Information Sciences 296 (2015) 46–60

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Greedy randomized adaptive search procedure with exterior path relinking for differential dispersion minimization ☆

Abraham Duarte [a,*], Jesús Sánchez-Oro [a], Mauricio G.C. Resende [b], Fred Glover [c], Rafael Martí [d]

[a] Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Spain
[b] Network Evolution Research Department, AT&T Labs Research, 200 S. Laurel Avenue, Room A5-1F34, Middletown, NJ 07748, USA
[c] OptTek Systems, Inc., Boulder, CO, USA
[d] Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain

ARTICLE INFO

Article history:
Received 3 April 2014
Received in revised form 5 September 2014
Accepted 6 October 2014
Available online 30 October 2014

Keywords:
Dispersion problems
GRASP
Variable neighborhood search
Path relinking
Path separation

ABSTRACT

We propose several new hybrid heuristics for the differential dispersion problem, the best of which consists of a GRASP with sampled greedy construction with variable neighborhood search for local improvement. The heuristic maintains an elite set of high-quality solutions throughout the search. After a fixed number of GRASP iterations, exterior path relinking is applied between all pairs of elite set solutions and the best solution found is returned. Exterior path relinking, or *path separation*, a variant of the more common interior path relinking, is first applied in this paper. In interior path relinking, paths in the neighborhood solution space connecting good solutions are explored between these solutions in the search for improvements. Exterior path relinking, as opposed to exploring paths between pairs of solutions, explores paths beyond those solutions. This is accomplished by considering an initiating solution and a guiding solution and introducing in the initiating solution attributes not present in the guiding solution. To complete the process, the roles of initiating and guiding solutions are exchanged. Extensive computational experiments on 190 instances from the literature demonstrate the competitiveness of this algorithm.

1. Introduction

Equity problems are a family of NP-hard optimization problems that are an actual concern in the context of facility location, where the fairness among candidate facility locations is as relevant as the dispersion of the selected locations [36]. Given a set of elements, the main objective of these problems is to find a subset of those elements that minimizes a similarity measure. These kinds of problems have also applications in the context of urban public facility location [36], selection of homogeneous groups [3], dense/regular subgraph identification [20], and equity-based measures in network flow problems [4]. In spite of all these applications, most of the previous works are focused on the opposite family of problems, the dispersion problems, in which the objective is to maximize the differences among the selected elements. The dispersion problems have been widely studied. See Duarte and Martí [7] for a recent review of this problem. However, we have identified only one

☆ AT&T Labs Research Technical Report.
* Corresponding author.
E-mail addresses: abraham.duarte@urjc.es (A. Duarte), jesus.sanchezoro@urjc.es (J. Sánchez-Oro), mgcr@research.att.com (M.G.C. Resende), glover@opttek.com (F. Glover), rafael.marti@uv.es (R. Martí).

http://dx.doi.org/10.1016/j.ins.2014.10.010
0020-0255/© 2014 Elsevier Inc. All rights reserved.

previous metaheuristic-based paper on equitable problems, in which Prokopyev et al. [29] adapt a simple generic GRASP algorithm to solve several equitable problems.

Let $G = (V, E)$ be an undirected complete graph, where V is the set of n vertices and E the set of $\binom{n}{2}$ edges. Each edge $(u, v) \in E$ with $u, v \in V$ has an associated distance d_{uv} between u and v. Dispersion, or diversity, problems (DP) consist in finding a subset $S \subseteq V$ with m elements, such that an objective function (based on the distances between elements in S) is maximized or minimized. According to Prokopyev et al. [29], the objective of a dispersion problem can be either to identify a subset with (i) maximum distance among its elements (diversity problems), or (ii) with maximum similarity among them (equity problems). The first class of problems has been intensively studied in the last ten years. For instance, Martí et al. [24,25] and Gallego et al. [12] present several exact, heuristic, and metaheuristic-based methods for the maximum diversity problem. Two important variants are, respectively, the sum (Maxsum DP) and minimum (Maxmin DP) of the distances in the selected set [1].

Prokopyev et al. [29] propose four distinct equity-based functions to balance the diversity among the selected elements: the mean-dispersion function minimizes the average dispersion of the selected elements; the generalized mean-dispersion function, which is an extension of the mean-dispersion function, considers vertex-weighted graphs; and the min-sum and the min-diff dispersion functions that consider the extreme equity values of the selected elements. In this paper we focus on the last function, whose associated optimization problem is referred to as the *Minimum Differential Dispersion Problem* (Min-Diff DP). The Min-Diff DP is strongly NP-hard, and it remains NP-hard even if sign restrictions for distances between vertices are imposed [29]. Therefore, heuristic procedures emerges as the best option to obtain high quality solutions in shorter computing time.

A feasible solution of the Min-Diff problem is a set $S \subseteq V$ of m elements, where m is a given input parameter. Each feasible solution has associated with it a cost which can be computed as follows. Let $\Delta(v)$ be the sum of distances between a vertex $v \in S$ and the remaining elements of S. Formally,

$$\Delta(v) = \sum_{u \in S} d_{uv}.$$

The objective function of a solution S, denoted by $diff(S)$, is then computed as

$$diff(S) = \max_{u \in S} \Delta(u) - \min_{v \in S} \Delta(v).$$

Therefore, the Min-Diff problem consists of finding a solution $S^\star \subseteq V$ with the minimum differential dispersion, i.e.

$$S^\star = \arg\min_{S \subseteq V_m} diff(S),$$

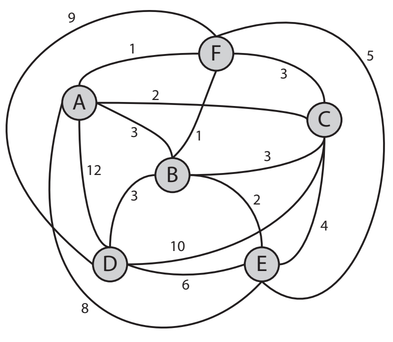
where V_m is the set of all subsets of vertices in V with cardinality m.

Fig. 1a shows an example of a graph with six vertices and 15 edges with their associated distances. Fig. 1b and c depict two possible solutions for the Min-Diff problem for $m = 4$. The selected vertices in the solution are shown in black while the edges in each solution are highlighted by solid lines. The vertices not in the solution are shown in gray while the edges not in the solution are dashed. To evaluate the quality of each solution, we first compute the $\Delta(v)$ value for all the elements in the solution. In particular, Fig. 1b shows a solution where $S = \{A, B, D, E\}$, $\Delta(A) = 3 + 12 + 8 = 23$, $\Delta(B) = 3 + 3 + 2 = 8$, $\Delta(D) = 12 + 3 + 6 = 21$, and $\Delta(E) = 8 + 2 + 6 = 16$. The *diff*-value is calculated by first selecting the vertices having the highest and lowest Δ-values and then taking the difference of their Δ-values. In this solution, these vertices are, respectively, A and B, and therefore $diff(S) = \Delta(A) - \Delta(B) = 23 - 8 = 15$. If we now consider the solution $S' = \{A, C, E, F\}$ in Fig. 1c, it is easy to verify that the associated objective function value is $diff(S') = 8$. Considering that the Min-Diff problem is a minimization problem, solution S' is better than solution S. The rationale behind this is that the distances among the elements in S' are more similar than those among the elements in S.

