Memetic Computing
https://doi.org/10.1007/s12293-020-00318-1

REGULAR RESEARCH PAPER

The capacitated dispersion problem: an optimization model and a memetic algorithm

Rafael Martí[1] · Anna Martínez-Gavara[1] · Jesús Sánchez-Oro[2]

Received: 3 July 2020 / Accepted: 7 December 2020

Abstract
The challenge of maximizing the diversity of a collection of points arises in a variety of settings, and the growing interest of dealing with diversity resulted in an effort to study these problems in the last few years. Generally speaking, maximizing diversity consists in selecting a subset of points from a given set in such a way that a measure of their diversity is maximized. Different objective functions have been proposed to capture the notion of diversity, being the sum and the minimum of the distances between the selected points the most widely used. However, in all these models, the number of points to be selected is established beforehand, which in some settings can be unrealistic. In this paper, we target a variant recently introduced in which the model includes capacity values, which reflects the real situation in many location problems. We propose a mathematical model and a heuristic based on the Scatter Search methodology to maximize the diversity while satisfying the capacity constraint. Scatter search is a memetic algorithm hybridizing evolutionary global search with a problem-specific local search. Our empirical analysis with previously reported instances shows that the mathematical model implemented in Gurobi solves to optimality many more instances than the previous published model, and the heuristic outperforms a very recent development based on GRASP. We present a statistical analysis that permits us to draw significant conclusions.

Keywords Maximum diversity · Metaheuristics · Mathematical formulations · Empirical comparison

1 Introduction

The problem of maximizing diversity deals with selecting a subset of elements from a given set, in such a way that the diversity among the selected elements is maximized. The most studied model is the Maximum Diversity Problem, MDP, in which the diversity is computed as the sum of the distances between the pairs of selected points. First papers on this topic can be traced back to the nineties, when Ghosh [6] proved the NP-completeness of the MDP, proposed a multi-start algorithm, and tested it on small instances. In line with these developments, we can find some seminal papers by Glover et al. [10, 11] with simple heuristics for the MDP, where the authors pointed out that different versions of this problem may include additional constraints, and their objective is to design heuristics with simple moves for transitioning from one solution to another to allow them to be adapted to multiple settings. In particular, the authors applied this model to the conservation of the crane family under resource constraints in the context of preserving biological diversity. More recently, complex metaheuristics have been proposed to them [3, 15].

As stated in Kuo et al. [12], there are basically two approaches to model diversity maximization: the MDP, also known as the Max-Sum problem, and the Max–Min problem (MMDP), although other variants have also been studied [19]. In the Max-Sum, we maximize the sum of the distances, while in the Max–Min, we maximize the minimum distance between the pairs of selected points. Pareño et al. [24] recently showed that both models produce solutions of a very different structure, as it is illustrated in Fig. 1 that depicts the optimal solutions obtained with these two models (on a MDP public domain instance with

✉ Rafael Martí
Rafael.Marti@uv.es

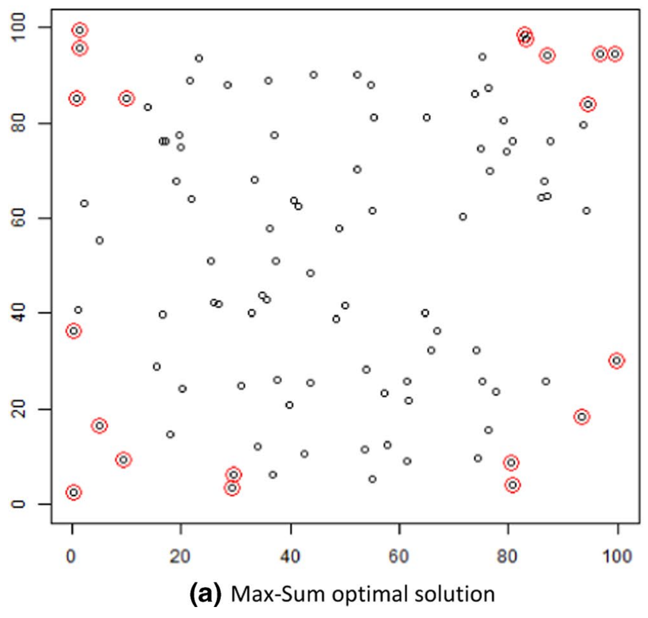
Anna Martínez-Gavara
gavara@uv.es

Jesús Sánchez-Oro
jesus.sanchezoro@urjc.es

1 Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Valencia, Spain

2 Department Computer Sciences, Universidad Rey Juan Carlos, Madrid, Spain

Published online: 05 January 2021



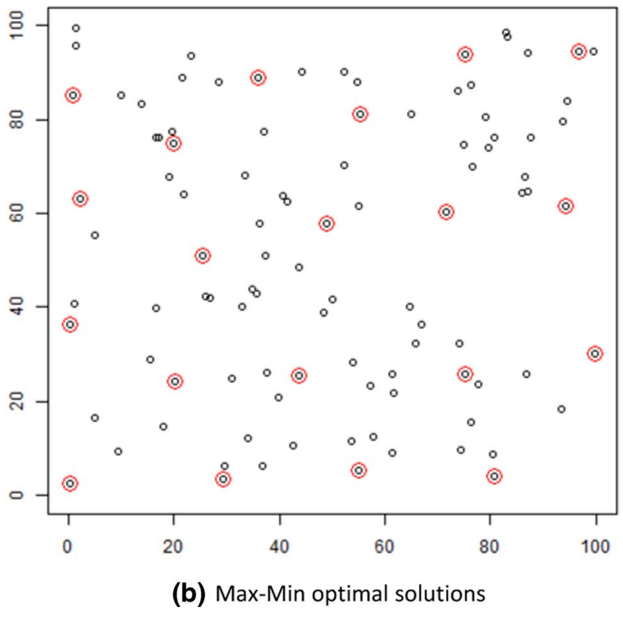


Fig. 1 Optimal solutions of a GKD-d instance with $n = 100, m = 20$

100 elements where 20 of them are selected). We can see that while the Max-Sum tends to allocate the points in the outer part of the region (solutions space), the Max–Min distribute them in a uniform way over the entire region. Both models have received a lot of attention in the last ten years, and the recent work by Parreño et al. [24] discloses the properties of the solutions obtained with each one. In general, the authors recommend the use of the Max–Min model.

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 $i, j \in V$. The MMDP consists in selecting a subset $M \subseteq V$ of m elements ($|M| = m$) in such a way that the minimum distance between the chosen elements is maximized. The Max–Min diversity problem, also known as m-dispersion problem, can be trivially formulated [26] by simply considering its objective function over the set M:

$$\begin{aligned} \text{Maximize} \quad & f(M) = \min_{i,j \in M} d_{ij} \\ \text{subject to} \quad & M \subseteq V \\ & |M| = m \end{aligned} \tag{1}$$

In spite of the early interest in constrained models, the extensive literature on diversity problems is mostly based on selecting a fixed number of points, m, avoiding the use of additional constraints. In this paper, we consider a model recently introduced [27], where the standard size constraint shown above is replaced with a capacity constraint. This new model has applications in location problems, to establish the minimum level required to provide a service. It is known as Capacitated Dispersion Problem (CDP), and can be easily formulated in mathematical terms in a similar way than the MMDP. Given a graph $G = (V, E)$, let c_i be the capacity of node $i \in V$, and B the total capacity required. The CDP is formulated with binary variables x_i that take value 1 if node (element) i is selected, and 0 otherwise. The set of selected elements is computed as $M = \{i \in V : x_i = 1\}$.

$$\begin{aligned} \text{Maximize} \quad & f(M) = \min_{i,j \in M} d_{ij} \\ \text{subject to} \quad & \sum_{i=1}^{n} c_i x_i \geq B \\ & x_i \in \{0, 1\} \qquad i = 1, \ldots, n \end{aligned} \tag{2}$$

Very recently, Peiró et al. [25] proposed a solving method based on the GRASP methodology [4, 26] to solve the CDP. This heuristic method implements a strategic oscillation [8] for an efficient search of the solution space. It is able to outperform the previous heuristic proposed by Rosenkrantz et al. [27], as empirically shown on their experimentation over 100 instances. Additionally, these authors proposed a mathematical model to optimality solve small and medium size instances.

