optimal in a neighborhood $N_i(M)$, it is not necessarily a local optimum for another neighborhood $N_j(M)$. Note that in our problem, we need to check that the solution is feasible after the exchange (i.e., that it verifies constraint (2) in Formulation F1). Specifically, when we replace k elements in the solution, the sum of the capacities of the selected elements after the replacement has to be larger than or equal to B. To simplify the description, we can just say that we only consider feasible exchanges.

Given a set V with n elements, and a feasible solution M with some of these elements selected, we compute for $i \in M$,

$$dm_i = \min_{j \in M} d_{ij}.$$

Note that the objective function value of this solution, $f(M)$, is computed as the minimum of the dm_i-values. It is clear that to improve a solution we need to remove (and thus replace) the elements i in the solution for which dm_i = $f(M)$. Our method initially scans, at each iteration, the list of elements in the solution ($i \in M$) with minimum dm_i value. In particular, it scans the list of elements in lexicographical order, and for each element i^* with a minimum dm_i value, it considers the list of unselected elements ($j \in V \setminus M$) in search for the first improving exchange in $N_1(M)$. Considering that the set $V \setminus M$ is relatively large (as compared with the set M), we implement a strategy to scan it in an efficient way to find a good exchange for i^*. In particular, we compute $de_j(i^*)$ for $j \in V \setminus M$, where:

$$de_j(i^*) = \min_{i \in M \setminus \{i^*\}} d_{ij}$$

and scan these vertices in the order induced by the de −values, where the one with the largest value comes first. It must be noted that the de −values are an indicator of the potential quality of an element to become part of the solution when we remove i^*. Since the objective function is to maximize the inter-distance values, the larger the de the better the element. This is why we explore first the elements with larger de −values in our search for a good exchange. It is indeed a steepest descent strategy (Glover and Laguna, 1997).

The method performs the first improving move and updates dm_i for all elements in M. In this first phase, the algorithm repeats iterations as long as improving exchanges can be performed and when no further improvement is possible, it resorts to consider exchanges of two elements, implementing in this way a VND. We limit our method to two neighborhoods, $N_1(M)$ and $N_2(M)$, to avoid the larger running times associated with large neighborhoods.

An important point in Max-Min problems is the definition of improving moves. Previous papers on this type of problems consider an extended definition, which includes not only when the move increases the value of $f(M)$, but also when a certain indicator improves (see for example Resende et al. 2010). In our VND algorithm we apply this extended criterion in which when the number of elements $i \in M$ for which dm_i=$f(M)$ is reduced, we consider that the solution improves.

As in typical VND implementations, our method explores the neighborhood structures in a sequential way (i.e., from 1 to 2). In particular, k is initially set to 1; then, in each step, an improving neighbor M' of M, is determined in $N_k(M)$: if M' is better than M according to the extended definition above, then M is replaced with M'; otherwise, k is incremented by one unit (i.e., $k = k + 1$). In other words, the algorithm performs a local search to improve the solution in $N_1(M)$ and only resorts to $N_2(M)$ when the search is trapped in a local optimum found in $N_1(M)$. Following the strategy called Basic VND (Duarte et al. 2018), when an improving move is performed, and the incumbent solution is updated, the method returns to the first neighborhood (i.e., $k = 1$). Finally, when both neighborhoods have been explored and no improvement is found ($k = 2$), the VND method stops.

4.3 SO – Strategic Oscillation

Strategic oscillation (Glover and Laguna, 1997) operates by orienting moves in relation to a critical level, which in our problem is defined as the capacity level B. Such a critical level or oscillation boundary often represents a point where the method would normally stop. Instead of stopping when this boundary is reached, the rules for selecting moves (in constructive method C1 and destructive method D1) are modified, to permit the region defined by the critical level to be crossed. The approach then proceeds for a specified depth beyond the oscillation boundary, and turns around. The oscillation boundary again is approached and crossed, this time from the opposite direction, and the method proceeds to a new turning point. The process of repeatedly approaching and crossing the critical level from different directions creates an oscillatory behavior, which gives the method its name.

Constructive method C1 described in Section 4.1 adds elements, one-by-one, to the current solution under construction, until the sum of the capacities of the selected elements is larger than or equal to B. At this point it has a feasible solution, say M_1 and the method stops. What we propose now is to “keep going” a few more steps. In particular, we consider the addition of extra elements to the solution M_1, with the same method C1, obtaining solution M_2. Then, we will apply a destructive method, such as D1, to remove from M_2 some elements. In this way, we may obtain a new feasible solution, M_3, that eventually could be better than the two previous ones.