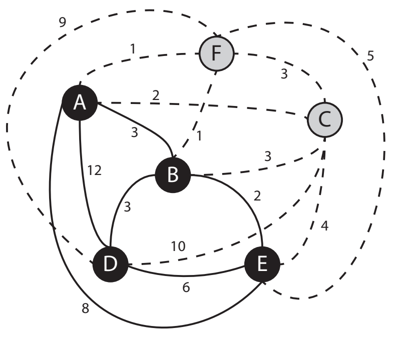
Prokopyev et al. [29] present a basic mixed linear $0-1$ formulation of the problem. Let L_i and U_i be lower and upper bounds on the value of $\sum_{j \in S} d_{ij}$, i.e. $L_i = \sum_{j \in S} \min\{d_{ij}, 0\}$ and $U_i = \sum_{j \in S} \max\{d_{ij}, 0\}$. Then, the mixed linear $0-1$ formulation of the Min-Diff DP is as follows:

$$\begin{aligned}
\min_{t,r,s,x} \quad & t \\
s.t. \quad & t \geqslant r - s, \quad i = 1, \ldots, n \\
& r \geqslant \sum_{j:j \neq i} d_{ij} x_j - U_i(1 - x_i) + M^-(1 - x_i), \quad i = 1, \ldots, n \\
& s \leqslant \sum_{j:j \neq i} d_{ij} x_j - L_i(1 - x_i) + M^+(1 - x_i), \quad i = 1, \ldots, n \\
& \sum_{i=1}^{n} x_i = m \quad x \in \{0, 1\}^n,
\end{aligned}$$

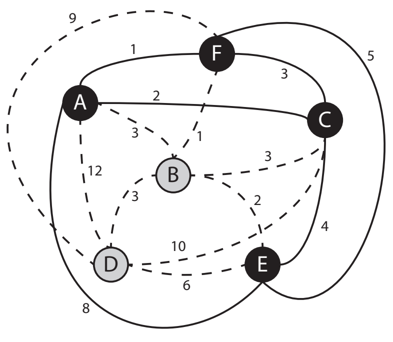
where M^+ is an upper bound on the U_i values, M^- is a lower bound on the L_i values, and the binary decision variable $x_i = 1$ if and only if node $i \in S$.



(a) Example of a graph



(b) Solution S



(c) Solution S'

Fig. 1. Example of two solutions on a graph with six vertices.

The computational experiments performed in Prokopyev et al. [29] show that a commercial solver using the mathematical formulation of the problem is only able to solve instances of small size (up to $|V| = 40$ and $m = 15$), requiring high CPU times (more than 2500 s on average). In particular, the authors use IBM ILOG CPLEX Optimizer, version 9.0 (usually known as CPLEX 9.0), which solves integer programming problems, very large linear programming problems (using either primal or dual variants of the simplex method or the barrier interior point method), convex and non-convex quadratic programming problems, and convex quadratically constrained problems (solved via second-order cone programming). We refer the reader to the original documentation[1] for a deeper description of this software. The authors also propose a generic GRASP that can be applied to different equity problems. Specifically, the method starts by constructing a solution from scratch. In each iteration, a new element is added to the solution under construction. In particular, the method forms a list of candidate elements to be added to the solution, ordering them according to its contribution to the objective function value. The method then selects the corresponding element in a probabilistic way among the top-evaluated ones (the lower the contribution the higher probability to be added). The authors also present a local search strategy based on random interchanges between vertex already in the solution and a vertex that is not in it. Only improving moves are accepted (i.e., those that reduce the value of the objective function). The local search ends when no improvement has been found after performing N interchanges, where N is a parameter of the algorithm. This two phases are repeated until a fixed number of iterations is reached. The objective of our paper is to propose a specialized GRASP that obtains high-quality solutions for the Min-Diff problem without requiring long running times. Additionally, we hybridize GRASP with path relinking for improved outcomes. Specifically, we propose in Section 2 two constructive procedures and three local search methods. We also introduce in Section 2.3 an improvement strategy based on the variable neighborhood search metaheuristic. Finally, in Section 3, we consider two post-processing strategies based on path relinking. It is worthwhile mentioning that we apply a new variant of path relinking, introduced in Glover [14] and called *exterior path relinking*, or *path separation*, and which, for this problem, obtains very promising results. In Section 4, we present computational experience. We first analyze and tune the proposed algorithms and then compare our best proposal with both the GRASP of Prokopyev et al. [29] and CPLEX 12.5.1 on the integer programming formulation proposed there and described above. Concluding remarks are outlined in Section 5.

2. GRASP

The greedy randomized adaptive search procedure (GRASP) is a metaheuristic developed in the late 1980s [10] and formally introduced in Feo et al. [11]. Resende and Ribeiro [31,32] present recent and thorough surveys of this method. GRASP is

[1] http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

a multi-start methodology where each iteration consists of two stages. The first is a greedy, randomized, and adaptive construction of a solution. The second stage applies an improvement method to obtain a local optimum from the constructed solution. These two phases are repeated until a termination criterion is met. The rest of this section is organized as follows. Section 2.1 presents two constructive procedures for the Min-Diff problem. Section 2.2 introduces three local search algorithms whose objective is to improve the constructed solution. Finally, Section 2.3 describes a more elaborated improvement strategy based on the variable neighborhood search (VNS) metaheuristic [26].

2.1. Constructive methods

GRASP constructive procedures apply a greedy function to evaluate the quality of the elements in a candidate list. Given a partial solution S, we propose the following greedy function to estimate the increment/decrement of the objective function when an element $v \in V \setminus S$ is added to S. Given the complexity of the objective function evaluation in the Min-Diff problem, the definition of such a greedy function is not trivial. For the sake of simplicity, the evaluation of the greedy function consists of four steps. The first step estimates the Δ-value of vertex u, denoted by $\delta(u)$, if it is included in the partial solution:

$$\forall u \in V \setminus S \rightarrow \delta(u) = \sum_{v \in S} d_{uv}.$$

The second step estimates the variation in the Δ-values of all vertices $v \in S$ if u is included in S:

$$\forall v \in S \rightarrow \delta(v) = \Delta(v) + d_{uv}.$$

Once these δ values are computed, the third step determines whether the potential inclusion of vertex $u \in V \setminus S$ in the partial solution modifies the maximum and/or the minimum δ-values. This values are, respectively, denoted as

$$\delta_{\max}(u) = \max\left\{\delta(u), \max_{v \in S} \delta(v)\right\},$$

and

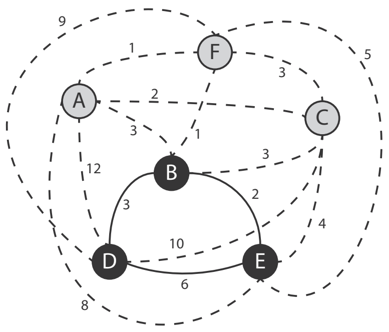
$$\delta_{\min}(u) = \min\left\{\delta(u), \min_{v \in S} \delta(v)\right\}.$$

The fourth step finally computes the greedy function g for each element $u \in V \setminus S$ as

$$g(u) = \delta_{\max}(u) - \delta_{\min}(u).$$

Let us illustrate the computation of the greedy function with an example. Fig. 2a shows a partial solution $S = \{B, D, E\}$, where vertices in S are highlighted in black and the vertices in $V \setminus S$ are shown in gray. If $m = 4$, we must include one vertex from $V \setminus S$ in the current partial solution. Fig. 2b shows the evaluations of the candidate vertices A, C, and F. For each candidate vertex, we compute its δ-value, as well as the δ-values for each vertex already in S. For example, if we introduce vertex A in the current partial solution, then $\delta(A) = d_{AB} + d_{AD} + d_{AE} = 3 + 12 + 8 = 23$. In addition, the inclusion of A would affect vertices B, D, and E as follows: $\delta(B) = \Delta(B) + d_{AB} = 5 + 3 = 8; \delta(D) = \Delta(D) + d_{AD} = 9 + 12 = 21; \delta(E) = \Delta(E) + d_{AE} = 8 + 8 = 16$. Then, we identify the $\delta_{\min}$ and $\delta_{\max}$-values ($\delta(B) = 8$ and $\delta(A) = 23$, respectively) and finally the greedy function value for the potential inclusion of A in the partial solution is $g(A) = \delta_{\max} - \delta_{\min} = \delta(A) - \delta(B) = 23 - 8 = 15$. Fig. 2b shows that the best option is to include vertex C, with $g(C) = 11$, in the current partial solution since this insertion produces the minimum increment in the objective function ($g(A) = 15$ and $g(E) = 12$).