When analyzed both the model and the heuristic in Peiró et al. [25], we identified some limitations which motivated our current development. On one hand, the model is a straightforward adaptation of the integer linear model introduced many years ago for the Max–Min unconstrained problem [12], which was recently outperformed by the model proposed by Sayyady and Fathi [28]. We therefore propose here to adapt this new model to the constrained variant. Our analysis will reveal that it is able to solve medium size CDP instances on a small fraction of the time employed by the previous model, and it can even solve some large size instances (with up to 500 elements). Regarding the previous GRASP heuristic, it mainly relies on an improvement method (VND) limited to two neighborhoods. The flat landscape (plateau) of this problem that maximizes a minimum value is a challenge for local search methods guided by changes in the objective function. In line with Neri and Cotta [21], we considered that the characteristics of memetic algorithms and their diversification power make them suitable to deal with this type of problem. In particular, we propose a new local search method based on a number of neighborhoods that are dynamically adjusted and we coupled it with a combination method, thus creating a memetic algorithm. Our empirical analysis will show that it obtains better solutions than the GRASP heuristic. It is worth mentioning that our new designs can be applied to other combinatorial optimization problems. In particular, the dynamic nested neighborhood exploration in local search, and the combination method based on the Path Relinking in memetic algorithms, can be easily adapted to other settings.

2 Memetic algorithms and the scatter search methodology

In this section we describe the elements and strategies of the Scatter Search methodology in connection with Memetic Algorithms. In general terms, we may say that Memetic Algorithms (MAs) constitute a class of solving methods with a particular structure given by population, generational evolution, and local search. In this context, Memetic Computing (MC) can be seen as the subject of study related to the algorithmic structures composed of multiple operators. In short, MAs are a subset or instance of MC. According to Neri and Cotta [20], "*Memetic computing is a broad subject which studies complex and dynamic computing structures composed of interacting modules (memes) whose evolution dynamics is inspired by the diffusion of ideas*". We limit our description here to the particular case of Scatter Search, a metaheuristic methodology belonging to the family of MAs.

Unlike other well-known metaheuristic methodologies, Memetic Algorithms are not a specific algorithm, but a more general methodology. Neri and Cotta [21] characterize them as a flexible class of algorithms, containing the previous evolutionary algorithms, and combining global and local search. In a basic MA, the initial population is generated following a systematic procedure. Then, the method iterates over a main loop that basically consists of three elements: cooperate, improve, and compete. Cooperate and Improve constitute the core of the MA, and as described in Neri and Cotta [20], diversity management is one of the key points in the interaction of these elements.

A key point in the design of any evolutionary method is the diversity management in the population. Laguna and Martí [13] proposed to solve the maximum diversity problem to create an initial population with a set of very different (diverse) solutions. Tirronen and Neri [29] proposed diversity self-adaptation techniques within the parameter settings of Differential Evolution (DE). In particular, scale factor, crossover rate, and population size are adaptively controlled with a mechanism based on the fitness diversity. An empirical comparison shows the merit of the new approach with respect to the standard DE.

It must be noted that this is not the first time that MAs have been applied to diversity problems. As a matter of fact, Wang et al. [30] proposed a hybrid method combining MAs and tabu search for the classic, unconstrained, maximum diversity problem. An interesting feature of that method is the combination operator applied to generate good offspring solutions. In particular, the authors applied the tabu search strategies known as "strongly determined" and "consistent variables", which basically specify those elements that contribute most to good solutions explored in the search process, in order to favor their selection in future solutions. The good results obtained with this method on the unconstrained version of the diversity problem triggered our interest to create a new MA for the constrained diversity problem. Specifically, we consider the Scatter Search methodology [18], which implements a relatively simple and effective MA.

Scatter search (SS) is a heuristic methodology [11] that explores solution spaces by evolving a set of solutions. SS follows the principle in Cotta et al. [1] that memetic algorithms integrate evolutionary methods with local search, performing and effective search of the solution space. SS is based on this framework, providing an interplay between global search (population based) and local search (individual based) that is able to obtain high-quality solutions to difficult optimization problems.

It has been well-documented that efficient MAs generate the initial population with a problem specific method, instead of a completely random generation typical of Genetic Algorithms (GAs). In line with that, SS starts with the application of a constructive method, called *diversification generation method*. Then, it selects a subset of solutions called **reference set** (*RefSet*), which is basically a collection of both high quality solutions and diverse solutions selected from the population, called *P*.

The reference set evolves by applying four additional methods: reference set update method, subset generation method, combination method, and Improvement method. We briefly describe now their role in the entire SS method. Reference set update consists in the updating mechanisms to keep the RefSet with the best solutions found so far. Note that the term best here refers to both quality and diversity. Figure 2 shows a graphical representation of the SS flow that illustrates how it works, and the interaction of its methods.

The **subset generation method** specifies the subsets of solutions that are combined. It usually consists of pairs of solutions but other designs are also considered. Instead of a sampling mechanism, like the so-called roulette wheel implemented in many GAs, all pairs of solutions in the Reference Set are considered for combination in SS. This is why the RefSet has to be kept small (around 10 solutions).

A **combination method** is applied to generate new solutions from each pair in the RefSet. As mentioned with the generation method, the combination usually exploits problem characteristics and solution representation to obtain high quality outputs. The strategy known as path relinking (PR), originally proposed within the tabu search methodology [8], has been extensively used as a combination method. This is the case here, since we also propose a PR for the capacitated dispersion problem.

The SS method operates in short as follows. It first generates *PSize* solutions to populate *P*, as shown in Fig. 2. Then, the main loop of method performs iterations as long as at least one solution is new in the *RefSet*. A solution is new when it has not been previously combined. If the RefSet contains new solutions, SS considers all pairs of solutions involving the new ones to apply the combination method,otherwise it stops. This avoids duplications of solutions from the application of the combination method to the same subset more than once. The method finishes when no new solution qualifies to enter in the *RefSet,* which is only updated if the new solution improves the worst one in the *RefSet*. The best solution in the *RefSet* is the output of the method. See Laguna and Martí [13] for a complete description of the method.

3 Mathematical model

Peiró et al. [25] proposed the linear integer formulation for the CDP (3) based on an upper bound *D* on the distances values, which is equivalent to the simpler (nonlinear) model shown above (2). It is based on binary variables y_{ij} that take the value 1 if and only if $x_i = x_j = 1$. In this way, it avoids the product of the *x*-variables.

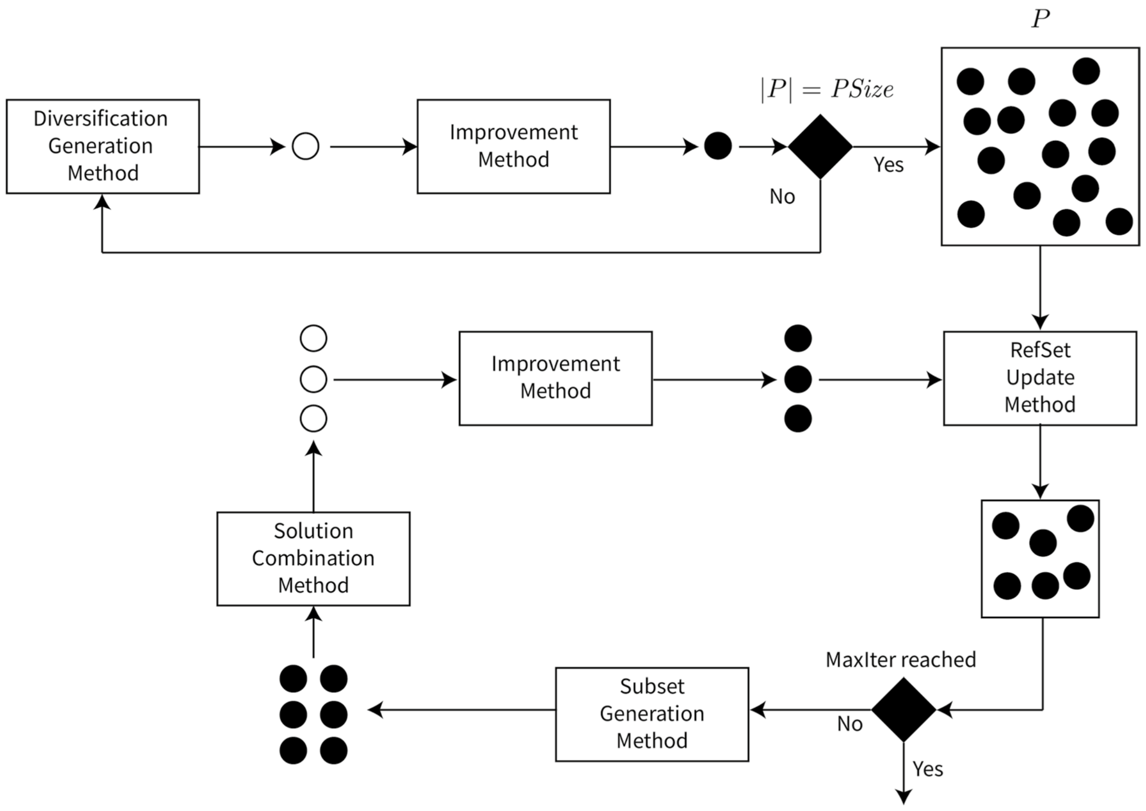


Fig. 2 Scatter Search diagram

$$
\begin{aligned}
&\text{Maximize} && z && \\
&\text{subject to} && && \\
& && \sum_{i=1}^{n} c_i x_i \geq B && \\
& && y_{ij} \leq x_i && 1 \leq i,j \leq n \\
& && y_{ij} \leq x_j && 1 \leq i,j \leq n \\
& && x_i + x_j \leq y_{ij} + 1 && 1 \leq i,j \leq n \\
& && z \leq d_{ij} y_{ij} + D\left(1 - y_{ij}\right) && 1 \leq i,j \leq n \\
& && y_{ij} \in \{0,1\}, x_i \in \{0,1\} && 1 \leq i,j \leq n
\end{aligned}
\tag{3}
$$

As will be shown in our empirical experience, this formulation is able to solve the small and some medium size CDP instances to optimality. We propose now an alternative formulation that is able to solve large instances.