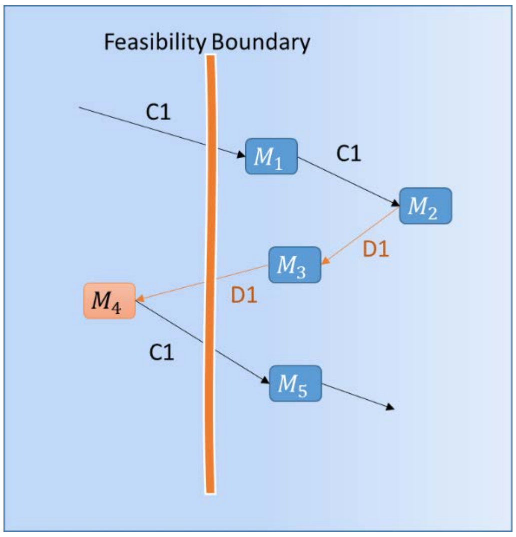

Figure 7. Strategic Oscillation pattern

In line with the “keep-going” strategy described above when adding elements, we propose to remove a few additional elements from the feasible solution. In particular, we apply D1 to remove some extra elements from M_3, thus obtaining a partial, unfeasible, solution M_4. If we repeat this scheme, adding now elements to M_4 with C1, obtaining a new feasible solution M_5, we have an oscillation pattern, crossing the feasibility boundary of the solution space, as illustrated in Figure 7.

In our oscillation strategy we add and remove vertices according to the capacity limit. Specifically, when applying C1 to add extra vertices to the solution, which corresponds to the steps from solution M_1 to M_2 in Figure 7, we multiply the sum of the capacities of the elements in the solution by a factor $\lambda \in [0,1]$ and stop when this product is larger than B. In mathematical terms, we apply C1 while $\lambda \sum_{i \in M} c_i \geq B$.

We propose a selective application of the VND method to improve some of the solutions, instead of applying it to all the solutions encountered in the process. In particular, if we consider the “path of solutions” when we apply C1 from the “unfeasible region” to the “feasible region”, we can identify the first feasible solution. That would correspond to solutions M_1 and M_5 in Figure 7. We apply VND to these solutions, and we do not apply it to the rest of the solutions in the path obtained with C1 (i.e., we skip M_2 and M_4 to the application of VND). Symmetrically, when applying the destructive method D1 from the “feasible region” to the “unfeasible region” we identify the last solution visited before abandoning the feasible region. That would be M_3 in Figure 7. We apply VND to these last solutions. In this way we save computational time and avoid to obtain the same local optima when applying VND from different initial solutions.

5. COMPUTATIONAL EXPERIMENTS

This section describes the computational experiments that we performed to first set the parameter values of our methods, and then compare them to the previous heuristic method, T1, and the Cplex optimal solutions, when solving the capacitated dispersion problem. Additionally, we compute the LocalSolver solutions for our test instances. In both solvers, Cplex and LocalSolver, we implement the model described in Section 2.

LocalSolver is a commercial optimization system that provides extremely robust performance across multiple classes of optimization problems. The search starts from a solution generated by a basic greedy randomized procedure. The search is then performed in the feasible region and moves are performed to transform one solution to another. As its name indicates, LocalSolver attempts to find local optima by way of standard ascent (for maximization problems) techniques. The embedded heuristics allow the process to select non-improving moves in order to escape local optimality. These heuristics include probabilistic models such as those typical to the simulated annealing methodology. A large catalog of moves is available during the search and the selection of the moves to try is dynamically adjusted. LocalSolver is free for academic uses and can be downloaded from http://www.localsolver.com/.

5.1 Problem Instances

For the experimentation, we use the public-domain MDPLIB (Martí et al., 2019) available at http://grafo.etsii.urjc.es/optsicom, which contains several data sets previously employed in different studies on diversity problems (Sandoya et al., 2018). We reviewed these instances and adapted them to the capacitated version of the diversity problem. In particular, for each original instance we randomly generate the capacity value of each node in the range $[1, 1000]$. Then, we compute the sum of all capacities and set B as this sum multiplied by a factor of 0.2 and 0.3 respectively, thus creating two instances for each original one. Our benchmark for the Capacitated Diversity Problem thus consists of the following 100 instances.

GKD: This data set, originally proposed by Glover et al. (1998), contains matrices for which the values were calculated as the Euclidean distances from randomly generated points with coordinates in the 0 to 10 range. This set contains three subset of instances:

GKD-a: Glover et al. (1998) introduced the small instances in this set with values of $n \leq 30$. We do not consider these instances because they are too small.