Algorithm 1 shows pseudo-code for C1, the first constructive algorithm. It follows the standard GRASP template, by initially creating a list of candidates (CL) which contains the elements that can be added to the partial solution under



(a) Partial solution.

v	A	C	F
$\delta(v)$	23	17	15
$\delta(B)$	8	8	6
$\delta(D)$	21	19	18
$\delta(E)$	16	12	13
$\delta_{\min}(v)$	8	8	6
$\delta_{\max}(v)$	23	19	18
$g(v)$	15	11	12

(b) Greedy function evaluation

Fig. 2. Example of computation of the greedy function.

construction. At this point, the CL contains all the vertices of the graph (step 2). Then, the method randomly selects the first vertex from CL (step 3) and includes it in the partial solution (step 4). The method thus iterates until it obtains a solution with m vertices (steps 6–13). In each iteration, C1 calculates the maximum ($g_{\max}$) and minimum ($g_{\min}$) values of the greedy function (steps 7–8). After that, C1 constructs a restricted candidate list (RCL) with all the candidates whose greedy value does not exceed a percentage α of the best greedy value (step 9). Finally, in the last step of the iteration the method selects at random one vertex from the RCL and adds it to the solution, updating CL (steps 10–12).

Algorithm 1. `C1`

```
1: $S \leftarrow \emptyset$
2: $\mathrm{CL} \leftarrow V$
3: $v_0 \leftarrow \mathtt{SelectRandom}(\mathrm{CL})$
4: $S \leftarrow S \cup \{v_0\}$
5: $\mathrm{CL} \leftarrow \mathrm{CL} \setminus \{v_0\}$
6: **while** $|S| < m$ **do**
7:    $g_{min} \leftarrow \min_{u \in \mathrm{CL}} g(u)$
8:    $g_{max} \leftarrow \max_{u \in \mathrm{CL}} g(u)$
9:    $\mathrm{RCL} \leftarrow \{v \in \mathrm{CL} | g(v) \leqslant g_{min} + \alpha \cdot (g_{max} - g_{min})\}$
10:   $u \leftarrow \mathtt{SelectRandom}(\mathrm{RCL})$
11:   $S \leftarrow S \cup \{u\}$
12:   $\mathrm{CL} \leftarrow \mathrm{CL} \setminus \{u\}$
13: **end while**
14: **return** $S$
```

We now consider C2, a second constructive procedure based on a different strategy introduced in Resende and Werneck [33]. Specifically, this alternative construction swaps the greedy and random stages of a standard GRASP construction. This construction template has been recently applied with success in other papers [5,30,28,8].

Algorithm 2 shows the pseudo-code of the proposed method whose first steps are similar to the ones of Algorithm 1. The differences between these constructive procedures are limited to the main loop (steps 6–11). In particular, C2 constructs the RCL by selecting $\alpha \times |\mathrm{CL}|$ elements from CL at random (step 7). Then, all the elements in the RCL are evaluated with the greedy function, selecting the one which presents the minimum greedy value (step 8). Finally, the solution and the associated candidate list are updated (steps 9 and 10). The method ends when the solution becomes feasible (i.e., $|S| = m$).

Algorithm 2. `C2`

```
1: $S \leftarrow \emptyset$
2: $\mathrm{CL} \leftarrow V$
3: $v_0 \leftarrow \mathtt{SelectRandom}(\mathrm{CL})$
4: $S \leftarrow S \cup \{v_0\}$
5: $\mathrm{CL} \leftarrow \mathrm{CL} \setminus \{v_0\}$
6: **while** $|S| < m$ **do**
7:    $\mathrm{RCL} \leftarrow \mathtt{ConstructRandom}(\mathrm{CL},\alpha)$
8:    $u \leftarrow \arg\min_{v \in \mathrm{RCL}} g(v)$
9:    $S \leftarrow S \cup \{u\}$
10:   $\mathrm{CL} \leftarrow \mathrm{CL} \setminus \{u\}$
11: **end while**
12: **return** $S$
```

The α parameter controls the greediness/randomness of the GRASP constructive procedures. Specifically, if $\alpha = 0$ the corresponding methods are purely greedy algorithms, while if $\alpha = 1$ they are totally random procedures. In Section 4 we investigate the influence of α.

2.2. Local search procedures

The second stage of a GRASP algorithm consists in improving the constructed solutions using a local search method, which will guide the search process to a local optimum. One of the key elements in designing an effective local search method is the definition of the move and the associated move value (change in the objective function value). In particular, for the Min-Diff problem we define $\texttt{move}(S, u, v)$ as the move that interchanges vertex $u \in S$ with vertex $v \in V \setminus S$. This move usually produces a variation in the objective function, denoted as $\texttt{move\_value}(S, u, v)$. As with the definition of the greedy function, the computation of this quantity is not trivial if we want to update the value of the objective function in an incremental way. Specifically, we need to identify the subset of edges of u (associated with the removed vertex) that no longer contribute to the objective function and the subset of edges of v (associated with the inserted vertex) which will be included in the computation of the objective function. Even without performing the move, we can estimate the Δ-values of the elements in S. We denote this estimate as δ (to be consistent with the notation introduced earlier). Therefore, if we were to remove vertex u and include vertex v in the solution S, the variation of the Δ-values would be computed as

$$\forall w \in S \setminus \{u\} \rightarrow \delta(w) = \Delta(w) - d_{wu} + d_{wv}.$$

We additionally must consider the estimation of including v in S, denoted as

$$\delta(v) = \sum_{w \in S \setminus \{u\}} d_{vw}.$$

The estimation of the objective function value if we would perform the move is computed as

$$\delta_{\min} = \min\left\{\delta(v), \min_{w \in S \setminus \{u\}} \delta(w)\right\},$$
$$\delta_{\max} = \max\left\{\delta(v), \max_{w \in S \setminus \{u\}} \delta(w)\right\}.$$

and

$$MinDiff(S \setminus \{u\} \cup \{v\}) = \delta_{\max} - \delta_{\min}.$$

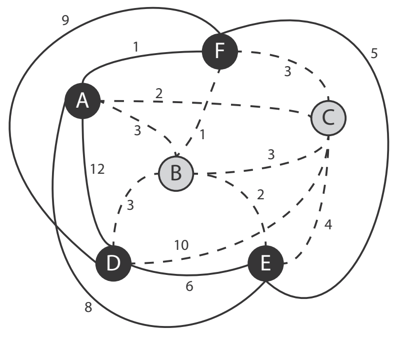
Therefore, the move value would be finally defined as

$$\texttt{move\_value}(S, u, v) = MinDiff(S \setminus \{u\} \cup \{v\}) - MinDiff(S).$$

This way, we can quickly compute the value of the move without computing the value of the objective function from scratch. In fact, we do not really perform the move to estimate the increment/decrement of the objective function. Starting from the solution depicted in Fig. 1b, we show in Fig. 3a the resulting solution after performing $\texttt{move}(S, B, F)$. Fig. 3b shows a table with the computation of the δ-values defined above. Taking these values into account, the minimum and maximum values are respectively $\delta_{\min} = 15$ for F and $\delta_{\max} = 27$ for D, resulting in a potential solution with $MinDiff(S \setminus \{B\} \cup \{F\}) = \delta_{\max} - \delta_{\min} = 27 - 15 = 12$. This move is accepted since it improves the current solution ($\texttt{move\_value}(S, B, F) = 12 - 15 = -3$).