Sayyady and Fathi [28] solved the MMDP with the node packing problem, in which given a threshold value l, a graph $G(l)$ is defined with the set V of n nodes of original graph $G = (V, E)$, and the set of edges $E(l) = \{(i,j) \in E\colon d_{ij} < l\}$. The node packing problem consists of finding a maximum cardinality subset of nodes so that no two nodes in this subset are adjacent to each other. It can be formulated in mathematical terms (4) with binary variables, x_i, indicating if node i is selected as:

$$
\begin{aligned}
&\text{Maximize} && \sum_{i=1}^{n} x_i && \\
&\text{subject to} && x_i + x_j \leq 1 && \forall (i,j) \in E(l) \\
& && x_i \in \{0,1\} && i = 1, \ldots, n
\end{aligned}
\tag{4}
$$

An optimal solution of the node packing problem in $G(l)$ provides a set of points with minimum distance larger than or equal to l. The authors solve a sequence of node packing problems for different values of l until they obtain a set of m points, which turns out to be the optimal solution of the Max–Min model. Specifically, they implement a systematic search in the interval $l \in \left[d_{min}, d_{max}\right]$, where d_{min} and d_{max} are the minimum and maximum values respectively among all the distances in the graph. The method performs a binary search over the ordered set of different distances in the graph, based on the minimum distance between consecutive values.

We adapt the node packing model to the CDP, and propose to solve model (5) for different values of the parameter l, which is completely different than the previous model for the CDP shown in (4), and only shares with it the capacity constraint:

$$
\begin{aligned}
&\text{Maximize} && \sum_{i=1}^{n} x_i && \\
&\text{subject to} && x_i + x_j \leq 1 && \forall (i,j) \in E(l) \\
& && \sum_{i=1}^{n} c_i x_i \geq B && i = 1, \ldots, n \\
& && x_i \in \{0,1\} && i = 1, \ldots, n
\end{aligned}
\tag{5}
$$

Note that the number of selected elements is not important in the CDP, so when solving (5), we do not need to consider the objective function, and we simply check its feasibility, which is significantly faster than obtain the optimal solution.

We implement a binary search [24] over the threshold value l that defines our model. We start by making l to the mean value in $\left[d_{min}, d_{max}\right]$, and solve the decision problem (5) described above. If we obtain a feasible solution, we have a feasible solution M of the original CDP with a value $f(M) \geq l$, and then we resort to the interval $\left[f(M), d_{max}\right]$,otherwise we consider $\left[d_{min}, l\right]$. We set now l as the mean value of the new interval and proceed in this way. We will compare in Sect. 6.2 both models, the original one (3) introduced in Peiró et al. [25], and (5), our adaptation of the Sayyady and Fathi [28], when solving the CDP on the 100 previously reported instances.

4 Previous heuristic methods

Rosenkrantz et al. [27] proposed the first heuristic for this problem, based on a binary search over the distances in the problem data. Their method, called T1, has a performance guarantee of 2, which means that for an instance with distances satisfying the triangle inequality, its optimal value divided by the value of the solution obtained with T1 is lower than 2. This property makes the method appealing from a theoretical perspective.

The T1 method can be summarized in a few steps as follows. It first sorts the elements in non-increasing capacity order, and create a list called Site_List. Then, in a second step, it sorts the inter-site distances in non-increasing order, eliminating duplicates. Let the resulting sorted list of distances be sorted in the array D such that $D[1] > D[2] > \ldots > D[t]$. Finally, the method carries out a binary search over D to find the index i such that for $\alpha = D[i]$, the procedure $Greedy\_Try(\alpha, B)$ returns "success" and for $\alpha' = D[\,i-1\,]$ it returns "failure". *Greedy_Try*, as its name indicates, is a greedy function that tries to create a solution of capacity larger than or equal to B, adding elements one by one. It is worth mentioning that this greedy procedure is based on removing from the input graph all the edges with a distance lower than α, thus obtaining a solution with a value larger than or equal to this threshold. Note that the exact method that we proposed in Sect. 3 to solve this problem, is based on the same principle.

Peiró et al. [25] proposed a complex metaheuristic to obtain high quality solutions for the CDP. In contrast to T1, their method, called SO, does not have a worst-case analysis, and theoretically speaking it could perform arbitrarily bad. However, their statistical study on a collection of 100 public domain instances proved that in practice it works remarkably well, obtaining much better results than T1. SO is basically a GRASP [26] with a Strategic Oscillation post-processing [8].

Given the set V with n nodes, the constructive procedure of SO performs consecutive steps, adding one node at a time, to produce a set (solution). As it is customary in the GRASP methodology, the constructive method computes an evaluation,$eval(i)$ for each candidate element i to be added to the partial solution under construction M. Restricted candidate list RCL contains the elements $i \in V \backslash M$ with a relative good evaluation. At each step, the method randomly selects an element in RCL and adds it to M until the capacity constraint is satisfied.

To compute the evaluation function, the method first calculates, $d_i = \sum_{j \in M} d_{ij}$ for any element $i \in V \backslash M$, and $d_{max} = \max_i d_i$. Similarly, from the capacity values c_i, it computes $c_{max} = \max_i c_i$. To put together distance and capacity values in a single expression, it uses the maximum values computed to adjust them to the range [0, 1] as follows:

$$eval(i) = w\frac{d_i}{d_{max}} + (1 - w)\frac{c_i}{c_{max}}$$

where the relative weight of distance and capacity, is empirically adjusted making $w = 0.6$.

Once a feasible solution is constructed with the method above, the SO procedure improves it with a VND method. As the authors stated, their VND method is limited since it only applies two neighborhoods, $N_1(M)$ and $N_2(M)$. The first one exchanges one element in the solution with one element out of it, while the second one performs a similar operation but exchanging two elements. The method only performs feasible moves, which means that the capacity constraint is always accomplished.

The VND implemented in SO 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. When an improving move is performed, the method returns to the first neighborhood. When both neighborhoods have been explored and no improvement is found, VND stops.

The SO method takes its name from the final strategic oscillation component that tries to improve the solution obtained with GRASP and VND. In particular, SO adds extra elements to the solution, which are actually unnecessary to fulfill the capacity constraint, and then removes other elements from the resulting extended solution. The addition and removal of elements creates an oscillation pattern that has a final objective to alter the solution structure in a beneficial way. The number of elements to be added and removed depend on the capacity level, which is artificially increase on a 60% of B according to the computational testing presented.

In the next section we propose a new heuristic method for this problem. The motivation of our current development is to improve the results of SO, and to design advanced search mechanism to perform a more efficient search of the solution space that can be applied to other optimization problems.

5 Scatter search for the capacitated dispersion problem

From the five SS methods described above, two of them are generic, and do not need to be customized for the specific problem solved. In particular, the reference set update and the subset generation method perform as described above. In this section, we describe our proposals to implement the other three methods to the capacitated dispersion problem.

5.1 Diversification generation method

Consider an input graph $G = (V, E)$, where V is the set of n nodes and c_i the capacity of each node i, E is the set of edges, and d_{ij} is the distance associated to edge $(i,j) \in E$ (i.e., the distance between elements i and $j \in V$).