GKD-b: Martí et al. (2010) generated these medium size instances with values of $25 \leq n \leq 150$. We consider 10 instances of size 50, and 10 of size 150 in this set, and generate two instances for each of them as described above.

GKD-c: Duarte and Martí (2007) generated these large instances with $n = 500$. We consider 10 instances in this set, and generate two of them with different capacity values as in the other sets.

SOM: This data set consists of 70 matrices with random numbers between 0 and 9 generated from an integer uniform distribution. Martí et al. (2010) created these instances to solve the maximum diversity problem, in which the objective function is the sum of the distances. Since we target here the version in which the objective is computed with the minimum distance, we found that most of these instances cannot be used, since many points are at a distance of 0, causing the objective function to be 0. We selected 10 instances in the SOM-a subset with $n = 50$ that can be used for our problem. As in the previous sets, we generated two instances for the capacity version with the factors 0.2 and 0.3.

MDG: This data set was generated by Duarte, and Martí (2007), and used in Gallego et al. (2009) and Palubeckis (2007). It consists of 100 matrices with real numbers randomly selected from a uniform distribution.

MDG-a: This set contains instances with real numbers in the range 0, 10. We do not consider them because they are not adequate for Max-Min problems since some points are at a distance of 0.

MDG-b: This set contains instances with real numbers in the range 0, 1000. We consider 10 instances with $n = 500$, for which we create two of them with the capacity factor set as 0.2 and 0.3 as described above.

Table 1 summarizes the 100 instances in our benchmark.

Name	n	Capacity parameter	Number of instances
GKD-b2	50, 150	0.2	20
GKD-b3	50, 150	0.3	20
GKD-c2	500	0.2	10
GKD-c3	500	0.3	10
SOM-a2	50	0.2	10
SOM-a3	50	0.3	10
MDG-b2	500	0.2	10
MDG-b3	500	0.3	10

Table 1. Sets of instances.

5.2 Algorithm Configuration and Fine-Tuning

The goal of our preliminary experimentation is to find effective configurations for our methods and to fine tune their algorithmic parameters. Specifically, we proposed in the previous sections the following heuristics, which require finding appropriate values for their parameters (in brackets):

- C1(α)
- C2(α, β)
- D1(γ)
- D2(β, γ)
- VND
- SO(λ)

We find the values of these parameters with a training set with 24 representative instances from the set of 100 described above (see Table 1). For each experiment, we report the following performance measures: Average number of minimum inter-distance value ($f(M)$), computing time in seconds (Time), average relative percentage deviation with respect to the best solution found in the experiment (Dev), and number of best solutions found in the experiment (Best). Note that both Dev and Best refer to the solutions found within the experiment and not the best solutions known for these problems. The relative deviation is computed for each instance as the difference of the best value obtained in this experiment minus the value obtained with each particular algorithm, divided by the best value. We multiply the result by 100 to represent it as a percentage.

In our first preliminary experiment, we test the constructive method C1 described in Section 4.1. The performance of C1 depends on the parameter α, which balances greediness and randomness. We tested four values of α on each procedure (0.2, 0.4, 0.6, and 0.8). Table 2 shows the results of this experiment.

Procedure	$f(M)$	Dev (%)	Best	Time (s)
C1(0.2)	32	5.8	4	<1
C1(0.4)	32	3.4	6	<1
C1(0.6)	33	2.1	9	<1
C1(0.8)	33	6.3	9	<1

Table 2. Comparison of different α values in constructive method C1.

The minimum deviation value is obtained with $\alpha = 0.6$, which is also able to obtain the largest number of best solutions (9). We therefore select this value to set α in C1.

In our second experiment we explore the different values of the two parameters in constructive method C2. As in C1, parameter α manages the restricted candidate list (RCL) composition. In line with the findings in our previous experiment, we test here relatively large values for this parameter, favoring the greediness in the RCL composition. In particular, we consider two values: $0.6, 0.8$. On the other hand, parameter β is the relative weight between distance and capacity in the selection process. It takes values in [0,1] where the closer the value to 1, the more important the distance is with respect to the capacity. We also test the values $0.6, 0.8$ since we consider that the distance plays a more important role in the problem. Table 3 shows the results of this experiment.

Procedure	$f(M)$	Dev (%)	Best	Time (s)
C2(0.6, 0.6)	58.5	6.8	5	<1
C2(0.6, 0.8)	54.0	17.2	3	<1
C2(0.8, 0.6)	57.7	7.7	6	<1
C2(0.8, 0.8)	54.8	17.5	4	<1

Table 3. Comparison of different α and β values in constructive method C2.

The minimum deviation in Table 3 is obtained with $\alpha = \beta = 0.6$, which exhibits a 6.8 percent deviation. We select this value for these two parameters.