In a straightforward implementation, the complexity of computing the $MinDiff$-value is $O(m^2)$ because the method should compute, for each one of the m vertices in S, the distance to remaining $m - 1$ vertices in S. However, using the proposed updating strategy defined above, the complexity reduces to $O(m)$ since it is only necessary to explore the $m - 1$ vertices in S adjacent to the removed vertex and the $m - 1$ vertices in S adjacent to the included vertex.

In this paper, we propose three local search procedures, denoted by LS1, LS2, and LS3, based on the move defined above. These three methods mainly differ in how the vertices are scanned. LS1 follows a best improvement template, resulting in an exhaustive search. Specifically, the method explores the vertices in the current solution S and those in $V \setminus S$. Then, it selects the best move between a vertex in S and a vertex in $V \setminus S$ (evaluating the aforementioned $\texttt{move\_value}$). Finally, if the best move found improves the current solution, the move is made, updating the solution. The second local search method, denoted LS2, follows a first improvement template. The algorithm is similar to LS1, but instead of exploring all possible moves, it performs the first move that improves the current solution. Vertices in S and $V \setminus S$ are randomly explored to avoid



(a) Resulting solution after performing $\texttt{move}(S, B, F)$.

v	$\delta(v)$
A	$23 - 3 + 1 = 21$
D	$21 - 3 + 9 = 27$
E	$16 - 2 + 5 = 19$
F	$1 + 5 + 9 = 15$
$\delta_{\min}$	15
$\delta_{\max}$	27
$MinDiff$	12

(b) Estimation of the δ-values for each vertex in the solution.

Fig. 3. Example of the computation of $\texttt{move\_value}(S, B, F)$.

focusing on the same subset of vertices. The third local search, LS3, also performs a first improvement strategy but ordering the vertices before exploring them. In order to start exploring the most promising moves, LS3 scans the vertices in S in descending order according to their Δ-values, while the vertices in $V \setminus S$ are scanned in ascending order according to their δ-values. Then, LS3 traverses both S and $V \setminus S$ performing the first move which improves the value of the current solution. The three local search methods end when no improvement is found after exploring all possible moves, returning the best solution found.

2.3. Variable neighborhood search

Variable neighborhood search (VNS) is a metaheuristic proposed by Mladenović and Hansen [26] as a general framework to solve hard optimization problems. It is based on the idea of performing systematic changes of neighborhood structures within the search procedure. Heuristics based on this metaheuristic have been successfully applied to a large variety of optimization problems. See for instance Duarte et al. [9], Sánchez-Oro et al. [34], Duarte et al. [6], and Lozano et al. [23]. We refer the reader to Mladenović [18] for a recent survey of VNS.

In this paper, we propose the use of a *Basic VNS* variant with a *Jump Neighborhood Change* strategy [19] in place of the standard local search used in GRASP. Algorithm 3 shows the pseudo-code of the VNS. It has three input arguments: the initial solution (S), the maximum neighborhood to be explored ($k_{\max}$), and the jump magnitude (k_{step}). The initial solution is built with one of the constructive procedures described in Section 2.1. The best constructive procedure as well as the values of $k_{\max}$ and k_{step} will be experimentally determined in Section 4.

The algorithm mainly consists in executing three strategies: shake, local search, and neighborhood change. First, given a solution S, the shake method generates a new solution, S', in the k-th neighborhood of the current solution (step 3). In the context of the Min-Diff problem, it consists in performing k moves at random. Then, S' is improved using a local search method, producing a new improved solution S'' (step 4). We will experimentally determine the best local search among the three proposed in this paper.

Algorithm 3. `BasicVNS` (S, k_{step}, k_{max})

```
1: $k \leftarrow k_{step}$
2: repeat
3:    $S' \leftarrow$ Shake($S, k$)
4:    $S'' \leftarrow$ LocalSearch($S'$)
5:    NeighborhoodChange($S, S'', k$)
6: until $k = k_{max}$
7: return $S$
```

The `NeighborhoodChange` function typically employed in a VNS compares the new solution S'' with the incumbent solution S obtained in the k-th neighborhood. If an improvement is obtained, k is reset to its original value (usually $k = 1$) and the solution S is updated with S''. Otherwise, the next neighborhood is considered for a further exploration (usually $k = k + 1$) without updating S. In this paper, we investigate the effect on the search of the so-called *jump neighborhood search*, where the `NeighborhoodChange` function considers the parameter k_{step} to control the change of the neighborhood. Specifically, when the VNS method performs an improving move, it sets $k = k_{step}$ instead of $k = 1$. Similarly, in non-improving moves, it sets $k = k + k_{step}$ instead of $k = k + 1$. As customary in VNS, the search ends when k reaches or surpasses $k_{\max}$, returning the best solution found. Note that the jumping strategy of this method skips some neighborhoods in the perturbation, which performs well on this type of nested neighborhoods of the same type of moves.

3. Path relinking

Path relinking (PR) is a metaheuristic introduced in Glover [13] and Glover and Laguna [16], originally proposed as a methodology to integrate intensification and diversification strategies in the context of tabu search. This metaheuristic explores trajectories that connect high-quality solutions, generating intermediate solutions that can eventually be better than the high-quality solutions being connected. Laguna and Martí [21] adapted PR in the context of GRASP as a form of intensification. The PR algorithm operates on a set of solutions, called the *elite set* (ES), typically sorted from best (first solution in ES) to worst (last solution in ES). In this paper, we limit ourselves to consider only a quality criterion to populate the elite set. Therefore, the ES consists of the best b solutions generated with GRASP. This design is usually referred to as *static* [30], since we first apply GRASP to construct the elite set and then we apply PR to explore trajectories between all pairs of solutions in the ES.

Given two solutions in ES, S and S', the standard implementation of path relinking, which in this paper we call *Interior Path Relinking* (IPR), starts from the *initiating solution* S and gradually transforms it into the *guiding solution* S'. This transformation is accomplished by swapping out elements selected in S with elements in S', generating a set of *intermediate solutions*. The elements present in both solutions ($S \cap S'$) remain selected in solutions generated in the path between them. The set of elements in S and not in S' is $S \setminus S'$. Symmetrically, $S' \setminus S$ is the set of elements selected in S' and not selected in S. To obtain the first intermediate solution in this path, we remove a single element $u \in S \setminus S'$ and include a single element $v \in S' \setminus S$, thus obtaining $S_1 = S \setminus \{u\} \cup \{v\}$. Notice that S_1 can be trivially generated with the move described above. For the sake of simplicity, we denote this move as $S_1 = \texttt{move}(S, u, v)$. In general, the k + 1-th intermediate solution is constructed from the previous solution as $S_{k+1} = \texttt{move}(S_k, u, v)$ with $u \in S_k \setminus S'$ and $v \in S' \setminus S_k$.

Given a graph with 12 vertices (labeled $\{A, B, \ldots, L\}$) and $m = 6$, let $S = \{A, B, C, D, E, F\}$ and $S' = \{A, B, C, G, H, I\}$. Fig. 4 illustrates the construction of two interior paths, one from S to S' and another from S' to S. As it was aforementioned, common vertices between both solutions appear in all intermediate solutions. Solution S_1 is obtained from S by performing $\texttt{move}(S, D, G)$. Similarly, S_2 is obtained after applying $\texttt{move}(S_1, E, H)$. Notice that the reverse path is similarly constructed. In all cases, the introduced vertices are highlighted in gray.

The election of vertices u and v can be performed in a greedy or a random fashion. In particular, the greedy strategy obtains S_{k+1} from S_k by evaluating all the possibilities for $v \in S_k \setminus S'$ to be unselected and $u \in S' \setminus S_k$ to be selected, and performs the best move. On the other hand, the random strategy constructs S_{k+1} by randomly selecting a vertex $v \in S_k \setminus S'$ to be unselected and a vertex $u \in S' \setminus S_k$ to be selected. In this paper, we propose two interior path relinking methods: IPR$_G$ which constructs the paths between each pair of solutions in the ES using a greedy strategy, and IPR$_R$, which follows the random strategy. The best solution generated in each path is subjected to the improvement method described in Section 2.3. The algorithm terminates when all pairs of solutions in the ES have been relinked, each pair by two paths. The best overall solution is returned.