The diversification generation method DGM is a construction procedure that performs steps to produce a solution applying a semi-greedy algorithm. The set M represents the partial solution under construction. At each step, DGM selects a candidate element $u^* \in V \backslash M$ with a large capacity value and a large distance to the elements in the partial solution M. Specifically, for each element $i \in V \backslash M$, the algorithm first computes the minimum distance d_i between element i and the elements in M.

$$d_i = min\{d_{ij} : j \in M\}$$

Then, it constructs a candidate list CL with the unselected elements $i \in V \backslash M$ with a distance value d_i within a fraction α $(0 \le \alpha \le 1)$ of the maximum distance $d^* = max\{d_i : i \in V \backslash M\}$.

$$CL = \{i \in V \backslash M : d_i \ge \alpha d^*\}$$

The method selects the element $u^* \in CL$ with the largest capacity value: $c_{u^*} = max_{i \in CL} c_i$ and adds it to M. DGM performs iterations in this fashion, updating the d_i-values and CL in each step, until the sum of the capacities of the elements in M is larger than or equal to B. At this stage, M represents a complete solution of the problem.

It is clear that a relatively large value of α (close to 1) will make the candidate list CL relatively small, and thus the selection of u^* will be mainly guided by the objective function (i.e., selecting an element far from the already selected). However, lower values of α will create a larger CL, and the selection of the element in CL with the largest capacity will have the opportunity of selecting an element with a relatively large capacity, thus being guided by both distance and capacity. The value of α will be empirically adjusted in our computational experience.

DGM starts by simultaneously selecting two elements to become part of M. In particular, the method computes the largest distance in the graph

$$d^* = max\left\{d_{ij} : (i,j) \in E\right\}$$

and initializes the set M by adding the two nodes at distance d^*. Then, it applies the steps described above until the minimum capacity B is satisfied, and a solution is completed. We now apply DGM to create a second solution with same procedure but this time considering the second largest distance in the graph and its associated endpoints (instead of the largest one considered in the first construction). Specifically, we initialize M in the second construction with the two elements at a distance equal to the second largest distance in the graph, and then apply several steps, computing the d_i-values and CL in each one, until the solution is completed. DGM proceeds in this way, initializing M with different pairs of points and applying the steps above, until *PSize* solutions have been created to populate the set P.

5.2 Improvement method

Given a solution M obtained with the diversification generation method described in Sect. 5.1, its objective function value is given by the expression.

$$f(M) = \min_{i,j \in M} d_{ij}$$

This means that there are, at least two elements directly involved in the objective function computation. Let d^* be this objective function value ($d^* = f(M)$) and i^* and j^* be the two elements in M at distance $d^* = d_{i^*j^*}$ that we call *critical*. To improve this solution, it is clear that we have to replace the critical elements in M for other elements at a larger distance. Note that in some cases we may have more than two critical elements in a solution.

Our improvement method first selects one of the critical elements, j^*, and then the method computes the candidate elements that can replace j^*. To do that, we first compute, for each element not in the solution, $i \in V \backslash M$, its minimum distance $d_i(j^*)$ to the elements in the solution without considering j^*. We exclude j^* from this calculation because it will not be part of the solution after the replacement.

$$d_i(j^*) = min\left\{d_{ij} : j \in M \backslash \{j^*\}\right\}$$

We consider a swap move in which j^* is replaced in the solution with an element i at a distance $d_i(j^*) > d^*$. In this way, if M only has two critical elements, we directly improve the objective function value when applying this swap move. Otherwise, we would need to replace another critical node (or even several nodes) to increase the minimum distance value in the solution. In any case, we consider that we improve our strategic situation since we are somehow closer to a better solution, we therefore perform the move.

To select the specific element $i \in V \backslash M$ that will replace j^* in M, we first compute a set C of good candidates. Instead of considering the elements satisfying $d_i(j^*) > d^*$, we are more selective, and try to find an element even at larger distance than d^* to M. We consider that if a move is able to eliminate a critical element and increase the objective function value, then it is likely that a near-critical element will become critical in subsequent iterations. This is why we try to replace j^* with an element at a significant larger distance to $M \backslash \{j^*\}$ to avoid a marginal improvement and make a "big change" in the solution value. We therefore create the candidate set as:

$$C = \left\{i \in V \backslash M : d_i(j^*) > \beta d^*\right\}$$

where $\beta > 1$ will be empirically adjusted. We select in C the element $\bar{i}$ with the largest capacity value and perform the move, thus obtaining a solution $M' = M \backslash \{j^*\} \cup \{\bar{i}\}$. If the capacity limit B is not satisfied, we select the second largest element in C according to its capacity value $\bar{\bar{i}}$ and also add it to the solution, thus resulting in $M' = M \backslash \{j^*\} \cup \left\{\bar{i}, \bar{\bar{i}}\right\}$. We keep in this fashion until the capacity limit is satisfied. If the set C is empty, or it does not have enough elements to satisfy the capacity limit after the move, we reduce the value of the parameter β by 0.1 ($\beta = \beta - 0.1$) iteratively until β equals 1. If at this stage, we were not able to replace j^* with a sequence of elements $\bar{i}, \bar{\bar{i}}, \ldots$ taken from C with a sum of capacities satisfying the capacity limit, then we have to resort to the other critical element i^* and check if we can replace it. If we cannot replace any of the two critical elements, then the improvement method stops; otherwise, it re-computes the critical elements of the new solution and computes again the associated swap moves.

It must be noted that we do not consider here a standard swap move, as Peiró et al. [25] did, because we are replacing one element in the solution with an arbitrary number of elements. This number is not set beforehand as in a standard neighborhood, but it is determined in each particular swap depending on the capacity limit. It can be seen as a nested swap neighborhood dynamically adjusted, which makes our method, called DynLS, especially efficient.

A typical way to nest neighborhoods in the metaheuristic literature is the Variable Neighborhood Descent method (VND), which is the case of Peiró et al. [25]. However, DynLS implements a variant with an important difference in the way moves are examined. The standard VND is based on exploring a set of neighborhoods, say $N_k(M)$ for $k = 1, 2, \dots, k_{max}$, that in our case would be the set of solutions that are obtained when we exchange one element in solution M with k elements in $V \backslash M$. Exchanges in this context consist of replacing a selected element with unselected ones verifying the capacity constraint. In a similar way than DynLS, in a VND method, we first identify the critical elements in the solution M and then perform a scan of $N_1(M)$ in search for a swap of a critical node with an element in $V \backslash M$. If this swap results in a feasible move, satisfying the capacity limit B, then it performs the move. If there is no feasible move in $N_1(M)$, then the method resorts to $N_2(M)$ to find a feasible exchange of the critical element with two elements in $V \backslash M$, and so on. In DynLS we do not scan the entire $N_1(M)$ before exploring $N_2(M)$ as VND does, but on the contrary, we only explore the most promising move in $N_1(M)$, and if it does not result in a feasible solution, the method directly tries with a move in $N_2(M)$ linked with it. This can also be considered a compound move [8], often called variable depth methods, constructed from a series of simpler components.

5.3 Combination method

We apply a generalized combination method called **Path Relinking** [13]. This approach generates new solutions by exploring trajectories or paths that connect high-quality solutions, by starting from one of these solutions, called an **initiating solution**, and generating a path towards the other solution, called **guiding solution**. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions. Examples of such attributes include edges and nodes of a graph, sequence positions in a schedule, and values of variables and functions of variables.

Path Relinking (PR) can be considered an extension of the Combination Method of scatter search. Instead of directly producing a new solution when combining two or more original solutions, PR generates paths between and beyond the selected solutions in the neighborhood space. The character of such paths is easily specified by reference to solution attributes that are added, dropped or otherwise modified by the moves executed.

As shown in Fig. 3 (taken from Laguna and Martí [13], two solutions A and B may be already connected with a path, depicted with a solid line, representing that the exploration that originated them performed a sort of local exploration

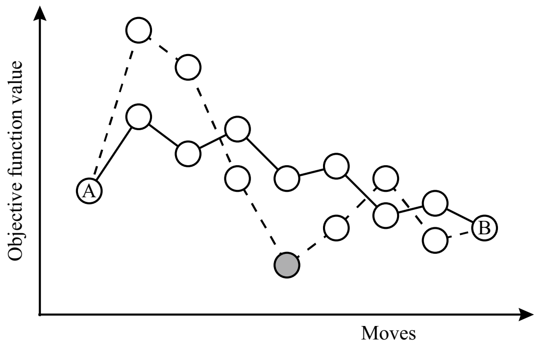


Fig. 3 Path Relinking diagram

that went from A to B through some intermediate solutions (i.e., A and B are linked with that path). The PR method consists in creating a new path, represented in the figure with a dotted line, to relink these solutions in search for a better one (depicted with a dark circle in Fig. 3 for a minimization problem). It may also be that A and B were not previously joined by a search path, as it is our case here where we generate solutions with the method described in Sects. 5.1 and 5.2, but in any case PR generates a path of solutions joining them.