In our next two experiments we explore the performance of destructive methods D1 and D2 as we just did with C1 and C2. For the sake of brevity, we do not reproduce here the tables for these experiments (they are similar to those reported above). The conclusion is that the best performance is achieved in D1 when $\gamma = 0.4$ and in D2 with $\beta = 0.6, \gamma = 0.5$. We will use these values in the following experiments.

We now compare the four methods proposed to generate solutions when coupled with the VND improvement method. We consider these methods with their key-search parameters set as adjusted in our previous experiments. Table 4 shows the results of our sixth preliminary experiment to disclose the best generation method. As in the previous experiment we consider the following four statistics to compare them: $f(M)$, Dev., Best, and Time (in seconds).

Procedure	$f(M)$	Dev (%)	Best	Time (s)
C1(0.6)+VND	97.4	3.7	8	8.1
C2(0.6,0.6)+VND	99.4	2.8	7	11.4
D1(0.4)+VND	98.7	3.8	2	10.1
D2(0.6,0.5)+VND	97.7	4.3	2	8.7

Table 4. Comparison of different GRASP methods.

The best constructive method in Table 4 is C2(0.6,0.6) + VND with a 2.8 percentage deviation and 7 best solutions, while the best destructive method is D1(0.4)+VND with a 3.8 percentage deviation and 2 best solutions. We therefore use them in the complete Strategic Oscillation (SO) heuristic, in which constructive and improvement methods are embedded. As described in Section 4.3, SO operates by first applying a constructive method in which additional iterations are performed beyond the feasible boundary as indicated by parameter λ. Then, a destructive method is applied to "come back" to that boundary and cross it again. The alternation of both methods establish the oscillation pattern that gives the name to the methodology. For the sake of simplicity we denote to this hybrid method as SO(λ). Table 4 shows the results when comparing different values of λ. To simplify it, we only report the average percentage deviation (Dev) and the running time in seconds (Time). We also include in this table different values for the number of oscillations.

	Number of Oscillations									
	1		2		3		4		5	
λ	**Dev**	**Time**	**Dev**	**Time**	**Dev**	**Time**	**Dev**	**Time**	**Dev**	**Time**
0.6	5.4%	37	1.3%	69	0.2%	131	0.5%	139	0.7%	168
0.7	5.3%	35	2.1%	71	3.5%	106	2.5%	148	2.3%	141
0.8	5.2%	36	3.9%	55	4.6%	89	4.0%	131	3.9%	137
0.9	5.4%	43	4.5%	68	4.6%	72	4.0%	145	4.1%	167

Table 5. Strategic Oscillation method.

Table 5 shows the evolution of the deviation of the best solution find with the SO method. For example, if we pay attention to the first row, which corresponds to $\lambda = 0.6$, we can see that after the first oscillation, the method obtains a solution with 5.4% of deviation on 37 seconds (this would be solution M_4 in Figure 7). Then, in the second oscillation, the method is able to improve this solution, obtaining a new solution with 1.3% of deviation, achieved on 69 seconds. If we continue with the oscillation process, after 5 oscillations, we can see that the method obtains a solution with a 0.7% of deviation achieved on 168 seconds. Comparing the different rows in this table, we conclude that $\lambda = 0.6$ obtains the best results, and therefore set this parameter to this value for the following experiments. Note that deviations can eventually increase in a row of this table since they represent the best solution in each oscillation, not the best solution overall.

We perform a final experiment to test the robustness of our SO method. In particular, we run it 30 times on each instance and collect the best value of the 30 runs, then we compute the average, minimum (Min.), maximum (Max.), standard deviation (St. Dev.) and variation coefficient (V.C.) for each instance in the training set. Table 6 reports the average results of these five statistics over the 24 instances in the set. We consider two versions of SO, a short run of about 1 second of computer, and a long one, of about 15 seconds.

Procedure	**Average**	**Min.**	**Max.**	**St. Dev.**	**V.C.**
SO (1 sec. each run)	99.09	96.21	102.18	1.72	0.02
SO (15 sec. each run)	101.55	99.91	103.48	0.94	0.01

Table 6. Average statistics on 30 runs (replications).

Table 6 shows that our SO method is quite robust. Even in short runs, it obtains similar solutions when replicated. In particular, the average standard deviation over 30 runs on the training set is 1.72, which implies a variation coefficient of 0.02. As expected, if we consider longer runs (of about 15 seconds), the method is even more stable, obtaining very similar solutions in different replications (with a standard deviation of 0.94).