Despite the widespread application of path relinking in combinatorial optimization, almost all PR implementations only consider the between-form of PR (Interior Path Relinking). This paper discusses the beyond-form of path relinking, introduced in Glover [14] and called *Exterior Path Relinking* (EPR), and focuses on its relevance for effectively solving the MinDiff problem. Instead of introducing into the initiating solution characteristics present in the guiding solution, this new strategy introduces in the initiating solution characteristics not present in the guiding solution. Specifically, it removes from the initiating solution those elements which also belong to the guiding solution, obtaining intermediate solutions which are further away from both the initiating the guiding solutions.

The relevance of paths that go beyond the initiating and guiding solutions was broached in Glover [13] as follows: the scope of strategies made available by path relinking is significantly affected by the fact that the term neighborhood has a broader meaning in tabu search than it typically receives in the popular literature on search methods. Often, the neighborhood terminology refers solely to methods that progressively transform one solution into another. Such neighborhoods are called transition neighborhoods in tabu search, and are considered as merely one component of a collection of neighborhoods that also include those operating in regions beyond solutions previously visited.

Given the initiating (S) and guiding (S') solutions for the MinDiff problem, the first intermediate solution in the exterior path beyond S is generated by removing a single element $u \in S \cap S'$ and adding a single element $v \in V \setminus (S' \cup S)$, thus obtaining $S_1 = S \setminus \{u\} \cup \{v\}$. Again, this solution can be directly obtained with the move operator described in Section 2.2. The $k + 1$-th intermediate solution is constructed from S_k, the previous solution, as $S_{k+1} = \texttt{move}(S_k, u, v)$ with $u \in S_k \cap S'$ and $v \in V \setminus (S' \cup S_k)$. As for IPR, we propose two methods: EPR$_G$ and EPR$_R$. EPR$_G$ constructs the paths using a greedy strategy while EPR$_R$ follows the random strategy. Again, the best solution generated in each path is subjected to the improvement method described in Section 2.3. The algorithm terminates when all pairs of solutions in the ES have been relinked, each pair by two paths, one beyond S and the other beyond S'.

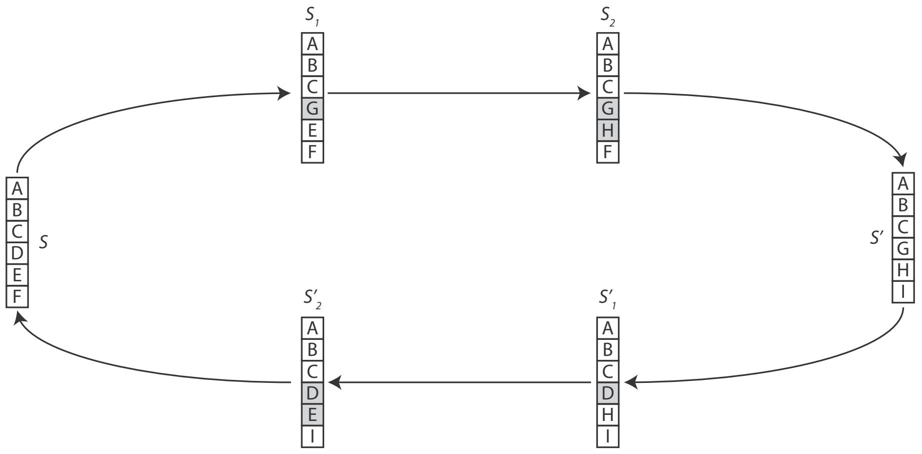


Fig. 4. Example of two interior paths between S and S'.

We illustrate in Fig. 5 the construction of the exterior paths by considering the same graph with 12 vertices labeled $\{A, B, \ldots, L\}$ introduced earlier, with $m = 6$ and the same initiating and guiding solutions. As it can be seen, the exterior path generates solutions S_1 and S_2 by performing $\texttt{move}(S, A, J)$ and $\texttt{move}(S_1, B, K)$, respectively. It is easy to see that those intermediate solutions (i.e., S_1 and S_2 are further from S' than S). The other exterior path (starting from S' and finishing in S'_2) is constructed in a similar way. As in the previous example, the introduced vertex is highlighted in gray.

4. Computational results

In this section, we report on the computational experiments performed to test the efficiency and effectiveness of the proposed strategies. All algorithms were implemented in Java 7 and the experiments were conducted on an Intel Core i7 2600 CPU (3.4 GHz) with 4 GB of RAM. We have used a comprehensive set of representative instances previously used for computational experiments in equity/diversity problems [25]. This benchmark is usually referred to as MDPLIB and it is publicly available at http://www.optsicom.es/mdp/mdplib_2010.zip. It is considered a standard for the comparison of equity/diversity methods, indeed more than 15 papers have used this benchmark in their computational experimentation. The MDPLIB is divided into three sets of instances:

- **SOM**: This data set consists of 20 inter-node distance matrices of sizes ranging from $n = 25$ and $m = 2$ to $n = 500$ and $m = 200$ and were collected by Duarte and Martí [7]. They were created with a generator developed by Silva et al. [35] and have been used in most of the previous papers dealing with the maximum diversity problem (see for example [2]).
- **GKD**: This data set consists of 70 inter-node distance matrices for which distance values were calculated as the Euclidean distance between pairs of randomly generated points with coordinates in the $[0, 10] \times [0, 10]$ square. The sizes of these instances range from $n = 10$ and $m = 2$ to $n = 500$ and $m = 50$. These instances were introduced in Glover et al. [15] and generated in Duarte and Martí [7] and Martí et al.[24].
- **MDG**: This data set consists of 100 inter-node distance matrices with real numbers randomly selected between 0 and 10 from a uniform distribution and size varying from $n = 500$ and $m = 50$ to $n = 3000$ and $m = 600$. These instances are extensively described in Duarte and Martí [7], Palubeckis [27], and Martí et al. [25].

The original MDPLIB contains 315 instances. We have excluded the smallest and/or easiest ones from our computational experience, since those instances are not really a challenge for modern heuristics. In particular, we have removed 75 instances of from GKD set (with size ranging from $n = 10$ to $n = 30$) and 50 instances of SOM (with size ranging from $n = 25$ to $n = 150$).

The experiment has two parts. In the first part, we adjust the parameters of the methods and select the best variants of the proposed algorithms on a subset of 25 (10 from SOM, 5 from MDG and 10 from GKD) instances from the MDPLIB. We have selected these instances from the three subsets with different values of $n = 25, \ldots, 500$ and $m = 15, \ldots, 200$, since these variations in the value of n, m and original subset allow us to construct a representative subset of the benchmark. The second part is devoted to a comparison of our best proposal with the current state of the art for this problem, including the solution of the mixed linear $0 - 1$ formulation of Prokopyev et al. [29] with the commercial MIP solver CPLEX 12.5.1. The number of iterations indicated in the paper determines the stopping criterion of all heuristic methods tested in this section. However, considering that we are solving large instances (up to 3000 vertices) we impose a maximum computing time of n seconds (n being the number of vertices) to avoid extremely large computing times. After that time limit, the heuristic algorithm is interrupted, returning the best solution found.