Given two solutions of the CDP, M and N, the path relinking procedure starts with the first solution M, and gradually transforms it into the second one N. To do that, the method basically swaps out elements selected in M with elements selected in N. Note that we are facing the same situation regarding the capacity that we describe above with the local search. In particular, when we replace one element in M we may obtain an unfeasible solution, and we have to consider the addition of further elements until the capacity constraint is satisfied. Elements can be replaced based either on their distance or their capacity, we therefore propose to create two paths from M to N, the first one, $PR_d(M, N)$, based on distance values, and the second one $PR_c(M, N)$, based on the capacities, thus implementing a **Multiple Path Relinking** strategy.

In line with what we do in the improvement method, in $PR_d(M, N)$ we first compute the critical elements at a distance $d^* = f(M)$, and we consider to remove one of them j^* selected at random (from those critical elements). Then, for each element in N not in M, $i \in N \backslash M$, we compute its minimum distance $d_i(j^*)$ to the elements in the solution without considering j^*. Applying a greedy strategy based on the distance, we select the element i^* with the maximum distance. In mathematical terms:

$$d_{i^*}(j^*) = \max_{i \in N \backslash M} d_i(j^*)$$

We replace now j^* with i^* in M. If the resulting solution satisfies the minimum required capacity B, then we have an intermediate feasible solution. Otherwise, we add to M more elements from $N \backslash M$ until it is satisfied. Following the distance criterion, we would select the second largest distance element, and proceed in this fashion. Let $pr_{d1}(M, N)$ be the feasible solution obtained.

We perform now a second step of $PR_d(M, N)$ by swapping elements in $pr_{d1}(M, N)$. Specifically, we apply the same mechanism described above to replace a critical element in this first intermediate solution, with one or more elements in $N \backslash pr_{d1}(M, N)$, thus obtaining a feasible intermediate second solution $pr_{d2}(M, N)$. We proceed in this way until we reach the guiding solution N.

It is clear that the neighborhood employed here is very similar to the one in the improvement method. Note however, that we are restricting the exploration to add only elements present in the guiding solution. As described in Glover and Laguna [8], in PR the objective function is subordinate to the inclusion of attributes of the guiding solution.

We consider a second path to join solutions, M and N. In particular, we remove and add now elements based on their capacity. In $PR_c(M, N)$ we first compute the element j^* with the lowest capacity, and replace it with the element $i^* \in N \backslash M$ with the largest capacity in that set. If the resulting solution is not feasible, we keep adding elements from $N \backslash M$ to this partial solution until it satisfies the capacity limit. These elements are now selected according to their capacity, where the one with the largest capacity comes first, and so on. Let $pr_{c1}(M, N)$ this first intermediate feasible solution. We perform now the second step, swapping out the lowest capacity element in $pr_{c1}(M, N)$ with one or more elements with the largest capacity in $N \backslash pr_{c1}(M, N)$, thus obtaining the second intermediate solution $pr_{c2}(M, N)$ and so on until the path reaches N.

The two paths generated above applied a greedy strategy. The first one the distance, and the second one the capacity. We can therefore say that we are implementing a Multiple Greedy Path Relinking. Resende et al. [26] described other strategies to create such paths. For example, in Greedy Randomized Path Relinking, the authors applied the restricted randomization selection, typical in GRASP, to the selection process to generate the intermediate solutions. We do not explore that variant here, but it may be worth to investigate it as well other extensions of PR as exterior, backward, truncated or evolutionary PR. All these alternatives involve trade-offs between computation time and solution quality.

5.4 Overall scatter search method

We consider two versions of Scatter Search, SS1 and SS2. SS1 is designed to be competitive in running time and it only applies one complete iteration (i.e., the MaxIter parameter shown in Fig. 2 is set to 1). In short, SS1 consists of four steps. In the first step it applies the constructive method DGM described in Sect. 5.1 and generates *PSize* solutions. Since this method relies on the parameter α, we denote it as $DGM(\alpha)$. In the second step, the improvement method DynLS is applied to each constructed solution. Since this method depends on the parameter β, we call it $DynLS(\beta)$. In the third step, SS1 populates the reference set, *RefSet*, with b solutions selected from the set of *PSize* improved solutions (we refer as P to this set). To do that, it first selects the best $b/2$ of them in terms of the objective function. The Reference Set Update method specifies that the remaining $b/2$ solutions are selected according to their diversity, and it is therefore necessary to define a distance measure between two solutions of the problem. Given two solutions of the CDP, M and N, we define its distance, $\delta(M, N)$, as the number of elements that they do not share. In mathematical terms, $\delta(M, N) = |M| + |N| - 2|M \cap N|$. We then use this distance measure to select $b/2$ solutions to complete the reference set.

We look for a solution in P that is not currently in the reference set and that maximizes the minimum distance δ to all the solutions currently in the reference set. If two or more solutions have the same distance value δ, we break the tie by selecting the one with the best objective function value. After including in the RefSet the solution at maximum distance, we update the distance values and repeat this selection $b/2$ times. Finally, in the fourth step, SS1 applies Path Relinking to all the pairs in the RefSet (i.e., it is applied to the $(b^2 - b)/2$ pairs). Each application of Path Relinking returns the best solution found in the path as the output of the method. In our implementation of Multiple Path Relinking, we actually generate two paths for each pair of solutions, so we consider the best solution of the two paths as this output. We call MPR to the set of $(b^2 - b)/2$ solutions obtained at the end of the process, where the best solution in this set is the final output of SS1. Figure 4 summarizes the entire method.

It is clear in Fig. 4 that the SS1 method has three parameters: α, β, and b. We will empirically adjust them in the next section. Although, not explicit in this figure, SS1 returns as the output the best solution obtained after the application of the Path Relinking (i.e., the best solution in MPR).

SS2 starts in the same way than SS1, applying the same four steps, but after them, it performs further iterations as

```
1. Generate PSize solutions with $DGM(\alpha)$
2. Create $P$ with the PSize solutions improved with $DynLS(\beta)$
3. Create the RefSet with the best $b/2$ solutions in $P$
    a. Select the best $b/2$ solutions in $P$ in terms of quality
    b. Select the best $b/2$ solutions in $P$ in terms of diversity
4. Apply Path Relinking to each pair of solutions in the RefSet
```

Fig. 4 SS1 outline

```
1.  Generate $PSize$ solutions with $DGM(\alpha)$
2.  Create $P$ with the $PSize$ solutions improved with $DynLS(\beta)$
3.  Create the RefSet with the best $b/2$ solutions in $P$
      a.  Select the best $b/2$ solutions in $P$ in terms of quality
      b.  Select the best $b/2$ solutions in $P$ in terms of diversity
      c.  NewSolutions = True
While (NewSolutions)
      4.  Apply Path Relinking to each pair of new solutions in RefSet
      5.  Update RefSet with the best solutions obtained
      If RefSet has not changed
            6.  NewSolutions = False
```

Fig. 5 SS2 outline

shown in Fig. 5. Note that SS2 is designed to be competitive in quality, running for longer CPU time than SS1. Solutions in RefSet are now ordered according to quality, where the best solution is the first one in the list. After the four steps outlined in Fig. 4, the Reference Set Update Method is applied again in SS2. In step 5, the method builds a new RefSet with the best solutions, according to the objective function value, from the current RefSet and the set MPR. This set contains the solutions obtained with the application of the Path Relinking (one for each pair of solutions). If RefSet changes, the NewSolutions flag is kept with the TRUE value, indicating that at least one new solution has been inserted in the reference set. Otherwise, it is switched to FALSE in Step 6. SS2 performs more steps as long as new solutions become part of the RefSet. Note however that Path Relinking is only applied to pairs of solutions not combined in previous iterations. SS2 terminates after all pairs of solutions have been subjected to the path relinking method and none of the new solutions are admitted to RefSet.

6 Computational Experiments

This section describes the computational experiments that we performed to test the effectiveness and efficiency of the procedures discussed above. The Scatter Search algorithm (described in Sect. 5 was implemented in Java, the previous methods, T1 and SO (described in Sect. 4, were implemented in C, and the mathematical programming model Sect. 3 was solved with Gurobi. All experiments were conducted on a 2.8 Ghz Intel Core i7 processor with 8 GB RAM. We first describe in Sect. 6.1 the benchmark employed to test the methods, then the preliminary experimentation Sect. 6.2 to set the values of key search parameters. In the last two subsections we describe the competitive testing, first in Sect. 6.3 the mathematical models (our proposal versus the previous integer model), and the comparison of heuristics in Sect. 6.4.

6.1 Problem instances

For the experimentation, we use the public-domain MDPLIB [14] available at http://grafo.etsii.urjc.es/optsicom, which contains several data sets previously employed in different studies on diversity problems, and adapted to the CDP by [25].