5.3 Competitive Testing

For the competitive testing, we compare the procedure that we developed, SO, with different methods to test its relative performance both in terms of running time and quality. In particular, we compare SO, set with C2(0.6,0.6) + VND as constructive process, D1(0.4)+VND as destructive process, and $\lambda = 0.6$ as oscillation value, with the following methods:

- The previous heuristic for this problem, T1, by Rosenkrantz et al. (2000)
- The Cplex solver, which provides the optimal solutions for small instances
- The LocalSolver general purpose heuristic

This experiment consists of executing the three procedures above on the entire set of 100 instances in our benchmark, and compare their results with those obtained with SO. We report the results in 3 tables, one for each method respectively. Table 7 shows the results obtained with SO and T1. As in the preliminary experiments, we report the objective function value, $f(M)$, the relative percentage deviation with respect to the best solution in this experiment, Dev, the number of instances in each set in which the method is able to match the best known solution, Best, and the running time in seconds, Time. Each row in Table 7 summarizes the results on the 10 instances in each set. The first five rows correspond to the instances generated with the capacity parameters 0.2, and the last five rows with 0.3.

		T1 previous heuristic				**SO**			
Set	n	$f(M)$	**Dev**	**Best**	**Time**	$f(M)$	**Dev**	**Best**	**Time**
GKD-b2	50	1057	6.4	0	0	1123	0.0	10	0
GKD-b2	150	1132	1.8	2	0	1152	0.0	9	0
GKD-c2	500	60	20.1	0	47	75	0.0	10	19
SOM-a2	50	36	12.0	6	0	41	0.0	10	0
MDG-b2	500	354	13.8	1	49	412	0.5	9	65
GKD-b3	50	925	4.9	2	0	970	0.0	9	0
GKD-b3	150	1046	0.2	7	0	1034	1.2	3	1
GKD-c3	500	51	21.9	0	47	65	0.0	10	41
SOM-a3	50	15	5.0	9	0	15	5.0	9	0
MDG-b3	500	86	21.6	2	49	111	1.9	8	45

Table 7. Comparison with previous heuristic.

Table 7 clearly shows the superiority of our method with respect to the previous heuristic T1 (see for example the Dev column in both methods). This is to be expected since T1 is a relatively simple approximation method, with a guarantee performance, and SO is a complex metaheuristic. Note however that SO not only outperforms T1 in solution quality, but it is also competitive in running time. On the other hand, T1 presents a very good performance considering its simplicity, with remarkable results on GKD-b3 and SOM-a3 sets of instances (7 and 9 best solutions out of 10 in each set respectively). Note that we set both methods, SO and T1, in a way that they run on average for a similar CPU time for a fair comparison. In this way, in many of the sets their times are equal or very similar (this is the case of GKD-b2, SOM-a2 or MDG-b3). However, there is one set, GKD-c2, with significantly different times (47 and 19 seconds respectively). We prefer to keep it this way because both methods are run with the same configuration across all instances.

To complement the analysis above we compare both methods with two well-known nonparametric tests for pairwise comparisons: the Wilcoxon test and the Sign test. The former one answers the question: Do the two samples (solutions obtained with T1 and SO in our case) represent two different populations? We obtain a z-value of -5.434 (with sum of pos. ranks of 23, and sum of neg. ranks of 923), and an associated p-value lower than 0.00001 in the Wilcoxon test, which indicates that the values compared do not come from the same method. On the other hand, the Sign test computes the number of instances on which an algorithm supersedes another one. The resulting z-value of 6.328 (with pos. sign count of 13, and neg. sign counts of 71), and an associated p-value lower than 0.00001 indicates again that there is a clear winner between both methods (SO) when we consider all the instances in the benchmark set.

		Cplex				**SO**			
Set	n	$f(M)$	**Dev**	**Best**	**Time**	$f(M)$	**Dev**	**Best**	**Time**
GKD-b2	50	112.3	0.0	10	2.8	112.3	0.0	10	0.1
GKD-b2	150	118.6	4.1	10	2926.0	118.4	4.3	6	37.5
SOM-a2	50	4.1	0.0	10	1.8	4.1	0.0	10	0.0
GKD-b3	50	97.8	0.0	10	4.3	97.8	0.0	10	1.6
GKD-b3	150	107.4	31.4	8	3600.0	107.2	31.6	5	56.3
SOM-a3	50	2.1	0.0	10	6.0	1.8	13.3	7	0.1

Table 8. Comparison with exact method.

In the following experiment, we compare SO with the solutions obtained with Cplex. We do not include in this experiment the large size instances ($n = 500$), since Cplex is not able to solve them in the time limit of 6,000 seconds considered. Table 8 reports the solutions of this experiment, where the deviation value, Dev, is computed with respect to the upper bound obtained with Cplex. On the other hand, the number of best solutions, Best, considers the lower bound obtained with Cplex, and the best solution obtained with SO. Note that when Cplex is able to terminate the branch and bound exploration, both values, lower and upper bound, are the same.