4.1. Algorithm configuration

The first experiment compares the two constructive methods described in Section 2.1. For GRASP, both the quality and variability of the constructed solutions are important for the success of local search. Ideally, we want to construct good solutions that are scattered about the solution space. For the Min-Diff problem we compute variability as the average number of steps in the neighborhood space among the constructed solutions. In other words, the variability between two solutions S and S' is defined as the cardinality of the set difference of the two solutions. Then, the variability of the set C of constructed solutions is defined as

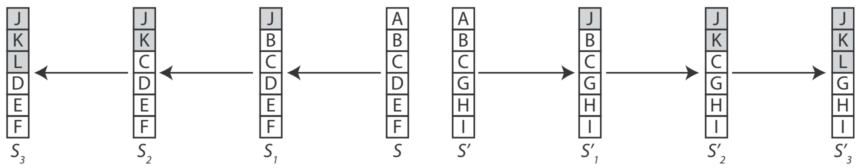


Fig. 5. Example of two exterior paths, one beyond S and the other beyond S'.

$$variability(C) = \frac{\sum_{S\in C}\sum_{S'\in C}|S \setminus S'|}{|C|}.$$

This experiment compares the quality and variability of the solutions produced by the constructive methods C1 and C2 by considering 100 independent constructions of each. The α parameter value is set to *random*, 0.25, 0.50, 0.75, where *random* indicates that the method randomly selects an α value in the range $[0,1]$ for each construction. Notice that the greater the value of α, the greater will be the expected variability of the constructed solutions. Fig. 6 shows the result of this comparison where the values of quality and variability have been normalized to fall between 0 and 1. This figure shows that C2(0.50) attains the largest quality but with relatively low variability. On the other hand, one of the most randomized methods, C1(0.75), produces poor-quality solutions, but having the largest variability among all methods. Finally, C2(0.25) shows a balance between quality and variability. Specifically, it presents slightly worse quality than the best method, but considerably larger variability.

In the experiments that follow, we limit ourselves to the constructive procedures identified above as having produced the best quality, C2(0.50), the best variability, C1(0.75), and a good tradeoff between quality and variability, C2(0.25).

We next study the efficiency of the three local search methods proposed in Section 2.2 when coupled with C2(0.50), C1(0.75), and C2(0.25), the three constructive procedure chosen above. Recall that these local search procedures are: LS1 – best improvement strategy; LS2 – first improvement with random selection; and LS3 – first improvement with ordered selection. We embed them in a GRASP algorithm, constructing and improving 100 solutions. We report, for each algorithm, the average objective function value, Avg., and the average CPU time in seconds, Time (s). In each experiment, we also compute for each instance the overall best solution value, *BestValue*, which is obtained by the executions of all methods considered in the corresponding experiment. Then, for each method, we compute the relative percentage deviation between the best solution value obtained with that method and *BestValue* for that instance. We then report the average of this relative percentage deviation, Dev(%), across all the instances considered in each particular experiment. We finally report, for each method, the number of instances (#Best) in which the value of the best solution obtained with this method matches *BestValue*. For the sake of clarity, we highlight in bold font the best combination of methods. Table 1 shows the results obtained by the different combinations of constructive and local search procedures.

In this experiment C2(0.50) coupled with LS2 emerges as the best GRASP variant. It obtains the smallest deviation (11.10%) and the largest number of best solutions (7 out of 25). It is important to remark that this method ranks second (very close to the fastest algorithm) when comparing CPU times of the nine variants tested. This experiment confirms that the compromise between quality and variability is crucial in a GRASP design. We therefore select C2(0.50) with LS2 as the best variant and use it in the remaining experiments.

Next, we analyze the effect of replacing the local search component of a GRASP by a VNS (see Section 2.3). Specifically, we compare the best algorithm identified above with nine variants of GRASP with VNS local search. These VNS procedures differ in the k_{step} and k_{max} parameters. We tested the values $k_{max} = \{0.1n, 0.2n, 0.3n\}$ and $k_{step} = \{0.01n, 0.025n, 0.05n\}$, where $n = |V|$, and denote the method by VNS(k_{step}, k_{max}). Table 2 shows that including a VNS in a GRASP procedure considerably improves the quality of the results. As a comparison of Tables 1 and 2 shows, the average objective function for constructive with local search varies from 162.65 to 184.11, while that average for constructive with VNS in place of local search varies from 136.56 to 155.11. However, as expected, the running time of the VNS grows with large values of k_{max} and/or with small values of k_{step}. To find a compromise between CPU time and quality, we select VNS(0.01, 0.1) for the next experiments, since it presents the best results in terms of average objective function (136.56) and number of best solutions found (9 out of 25).

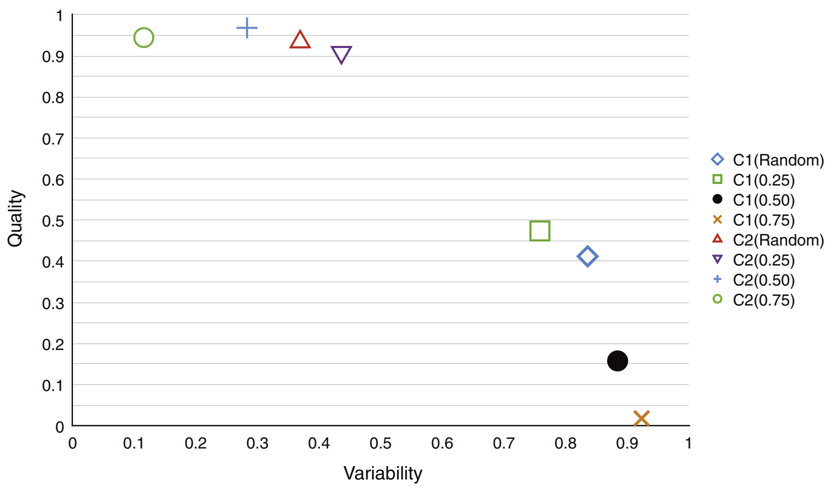


Fig. 6. Comparison of quality and variability of the constructive methods.

Table 1
Three local search methods coupled with three constructive procedures.

	Avg.	Time (s)	Dev. (%)	#Best
C1(0.75)				
LS1	163.51	77.09	17.52	6
LS2	162.65	16.95	12.18	6
LS3	166.87	88.30	16.06	2
C2(0.25)				
LS1	184.11	30.23	18.65	3
LS2	180.16	13.08	14.72	3
LS3	181.53	63.57	13.38	4
C2(0.50)				
LS1	172.21	30.24	14.74	3
LS2	**171.49**	**14.08**	**11.10**	**7**
LS3	173.23	60.59	12.28	4

Additionally, it ranks third (out of 9 methods) when considering the average deviation (8.80%) but taking about half the CPU time (156.64 s) of VNS(0.01, 0.2) and VNS(0.01, 0.3) whose computing times are 226.90 and 300.81 s, respectively.

To single out the contribution of the VNS we conduct an additional experiment. In particular, we compare the GRASP method in which construction is C2(0.50) and improvement is LS2 with the GRASP method in which LS2 is replaced by VNS, allowing both algorithms to run for 150 s, on average, which is the average running time taken by the best variant with VNS. The algorithm with VNS consistently produces better outcomes. Specifically, it obtains a lower average objective function value (136.48 versus 156.41), a lower average deviation (0.56% versus 6.89%) and a larger number of best solutions found (20 and 9).

We next compare the four path relinking algorithms described in Section 3 by incorporating them into the GRASP with VNS local search. As it is documented in Laguna and Martí [22], the size of the elite set must have a relatively small size (around 10). We have performed several preliminary experiments to test the impact of this parameter in the performance of the whole algorithm. These results show that augmenting the size of the elite set also increases the computing time, but barely improving the quality. Therefore, we select the best 10 solutions (among the solutions generated by GRASP with VNS) to conform the elite set. Then, each path relinking variant creates a path between each pair of solutions in the elite set. Table 3 compares the methods with interior path relinking with greedy (IPR_G) and random (IPR_R) construction of the path, as well as the exterior path relinking with the same two strategies of exploring the path ($\mathrm{EPR}_G, \mathrm{EPR}_R$). We can observe that the random and the greedy strategy obtain similar results in the case of Interior Path Relinking. Specifically, IPR_G obtains slightly better deviation, while IPR_R gets one more best solution. In the case of exterior path relinking, the greedy strategy consistently produces better outcomes. In fact, the greedy exploration of the path barely affects the computing time. This is true mainly because we use the move strategy described in Section 2.2. Another relevant observation is that the two exterior versions of PR clearly outperform the interior variants. With this, the GRASP with VNS using the EPR_G strategy emerges as the best algorithm for all statistics. We believe that this result could be an important lesson for future implementations of path relinking.