In particular, for each original instance these authors randomly generate the capacity value of each node in the range [1, 1000]. Then, they 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. The benchmark for the Capacitated Diversity Problem thus consists of the following 100 instances summarized in Table 1. The benchmark collects the following three sets.

6.1.1 GKD

This data set, originally proposed by Glover et al. [11], contains matrices for which the values were calculated as the Euclidean distances from randomly generated points with coordinates in the 0 to 10 range.: Martí et al. [16]: generated the GKD-b medium size instances with values of $25 \leq n \leq 150$. Duarte and Martí [2] generated the GKD-c large instances with $n = 500$.

6.1.2 SOM

This data set consists of 70 matrices with random numbers between 0 and 9 generated from an integer uniform distribution. Martí et al. [16] created these instances to solve the maximum diversity problem.

Table 1 Sets of instances

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

6.1.3 MDG

This data set was generated by Duarte and Martí [2], and used in Gallego et al. [5] and Palubeckis [23]. The MDG-b 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.

6.2 Analysis of heuristic strategies

In this section, we set appropriate values for the parameters in the search strategies of our heuristics. To avoid the overtraining of the methods, we consider a subset of 30 representative instances in this preliminary experimentation. For each experiment, we report the following performance measures: Average objective function (Value), average deviation with respect to the best solution found in the experiment (DevB), number of best solutions found in the experiment (#Best), and computing time in seconds (Time). Note that both DevB and #Best refer to the solutions found within the experiment and not the best solutions known for these problems.

In our first preliminary experiment we study the parameter α in the constructive method DGM described in Sect. 5.1. In particular, we test three values of this parameter: 0.25, 0.5, and 0.75, and we generate 100 solutions with each value on each instance, and report average values. Additionally, we consider a version (Rnd) in which the value is randomly generated in each construction in the range [0, 1]. The results, shown in Table 2, indicate that this random version performs better than the others (see for example the number of best solutions). We therefore set our constructive method with this strategy in the rest of the experimentation.

In our second preliminary experiment, we study the parameter β in the improvement method DynLS Sect. 5.2, testing three values of this parameter: 1, 1.25, and 1.5, and a random version (Rnd) in which the value is randomly selected in [1, 1.5] at each iteration.

The statistics collected to analyze the outcomes of this experiment, shown in Table 3, are not conclusive. As a matter of fact, the non-parametric Friedman test indicates that the differences are not significant. We represent in a box-and-whisker plot Fig. 6 the deviation values of each variant on each instance to further analyze them.

Table 2 Parameter in constructive method α

α	Value	DevB	#Best	Time
Rnd	48.76	0.24%	25	0.83
0.25	48.28	0.70%	21	0.89
0.5	45.51	2.55%	17	0.79
0.75	42.61	5.48%	13	0.73

Table 3 Parameter in improvement method β

β	Value	DevB	#Best	Time
Rnd	49.86	0.61%	20	1.40
1	50.07	0.57%	24	1.43
1.25	49.98	0.55%	19	1.30
1.5	49.58	1.20%	12	1.45

Figure 6 indicates that DynLS with $\beta = 1$ seems to perform slightly better than the other variants, since the box of the diagram is located in the 0% deviation. Considering that this variant is able to obtain the largest number of best solutions (24 as shown in Table 3), we set $\beta = 1$ for the rest of our experimentation.

In our third preliminary experiment, we analyze the size of the reference set, b. This is a critical search parameter since the scatter search method heavily relies on this set (it performs all the combinations applying path relinking in our case). Table 4 reports the results obtained of applying SS1 with three values of b : 5, 10, and 15.

We can see in Table 4 that as expected, the larger the b value the better the result. However, running times also increase with the size of b, and therefore we have to achieve a balance between quality and CPU time. We then set $b = 5$ in SS1, since this variant is meant to be competitive in time, and $b = 15$ in SS2, since this variant is designed to be competitive in quality.

In our final experiment we evaluate the contribution of each element to the quality of the final solution of our scatter search algorithm. We record the objective function value of the best solution obtained with the diversification generation method (DGM), which is the best value in P. Then, we apply the improvement method to these solutions, and record the best value obtained (DynLS), which is the best value in RefSet. Finally, we apply the Path Relinking combination method, to the solutions in RefSet and record in PR the best value resulting from all these combinations. This value is the output of the scatter search algorithm SS1 that only performs one global iteration. When we update the solutions in the RefSet and apply Path Relinking again to its solutions repeatedly, we obtain the solution of the SS2 method, recorded here in Final output. Figure 7 reports the average values of the four variables, DGM, DynLS, PR, and Final output, over our set of instances.

Figure 7 shows that the diversification generation method, DGM, obtains an average value of 6.0% deviation with respect to the best known solution, which corresponds to an objective function value of 48.8 over the 30 instances in our set. Then, the improvement method is able to raise this value up to 50.1 (represented as a deviation of 3.5% in

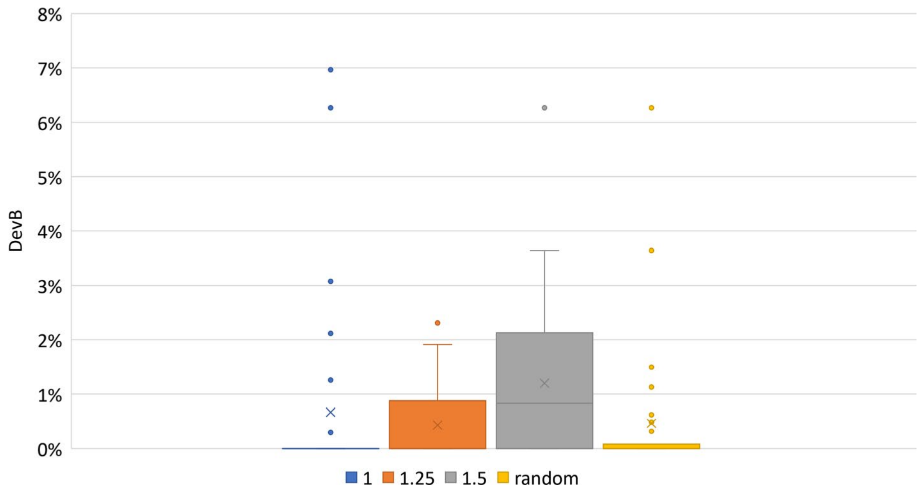


Fig. 6 Box-and-whisker plot of DynLS with different β values

Table 4 Size of the RefSet b

b	Value	DevB	#Best	Time
5	50.84	2.92%	22	1.32
10	51.38	1.04%	24	1.60
15	51.78	0.12%	29	2.28

the figure), which is further improved by the path relinking, obtaining an average of 51.4 (1.0%). Further iterations of the path relinking method marginally improve solutions, achieving a final average result of 51.9, which matches the best known solutions.

6.3 Mathematical models

In this section we compare the previous mathematical model by Peiró et al. [25], called CDP_standard, with our adaptation of the model by Sayyady and Fathi [28] from the Node Packing problem to the CDP, called CDP_NodePacking. Table 5 summarizes the results obtained with Gurobi with each model on each instance for a maximum of 3,600 s. This table reports the average lower bound (LB) the average upper bound (UB), and the average running time in seconds (CPU) on each data set of 10 instances.

Results in Table 5 clearly indicate the superiority of the new model based on the node packing, in terms of the number of instances that it is able to solve and the CPU time. Gurobi with the new formulation is able to solve all the small instances ($n = 50$) and medium instances ($n = 150$) in our testbed in less than 1 s. Additionally, it is able to solve some of the large instances (those in the GKD set with $n = 500$) in a few seconds. The only instances in which it encounters difficulties are the 10 large instances ($n = 500$) in the MDG set, for which it has a relatively large gap (difference between the upper and lower bounds). In all the sets, the previous formulation, CDP_standard, exhibits significant longer running times (in several orders of magnitude). The