Results in Table 8 indicate that Cplex is able to solve the small instances, with $n = 50$, to optimality in moderate running times (lower than 10 seconds). However, when we move to the medium size instances, with $n = 150$, it only solves a fraction of them within the time limit of 3,600 seconds (note the deviations of 4.1 and 31.4 in the GKD-b2 and GKD-b3 sets respectively). As mentioned above, Cplex is not able to solve the large size instances $n = 500$ in any of the cases tested within the 3,600 seconds considered.

Considering now our SO method, we can see in Table 8 that in the small GKD instances (sets GKD-b2 and GKD-b3 with $n = 50$), it is able to match the 20 optimal solutions in a very small running time (less than 2 seconds). In the larger instances in these sets, with $n = 150$, our SO heuristic obtains good solutions but not optimal in all the cases. In particular, in GKD-b2 it obtains 6 out of 10, and in GKD-b3, 5 out of 10. Note however, that we only run it for a moderate running time (less than 1 minute).

Table 8 reveals that the instances in the SOM-a3 set constitute a challenge for heuristic methods. These are small instances with $n = 50$ in which Cplex is able to compute the optimal solutions. However, our method only obtains 7 out of the 10 optimal solutions. Considering that in Table 6, T1 exhibits a similar performance than SO in this set, we can conclude that these instances are difficult to solve for these two heuristics. Note however, that in all the small instances, included this set, we run our method for a very short CPU time (0.1 seconds), in line with previous works.

In the following experiment we undertake to compare SO with the well-known LocalSolver. We run this method with formulation F2 shown in Section 2. We run both methods with a time limit of 60 seconds. Table 9 shows the results.

		LocalSolver				**SO**			
Set	$\boldsymbol{n}$	$\boldsymbol{f(M)}$	**Dev**	**Best**	**Time**	$\boldsymbol{f(M)}$	**Dev**	**Best**	**Time**
GKD-b2	50	1123.3	0.0	10	4	1123.3	0.0	10	0
GKD-b2	150	1181.0	0.3	7	34	1181.5	0.3	6	15
GKD-c2	500	58.2	21.9	0	47	74.6	0.0	10	19
SOM-a2	50	4.0	90.0	1	3	41.0	0.0	10	0
MDG-b2	500	56.6	83.5	0	59	343.5	0.0	10	25
GKD-b3	50	977.9	0.0	10	12	969.9	0.7	7	0
GKD-b3	150	1072.5	0.1	8	26	1068.5	0.6	4	47
GKD-c3	500	57.4	11.5	0	45	64.9	0.0	10	41
SOM-a3	50	0.0	100.0	0	1	15.0	25.0	5	0
MDG-b3	500	35.1	65.3	0	57	104.7	0.0	10	45

Table 9. Comparison of SO with general purpose heuristic.

Table 9 shows that, in general terms, SO obtains better solutions than LocalSolver in shorter running times. This is to be expected since SO is customized to the Capacitated Dispersion Problem while LocalSolver is a general solver, although it is worth mentioning that it also implements complex metaheuristics, and is nowadays a reference method, given its remarkable performance in many combinatorial optimization problems. As a matter of fact, in the GKD-b2 sets ($n = 50$ and $n = 150$), LocalSolver and SO perform very similar; in the GKD-b3 sets ($n = 50$ and $n = 150$), LocalSolver performs better than SO; but in the rest of the sets, SO clearly outperforms LocalSolver. To confirm the superiority of our method, we perform the two statistical tests described above, the Wilcoxon and the Signs, and obtain in both cases a p −value of 0.00, thus indicating that there are significant differences between both methods. Table 10 in the Appendix shows the individual results of the SO method on the 100 instances in our study.

Results in Table 9 also confirm that the SOM sets of instances are a challenge for heuristic methods, and therefore we can conclude that there is still room for improvement in heuristic developments for this problem.

To complement the analysis above, we run the three heuristic methods under comparison for a relatively long running time (500 seconds) on a representative MDG-b large instance. Figure 9 shows the search profiles in which the best solution obtained with each method is depicted to see its evolution over the time.