We conduct an additional experiment to single out the actual contribution of PR in final design of the algorithm. In particular we compare EPR_G with the best GRASP algorithm executed for the same computing time (about 150 s on average). EPR_G obtains a lower average deviation (0.71 % versus 5.69 %) and a larger number of best solutions found (23 versus 14).

In the remainder of the paper we refer to this GRASP with VNS and Greedy exterior path relinking algorithm simply as EPR.

Table 2
Influence of the k_{step} and k_{max} parameters in the VNS algorithm.

	k_{step}	Avg.	Time (s)	Dev. (%)	#Best
	0.01	**136.56**	**156.64**	**8.80**	**9**
0.1	0.025	149.09	93.22	15.82	5
	0.05	155.11	59.16	17.68	6
	0.01	143.20	226.90	8.20	5
0.2	0.025	142.02	141.40	10.16	7
	0.05	150.79	92.54	17.38	3
	0.01	141.54	300.81	7.01	8
0.3	0.025	143.74	172.98	12.38	4
	0.05	151.47	122.31	13.72	6

The best parameters configuration is highlighted in bold font.

Table 3
Comparison among the proposed path relinking algorithms.

	Avg.	Time (s)	Dev (%)	#Best
IPR_G	136.81	151.22	4.32	13
IPR_R	137.01	150.81	4.60	14
EPR_G	**136.23**	**156.05**	**0.80**	**20**
EPR_R	137.09	162.73	3.28	14

The best variants is highlighted in bold font.

4.2. Algorithm evaluation

In Section 4.1, we identified EPR as being our best proposed procedure. We next compare it with the current state-of-the-art methods for the Min-Diff problem. These methods are a GRASP proposed by Prokopyev et al. [29] and the commercial MIP solver CPLEX 12.5.1 on their exact formulation.

Our evaluation consists of two experiments. In the first, we compare EPR with two variants of the GRASP of Prokopyev et al. [29], one denoted as GRASP1 which runs for 500 iterations and the other, GRASP2, which runs for 1000 iterations. In the second experiment, we compare EPR with CPLEX on all instances that fit in memory.

For the first experiment, we have implemented all of the algorithms in Java. In order to have a fair comparison, both GRASP1 and GRASP2 use the partial evaluations described in Section 2.1. The experiments are run on the same computer. Table 4 summarizes the results of this experiment, where we consider the three sets of instances (GKD, MDG, and SOM) and the three algorithms (EPR, GRASP1, and GRASP2). The table is organized in three groups of rows (one for each type of instance). For each pair of instance type and algorithm, the table lists average solution value over all instances in the set, the average CPU time in seconds, the average percent deviation from the best known solution, and the number of times that the methods matches the best known solution. We also provide the average number of evaluations needed by each algorithm to reach their best result (#EvalsToBest) and the average number of evaluations performed (#TotalEvals).

Each algorithm is run a single time on each instance. With respect to solution quality, EPR clearly outperforms both GRASP1 and GRASP2. It should be noted that the GRASP proposed by Prokopyev et al. [29] was designed to work on a number of equitable dispersion problems and not specifically on the MinDiff problem as EPR is designed for.

EPR finds the best known solution in 188 of 190 instances, while GRASP1 and GRASP2 do so for only 12 and 18 instances, respectively. The instances for which GRASP1 and GRASP2 find the best known solution are all in the class GKD, which has

Table 4
EPR compared with the GRASP algorithms of Prokopyev et al. [29].

Instance set	EPR	GRASP1	GRASP2
GKD			
Avg.	52.57	107.82	90.19
Time (s)	56.99	76.03	152.07
Dev (%)	0.00	74.19	69.69
#Best	68	12	18
#EvalsToBest	$2.84 \cdot 10^5$	$2.01 \cdot 10^6$	$5.2 \cdot 10^6$
#TotalEvals	$2.84 \cdot 10^5$	$4.50 \cdot 10^6$	$9.1 \cdot 10^6$
MDG			
Avg.	3567.63	5981.61	5981.80
Time (s)	1472.35	1421.10	2793.23
Dev (%)	0.00	81.51	79.57
#Best	100	0	0
#EvalsToBest	$1.02 \cdot 10^6$	$5.83 \cdot 10^7$	$6.94 \cdot 10^7$
#TotalEvals	$1.68 \cdot 10^6$	$1.15 \cdot 10^8$	$1.40 \cdot 10^8$
SOM			
Avg.	23.35	37.90	37.25
Time (s)	173.41	124.40	198.82
Dev (%)	0.00	61.54	58.18
#Best	20	0	0
#EvalsToBest	$4.87 \cdot 10^5$	$6.69 \cdot 10^6$	$1.32 \cdot 10^7$
#TotalEvals	$5.73 \cdot 10^5$	$1.20 \cdot 10^7$	$2.40 \cdot 10^7$
ALL			
Avg.	1899.53	3200.00	3189.18
Time (s)	814.17	789.05	1547.07
Dev (%)	0.00	76.71	73.68
#Best	188	12	18
#EvalsToBest	$5.98 \cdot 10^5$	$2.23 \cdot 10^7$	$2.93 \cdot 10^7$
#TotalEvals	$8.47 \cdot 10^5$	$4.37 \cdot 10^7$	$5.97 \cdot 10^7$

the smallest instances as well as Euclidean distances, making them easier to solve as we will see later in this section in the experiments with CPLEX. In addition to not finding many best known solutions, both GRASP1 and GRASP2 find solutions that have a high percent deviation from the best known solutions, varying, on average, from 58% for GRASP2 on SOM to 81% for GRASP1 on MDG. As expected, running times for EPR are comparable to those of GRASP1, which, also as expected, are about one half of those of GRASP2. If we now analyze the number of evaluations needed by each algorithm to reach their best results and the total number of evaluations performed, we can see that EPR clearly requires less evaluations to reach the best solution, in all the set of instances. Specifically, EPR needs roughly 2.5% of the evaluations needed by GRASP1 and GRASP2 to reach the best result. Even more, the total number of evaluations performed by EPR is considerably lower than the evaluations needed by GRASP1 and GRASP2, with EPR requiring less than 2.0% of evaluations needed by GRASP1 and GRASP2.

We apply the Friedman test to the raw data obtained in the previous experiment. This test ranks each method for each instance in the data set. That is, for each instance, the method that performs the best is assigned the number 1, followed by the second best (assigned number 2), and finally the worst method receives the number 3. Then, an average ranking is calculated for each method. A small *p*-value associated with this test indicates that the averages are indeed significantly different. We obtain a *p*-value lower than 0.001 indicating significant difference among the methods. Additionally, the test provides the ranking in which the best method is EPR with an average ranking of 1.10, followed by GRASP2 (average ranking of 2.31), followed by GRASP1 (average ranking of 2.60).

Finally, we compare EPR with GRASP2 by considering two well-known non-parametric tests for pairwise comparisons: the Wilcoxon test and the Sign test. The former answers the question: Do the two samples (in our case, solutions obtained with EPR and GRASP2) represent two different populations? The resulting *p*-value lower than 0.001 indicates that the values compared come from different algorithms and there are significant differences between both methods. On the other hand, the Sign test computes the number of instances on which an algorithm supersedes another. The resulting *p*-value lower than 0.001 again indicates that there are significant differences between EPR and GRASP2, confirming the superiority of the method proposed in this paper.