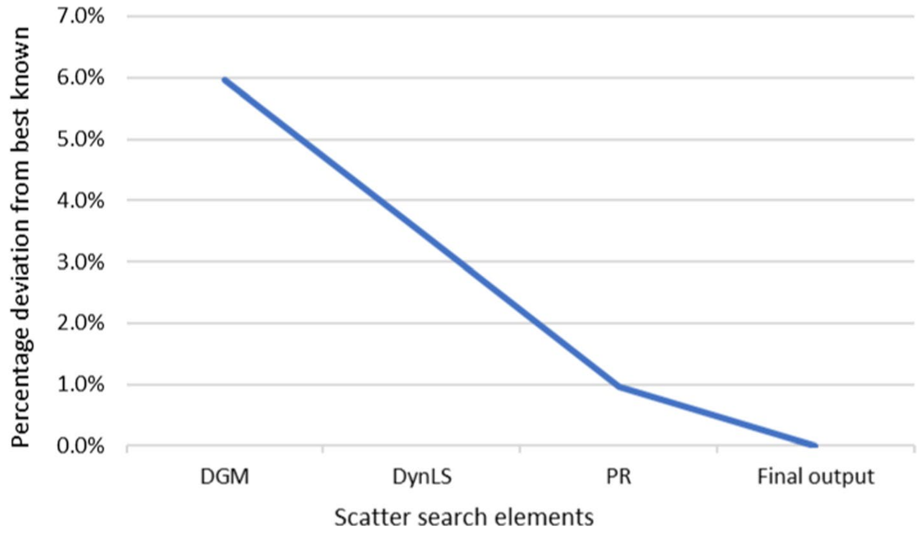


Fig. 7 Contribution of search elements

Table 5 Mathematical models

B factor	Set	n	CDP_Node Packing			CDP_standard		
			LB	UB	CPU	LB	UB	CPU
0.2	GKD	50	112.3	112.3	0.1	112.3	112.3	2.0
	GKD	150	118.7	118.7	0.6	118.7	118.7	669.9
	GKD	500	9.4	9.4	5.7	6.8	20.5	3600.2
	SOM	50	4.1	4.1	0.0	4.1	4.1	1.3
	MDG	500	0.0	125.0	3643.5	11.5	973.8	3600.2
0.3	GKD	50	97.8	97.8	0.0	97.8	97.8	2.5
	GKD	150	108.1	108.1	0.3	108.1	108.2	1519.5
	GKD	500	8.4	8.4	1.4	4.9	22.8	3600.1
	SOM	50	2.1	2.1	0.0	2.1	2.1	1.0
	MDG	500	3.1	60.2	3650.9	0.9	992.0	3600.1

most important difference in the performance of both models can be found in the large GKD sets, for which the previous formulation cannot solve any of its instances to optimality in 3600 s and the new formulations is able to solve all of them in few seconds.

The conclusion of this first experiment is that the GKD instances are easy to solve while the MDG are the most challenging for the CDP. This is very interesting since the recent study by Parreño et al. [24] concludes that the MDG are the most representative set of instances in diversity problems, and they recommend to use this set when comparing algorithms. We will use the entire benchmark to compare heuristics for the sake of completeness, although from a practical point of view, heuristics are needed to target MDG instances.

6.4 Competitive Testing of Heuristics

In this final experiment we employ the entire set of 100 instances to compare the four heuristics for this problem. Specifically, we compare T1 [27], SO [25], and our two methods, SS1(b = 5) and SS2 (b = 15). Since we obtained the optimal solutions in most of the instances (with the exception of the MDG set with $n = 500$), we report now the average percentage deviation with respect to the best known solution, DevB, and with respect to the optimal solution, DevO, when it is available. Note that DevB and DevO are not directly comparable because they are computed over different sets. Table 6 summarizes the results of this experiment over the entire benchmark of instances, and Table 7 reports the results on each set of 10 instances.

Table 6 shows that our two new methods, SS1 and SS2, outperform the previous methods, T1 and SO. Even our fast version SS1, is able to improve upon both previous methods, and just using a small fraction of their times. In particular, SS1 employs 1.55 s on average, while the two previous heuristics employ close to 20 s. In that time, SS1 is able to obtain solutions on a 1.6% deviation from the best known, while T1 and SO obtain solutions on a 21.5% and 13.4%

Table 6 Comparison of heuristics methods

Method	Value	DevB	DevO	Time
T1	47.62	21.5%	16.7%	19.16
SO	49.964	13.4%	9.6%	17.07
SS1	52.25	1.6%	1.8%	1.55
SS2	**52.76**	**0.1%**	**1.4%**	2.30

deviation respectively. On the other hand, our longer version SS2, as compared with SS1, exhibits remarkable results, with a 0.1% deviation with respect to the best known results. As a matter of fact, this method obtains the best known solution in 96 out of the 100 instances in our study, and still employs on average significant lower times than the previous heuristics. Regarding the optimal solutions known, SS2 also obtains a small deviation (1.4% on average).

Table 7 reports the same results than Table 6, but here disaggregated by type and size of instance. The first three columns specify it: capacity factor, name, and size, respectively. The first two rows in Table 7 depict the name of the heuristic and the statistics reflected. This table has 10 rows of results, each one reporting the statistics of the four heuristics on each set with 10 instances. Additionally, the bottom row collects the overall average results depicted in Table 6. In bold font we highlight the best average values obtained. They clearly show the poor performance of T1 and, on the other hand, the remarkable results obtained with SS2. It must be mentioned however, that T1 has a bounded performance guarantee of 2 (i.e., it is an approximate heuristic), which theoretically gives this method an added value,although in practice it does not perform very well.

If we compare the performance of the heuristics on the different sets of instances, we can find important differences. As expected, small instances (with 50 elements) are relatively easy to solve with all the heuristic methods (with the exception of T1). On the contrary, large instances (with 500 elements) are difficult to solve by heuristics, and it is

Table 7 Comparison of heuristic methods over each set of instances

			T1				SO				SS1				SS2			
			Value	DevB	DevO	Cpu	Value	DevB	DevO	Cpu	Value	DevB	DevO	Cpu	Value	DevB	DevO	Cpu
0.2	GKD	50	105.69	6.4%	6.4%	0.01	**112.33**	0.0%	0.0%	0.01	111.72	1.1%	1.1%	0.02	111.72	1.1%	1.1%	0.01
		150	113.24	4.0%	4.9%	0.10	115.15	2.2%	3.2%	0.40	117.46	0.1%	1.1%	0.11	**117.62**	0.0%	1.0%	0.13
		500	6.0	35.0%	36.6%	47.00	7.5	18.6%	20.6%	18.50	**9.2**	0.0%	2.4%	2.87	**9.2**	0.0%	2.4%	3.60
	SOM	50	3.6	12.0%	12.0%	0.00	**4.1**	0.0%	0.0%	0.00	**4.1**	0.0%	0.0%	0.00	**4.1**	0.0%	0.0%	0.00
	MDG	500	35.4	26.4%	–	48.60	41.2	14.5%	–	64.70	43.5	9.6%	–	2.37	**48.1**	0.0%	–	5.21
0.3	GKD	50	92.5	5.6%	5.6%	0.00	96.9	0.7%	0.8%	0.10	**97.6**	0.2%	0.3%	0.02	**97.6**	0.1%	0.2%	0.01
		150	104.6	2.5%	3.5%	0.20	103.4	3.5%	4.4%	0.80	**107.0**	0.0%	1.0%	0.15	106.9	0.2%	1.2%	0.19
		500	5.1	38.2%	39.4%	47.10	6.5	20.8%	22.5%	40.90	**8.2**	0.0%	2.0%	5.10	**8.2**	0.0%	2.0%	5.80
	SOM	50	1.5	23.3%	25.0%	0.00	1.5	23.3%	25.0%	0.00	1.9	3.3%	6.7%	0.00	**2.0**	0.0%	3.3%	0.01
	MDG	500	8.63	61.1%	–	48.60	11.09	50.2%	–	45.30	21.8	1.8%	–	4.89	**22.21**	0.0%	–	8.03
			47.62	21.5%	16.7%	19.16	49.964	13.4%	9.6%	17.07	52.247	1.6%	1.8%	1.55	**52.761**	0.1%	1.4%	2.30

Running times in Table 7 represented as 0.00 mean lower than 0.001

there where our proposals, SS1 and SS2, emerge as the clear winners. Specifically, if we consider the set 0.3-GKD-500, we can see an average deviation (DevB) of 38.2%, 20.8%, 0.00%, and 0.0% for T1, SO, SS1, and SS2 respectively. Similarly, in the set 0.3-MDG-500 these respective deviations are 61.1% (T1), 50.2% (SO), 1.8% (SS1), and 0.0% (SS2). Note that these are the instances in which we need to apply a heuristic, because in the small and medium ones our mathematical model is able to obtain the optimal solutions in moderate running times.

We applied the non-parametric Friedman test for multiple correlated samples to the solutions obtained by each of the four methods. This test computes, for each instance, the rank value of each method according to solution quality (where rank 4 is assigned to the best method and rank 1 to the worst one). Then, it calculates the average rank values of each method across all the instances solved. If the averages differ greatly, the associated p-value or significance will be small. The resulting p-value < 0.00001 obtained in this experiment clearly indicates that there are statistically significant differences among the four methods tested. Additionally, we apply the Sign test for a pairwise comparison between the best previous heuristic, SO, and our fast method, SS1. This test computes the number of instances on which an algorithm supersedes another one. The resulting p-value = 0.00004 confirms that SS1 outperforms SO (a similar result is obtained when comparing SO and SS2).