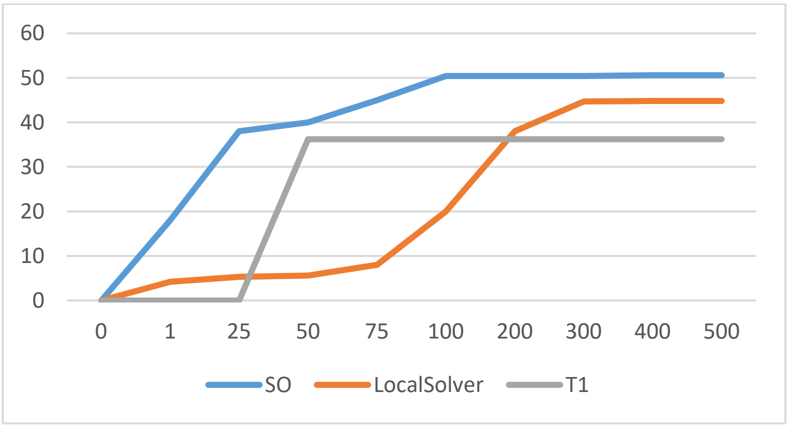

Figure 8. Search profiles on MDG-b instance.

Figure 8 shows that SO is the leading method in the entire search period. LocalSolver requires a bit longer setup time, and it starts to obtain high-quality solutions after 100 seconds of exploration. Finally, since T1 is a simple method, once it obtains a solution, there is no further improvement. All the methods stagnate after 300 seconds, and therefore there is no point on running them longer. Figure 9 shows a similar search profile for a GKD-c instance.

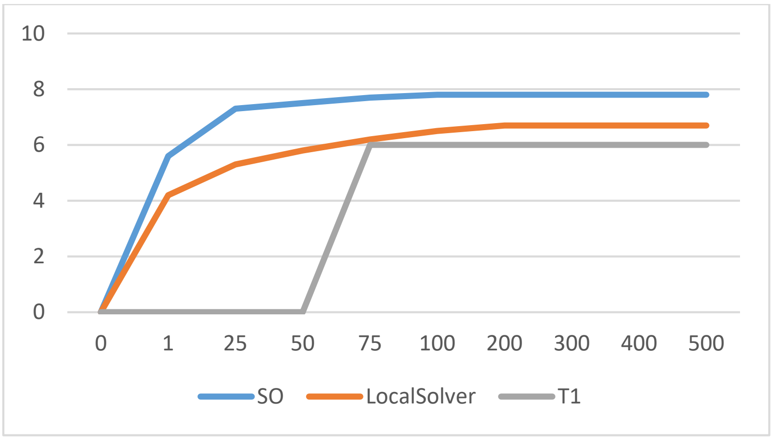

Figure 9. Search profiles on GKD-b instance.

Conclusions

This paper proposed several new hybrid heuristics for the capacitated dispersion problem. The heuristics used components of GRASP, variable neighborhood descent (VND), and strategic oscillation (SO). To find a good configuration for our best heuristic, we considered four constructive procedures (including constructive and destructive neighborhoods) with a parameter to control the randomization in the selection process, and another parameter to set the level of oscillation in the SO. Our extensive preliminary experimentation, reported in Tables 2, 3, 4, and 5, set appropriate values for these parameters, providing a good balance among the three methodologies in terms of search intensification and diversification.

We had a twofold goal for this work, to experiment with the hybridization of GRASP, VNS, and SO and, in the process, to develop a state-of-the-art procedure for the capacitated dispersion problem. We believe that we have achieved the first goal with the proposed design, simply called SO. The merit of this hybrid method is that it preserves the main characteristics of each element while combining their capabilities. In terms of our second goal, the results reported in Tables 6, 7, and 8 are very strong in favor of SO. In particular, the comparisons with Cplex, Localsolver, and a previous specific method, T1, clearly show that SO is able to obtain high-quality solutions in short computational times.

Acknowledgement

This research has been partially supported by the National Science Secretary of Panama (SENACYT) project ref. N-78-2016, and by the Spanish Ministry with grant ref. PGC2018-0953322-B-C21/MCIU/AEI/FEDER-UE.

References

Duarte A., Martí R. (2007) Tabu search and GRASP for the MDP. European Journal of Operational Research 178, 71-84.

Duarte, A., Sánchez-Oro, J., Mladenovic, N., Todosijevic, R. (2018) Variable Neighborhood Descent, In: Martí, Resende, Pardalos (Eds) Handbook of Heuristics, Springer.

Feo, T., Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. Journal of Global Optimization 6, 109-133.

Gallego, M., Duarte, A., Laguna, M., Martí, R. (2009) Hybrid heuristics for the maximum diversity problem. Computational Optimization and Applications, 44(3):411.

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. (1997) Tabu search. Kluwer, Norwell, MA.