To easily visualize the comparative results in Table 7, we represent in Fig. 8 the average percentage deviation of each of the four heuristics, T1, SO, SS1, and SS2, in each of the 10 instance sets. These sets are represented in the x-axis. For example, the first one 0.2.GKD.50 collects the 10 instances in the GKD set with size $n = 50$ and capacity coefficient equal to 0.2. This diagram shows that SS1 and SS2 consistently outperform the previous methods in all the sets, with the minor exception of the first set, in which SO obtains the best results.

7 Conclusions

We had a twofold goal for this work, to experiment with the hybridization of PR and SS and, in the process, to develop a state-of-the-art memetic algorithm for the capacitated dispersion problem (CDP). We believe that we have achieved our goals. The design of our method is simple but effective, as shown in the computational experience comparing it with the previous methods.

In this paper, we also propose an improved mathematical programming model for the CDP. The model is based on a related problem and, although it has to be solved iteratively to obtain a solution of our problem, it turns out to present a remarkable performance in practice, obtaining the optimal solutions of both small and medium size instances.

The comparison between the best heuristic solutions obtained so far and the optimal values reveals that there is still a small room for improvement in the metaheuristic arena. We hope that our study triggers the interest of other researchers to follow this line of work on this interesting and challenging problem.

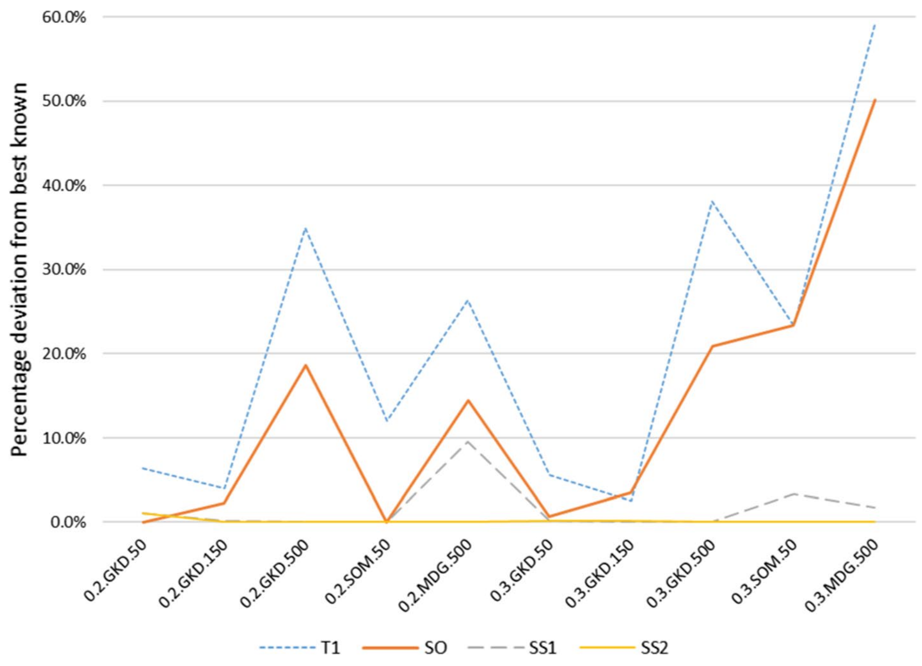


Fig. 8 Deviation of heuristics in each instance set

Acknowledgements This research has been partially supported by the Spanish Ministry with grant ref. PGC2018-0953322-B-C21/MCIU/AEI/FEDER-UE and PGC2018-095322-B-C22, "Comunidad de Madrid" and "Fondos Estructurales" of European Union with grant refs. S2018/TCS-4566, Y2018/EMT-5062.

References

1. Cotta C, Mathieson L, Moscato P (2018) Memetic Algorithms, In: Handbook of Heuristics (Martí, Pardalos, and Resende eds.), pp 607–638, Springer, Heidelberg
2. Duarte A, Martí R (2007) Tabu search and GRASP for the MDP. Eur J Oper Res 178:71–84
3. Duarte A, Sánchez-Oro J, Resende M, Glover F, Martí R (2015) GRASP with exterior path relinking for differential dispersion minimization. Inf Sci 296:46–60
4. Feo T, Resende MGC (1995) Greedy randomized adaptive search procedures. J Global Optim 6:109–133
5. Gallego M, Duarte A, Laguna M, Martí R (2009) Hybrid heuristics for the maximum diversity problem. Comput Optim Appl 44(3):411–426
6. Ghosh JB (1996) Computational aspects of the maximum diversity problem. Operations Research Letters 19:175–181
7. Glover F (1977) Heuristics for integer programming using surrogate constraints. Decis Sci 8:156–166
8. Glover F, Laguna M (1997) Tabu Search. Kluwer, Norwell, MA
9. Glover F (1998) A template for scatter search and path relinking. In: Hao J-K, Lutton E, Ronald E, Schoenauer M, Snyers D (eds) Artificial evolution. Lecture Notes in Computer Science, vol 1363. Springer, Berlin, pp 13–54
10. Glover F, Kuo CC, Dhir KS (1995) A discrete optimization model for preserving biological diversity. Appl Math Model 19:696–701
11. Glover F, Kuo CC, Dhir KS (1998) Heuristic algorithms for the maximum diversity problem. J Inf Optim Sci 19(1):109–132
12. Kuo CC, Glover F, Dhir KS (1993) Analyzing and modeling the maximum diversity problem by zero-one programming. Decis Sci 24:1171–1185
13. Laguna M, Martí R (2003) Scatter search: methodology and implementations in C. Kluwer Academic Publishers, Boston
14. Martí R, Duarte A (2010) The MDPLIB at Optsicom, http://grafo.etsii.urjc.es/optsicom/
15. Martí R, Gallego M, Duarte A, Pardo E (2013) Heuristics and Metaheuristics for the maximum diversity problem. J Heuristics 19(4):591–615
16. Martí R, Gallego M, Duarte A (2010) A branch and bound algorithm for the maximum diversity problem. Eur J Oper Res 200(1):36–44
17. Martí R, Laguna M, Campos V (2005) Scatter search versus genetic algorithms: an experimental evaluation with permutation problems. In: Rego C, Alidaee B (eds) Metaheuristic optimization via adaptive memory and evolution: tabu search and scatter search. Kluwer Academic Publishers, Norwell, MA, pp 263–282
18. Martí R, Laguna M, Glover F (2006) Principles of scatter search. Eur J Oper Res 169:359–372
19. Martínez-Gavara A, Campos V, Laguna M, Martí R (2017) Heuristic solution approaches for the maximum minsum dispersion problem. J Glob Optim 67(3):671–686
20. Neri F, Cotta C (2012) Memetic algorithms and memetic computing optimization: a literature review. Swarm and evolutionary computation, vol 2. Elsevier, Amsterdam, pp 1–14
21. Neri F, Cotta C (2012) A primer on memetic algorithms, handbook of memetic algorithms, chapter 4. In: Neri F, Cotta C, Moscato P (eds) Studies in computational intelligence, vol 379. Springer, Berlin, pp 43–54
22. Neri F (2012) Diversity management in memetic algorithms. In: Neri F, Cotta C, Moscato P (eds) Handbook of memetic algorithms, studies in computational intelligence, vol 379. Springer, Berlin, pp 153–165
23. Palubeckis G (2007) Iterated tabu search for the maximum diversity problem. Appl Math Comput 189:371–383

24. Parreño F, Álvarez-Valdés R, Martí R (2021) Measuring diversity. A review and an empirical analysis. Eur. J. Oper. Res. 289:515–532
25. Peiró J, Jiménez I, Laguardia J, Martí R (2021) Heuristics for the capacitated dispersion problem. International transactions in operational research 28:119–141
26. Resende MG, Martí C, Gallego M, Duarte A (2010) GRASP and path relinking for the max–min diversity problem. Comput Oper Res 37(3):498–508
27. Rosenkrantz DJ, Tayi GK, Ravi SS (2000) Facility dispersion problems under capacity and cost constraints. J Comb Optim 4:7–33
28. Sayyady F, Fathi Y (2016) An integer programming approach for solving the p-dispersion problem. Eur J Oper Res 253:216–225
29. Tirronen V, Neri F (2009) Differential evolution with fitness diversity self-adaptation. In: Chiong R (ed) Nature-inspired algorithms for optimisation, studies in computational intelligence, vol 193. Springer, Berlin, pp 199–234
30. Wang Y, Hao J-K, Glover F, Lü Z (2014) A tabu search based memetic algorithm for the maximum diversity problem. Eng Appl Artif Intell 27:103–114