Hansen, P., Mladenović, N., Todosijević, R., Hanafi, S. (2017). Variable neighborhood search: basics and variants. EURO Journal on Computational Optimization, 5(3), 423-454.

Martí, R., Duarte, A., Gallego, M. (2019) MBPLIB- Maximum Diversity Problem Library, http://grafo.etsii.urjc.es/optsicom.

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

Martí, R., Sandoya F. (2013) GRASP and Path Relinking for the equitable dispersion problem. Computers and Operations Research 40, 3091-3099.

Martínez-Gavara, A., Landa-Silva, D., Campos, V., Martí, R. (2017a) Randomized Heuristics for the Capacitated Clustering Problem. Information Sciences 417, 154-168.

Martínez-Gavara, A., Campos, V., Laguna, M., Martí, R. (2017b) Heuristic Approaches for the Maximum MinSum Dispersion Problem. Journal of Global Optimization 67, 671-686.

Mjirda, A., Todosijević, R., Hanafi, S., Hansen, P., Mladenović, N. (2017). Sequential variable neighborhood descent variants: an empirical study on the traveling salesman problem. International Transactions in Operational Research, 24(3), 615-633.

Palubeckis, G. (2007) Iterated tabu search for the maximum diversity problem. Applied Mathematics and Computation, 189:371-383.

Prokopyev O.A., Kong N., Martinez-Torres D.L. (2009) The equitable dispersion problem. European Journal of Operational Research 197, 59-67.

Resende M.G.C., Martí R., Gallego M., Duarte A. (2010) GRASP with path relinking for the max-min diversity problem, Computers and Operations Research 37, 498-508.

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.

Sandoya, F., Martínez-Gavara, A., Aceves, R., Duarte, A., Martí, R. (2018) Diversity and Equity Models, In: Martí, Resende, Pardalos (Eds.) Handbook of Heuristics, 979-998 Springer.

Silva, G.C., Andrade, M.R.Q., Ochi, L.S., Martins, S.L., Plastino, A. (2007) New Heuristics for the Maximum Diversity Problem, Journal of Heuristics 13(4), 315-336.

Silva G.C., Ochi L.S., Martins S.L. (2004) Experimental Comparison of Greedy Randomized Adaptive Search Procedures for the Maximum Diversity Problem. In: Ribeiro C, Martins C Simone L. (Eds.), Experimental and efficient algorithms, vol. 3059. Springer: Berlin. 498-512.

Appendix

Table 10 shows the individual results obtained with SO on each instance. CPU times (Time) are reported in seconds.

		Capacity 0.2		Capacity 0.3	
Instance set	id	SO value	SO Time	SO value	SO Time
	11	147.2	0	132.8	0
	12	178.1	0	154.7	0
	13	96.1	0	77.5	0
	14	84.6	0	71.2	1
GKD-b	15	154.9	0	134.0	0
$n = 50$	16	77.7	0	63.1	0
	17	41.8	0	27.9	0
	18	108.5	0	104.4	0
	19	119.1	0	102.5	0
	20	115.3	0	101.8	0
	41	163.4	13	153.8	33
	42	84.1	19	70.2	70
	43	63.1	22	53.9	71
	44	103.3	13	87.6	44
GKD-b	45	106.4	14	94.5	34
$n = 150$	46	124.5	15	109.7	27
	47	162.2	14	154.8	61
	48	98.1	12	85.7	32
	49	166.3	18	158.3	50
	50	110.1	12	100.0	52
	1	7.2	18	6.8	45
	2	7.6	15	6.3	41
	3	7.2	22	6.5	37
	4	7.5	15	6.5	35
GKD-c	5	7.7	21	6.4	45
$n = 500$	6	7.2	19	6.5	45
	7	7.6	18	6.5	43
	8	7.8	14	6.5	44
	9	7.4	23	6.4	42
	10	7.4	20	6.5	32

	11	4.0	0	2.0	0
	12	4.0	0	1.0	0
	13	5.0	0	1.0	0
	14	4.0	0	2.0	0
	15	4.0	0	2.0	0
SOM-a	16	4.0	0	1.0	0
$n = 50$	17	4.0	0	2.0	0
	18	4.0	0	1.0	0
	19	4.0	0	2.0	0
	20	4.0	0	1.0	0
	1	37.9	27	8.2	34
	2	32.0	22	12.4	47
	3	33.4	24	7.2	39
	4	34.0	30	8.1	38
	5	36.2	23	10.3	39
MDG-b	6	31.8	26	9.9	57
$n = 500$	7	28.5	19	13.3	56
	8	36.1	32	11.6	55
	9	38.3	21	13.2	61
	10	35.3	30	10.5	27

Table 10. Individual results of SO heuristic.