We compare EPR with CPLEX 12.5.1 only on the 30 instances which fit in memory for CPLEX. Most of these instances are from the GKD set. Three are from SOM and none were from MDG (the smallest instance in MDG has 500 vertices). Running times for CPLEX are limited to 1800 s. We provide a different stopping criterion since CPLEX is an exact procedure that typically requires larger computing time (even in instances with moderate size) than heuristic procedures. EPR again do 100 GRASP iterations, followed by exterior path relinking between all pairs of the 10 elite set solutions. Table 5 summarizes these

Table 5
Comparison of EPR and CPLEX (limited to 1800 s).

Instance	EPR		CPLEX			
	MinDiff	Time (s)	UB	LB	Time (s)	Opt
GKD-b_10_n25_m7	**23.26523**	0.312	23.26523	23.26523	1.014	1
GKD-b_11_n50_m5	**1.9261**	0.187	1.9261	1.9261	8.814	1
GKD-b_12_n50_m5	2.12104	0.171	2.0513	2.0513	9.044	1
GKD-b_13_n50_m5	**2.36231**	0.187	2.36231	2.36231	6.668	1
GKD-b_14_n50_m5	**1.6632**	0.188	1.6632	1.6632	6.428	1
GKD-b_15_n50_m5	**2.85313**	0.187	2.85313	2.85313	8.3	1
GKD-b_16_n50_m15	**42.74578**	1.389	42.74578	42.74578	54.937	1
GKD-b_17_n50_m15	**48.10761**	1.608	48.10761	48.10761	36.853	1
GKD-b_18_n50_m15	**43.19609**	1.343	43.19609	43.19609	441.016	1
GKD-b_19_n50_m15	**46.41245**	1.358	46.41245	46.41245	347.622	1
GKD-b_1_n25_m2	**0**	0	0	0	0.012	1
GKD-b_20_n50_m15	**47.71511**	1.265	47.71511	47.71511	599.622	1
GKD-b_21_n100_m10	13.83202	1.171	12.30384	0	1330.461	0
GKD-b_24_n100_m10	8.64064	1.202	9.81926	0	1500.213	0
GKD-b_26_n100_m30	168.72959	9.439	176.86238	0	1104.597	0
GKD-b_27_n100_m30	127.09726	9.72	205.76481	0	1099.519	0
GKD-b_28_n100_m30	106.37919	10.422	148.59098	0	1212.293	0
GKD-b_29_n100_m30	137.45316	10.048	176.75614	0	1091.947	0
GKD-b_2_n25_m2	**0**	0	0	0	0.018	1
GKD-b_30_n100_m30	127.47974	9.283	134.10651	0	1140.244	0
GKD-b_3_n25_m2	**0**	0.017	0	0	0.016	1
GKD-b_4_n25_m2	**0**	0.016	0	0	0.016	1
GKD-b_5_n25_m2	**0**	0.015	0	0	0.016	1
GKD-b_6_n25_m7	**12.71796**	0.173	12.71796	12.71796	0.468	1
GKD-b_7_n25_m7	**14.09875**	0.156	14.09875	14.09875	1.732	1
GKD-b_8_n25_m7	**16.76119**	0.156	16.76119	16.76119	1.186	1
GKD-b_9_n25_m7	**17.06921**	0.172	17.06921	17.06921	0.173	1
SOM-b_2_n100_m20	6	3.042	6	0	663.937	0
SOM-b_3_n100_m30	10	5.796	12	0	99.342	0
SOM-b_4_n100_m40	13	8.715	14	0	121.602	0

Instances where CPLEX finds the optimal solution are highlighted in bold font.

runs. For each instance, the table lists the solution values and CPU times in seconds for EPR, as well as the upper and lower bounds found by CPLEX and the time taken by CPLEX. The last column in the table indicates whether CPLEX was able to prove optimality. Though CPLEX is limited to 1800 s, it often terminated before that, even when it cannot prove optimality. This occurs because its search tree could no longer fit in memory. CPLEX is able to prove optimality in 20 of the 30 instances, which are highlighted in bold font in Table 5. In 19 of those 20 instances, EPR is able to match the value of the optimal solution in a single run. In the remaining ten instances, EPR is able to improve the upper bound found by CPLEX in eight of them, match it in one, and do worse in only one. Running times for EPR are comparable to those of CPLEX on the smaller instances and, as expected, are orders of magnitude smaller than those of CPLEX on the larger instances.

5. Conclusions

This paper proposed several new hybrid heuristics for the minimum differential dispersion problem. The heuristics used components of GRASP, variable neighborhood search (VNS), and path relinking. To find a good configuration for our best heuristic, we considered two constructive procedures (four variants of each one, varying the alpha parameter), four local search procedures, including one based on VNS, and four path relinking strategies. The best configuration consisted of a GRASP with sampled greedy construction and VNS for local search. As opposed to the standard way of applying VNS where the starting solution is random, the sampled greedy constructed solution is used. During the search, and elite set of the best solutions found (with no repetition allowed) is built and maintained. After performing all the GRASP iterations, exterior path relinking is applied as a post optimization strategy, by combining all pairs of elite set solutions. Finally, the best solution found is returned.

Exterior path relinking, or *path separation*, introduced in Glover [14] and first used here, is a variant of the more common interior path relinking. In interior path relinking, paths in the neighborhood solution space connecting good solutions are explored between the solutions in the search for improvements. Exterior path relinking, as opposed to exploring paths between pairs of solutions, explores paths beyond those solutions. This is accomplished by considering an initiating solution and a guiding solution and introducing in the initiating solution attributes not present in the guiding solution. To complete the process, the roles of initiating and guiding solutions are exchanged.

Extensive computational experiments on 190 instances from the literature demonstrated the competitiveness of this algorithm. Not only was it able to outperform the GRASP heuristic of Prokopyev et al. [29] and find optimal solutions to all but one of the instances that CPLEX is able to solve, it improved the CPLEX upper bound on all but one of the instances that CPLEX failed to solve.

For future research, we note the possibility of applying a form of multiple neighborhood search different from VNS by reference to the strategic oscillation (oscillating assignment) framework as implemented in Glover et al. [17] and elaborated in Glover and Laguna [16]. We also observe the relevance of additional variants of exterior path relinking suggested in Glover [14]. These variants open the door to a wide variety of possibilities that invite closer examination and that may give an interesting basis for future research.

Acknowledgment

This research has been partially supported by the Spanish Ministry of ''Economía y Competitividad'', Grant Ref. TIN2012-35632-C02 and the Government of the Community of Madrid, grant S2013/ICE-2894.

References

[1] S. Agca, B. Eksioglu, J.B. Ghosh, Lagrangian solution of maximum dispersion problems, Naval Res. Logist. 47 (2000) 97–114.
[2] R. Aringhieri, R. Cordone, Y. Melzani, Tabu search vs. GRASP for the maximum diversity problem, 4OR: A Quart. J. Oper. Res. 6 (1) (2008) 45–60.
[3] J.R. Brown, The knaspack sharing problem, Oper. Res. 27 (2) (1979) 341–355.
[4] J.R. Brown, The sharing problem, Oper. Res. 27 (2) (1979) 324–340.
[5] V. Campos, R. Martí, J. Sanchez-Oro, A. Duarte, GRASP with PR for the orienteering problem, J. Oper. Res. Soc. (2013).
[6] A. Duarte, L.F. Escudero, R. Martí, N. Mladenović, J.J. Pantrigo, J. Sánchez-Oro, Variable neighborhood search for the vertex separation problem, Comput. Oper. Res. 39 (12) (2012) 3247–3255.
[7] A. Duarte, R. Martí, Tabu search and GRASP for the maximum diversity problem, Eur. J. Oper. Res. 178 (1) (2007) 71–84.
[8] A. Duarte, R. Martí, M.G.C. Resende, R.M.A. Silva, GRASP with path relinking heuristics for the antibandwidth problem, Networks 58 (3) (2011) 171–189.
[9] A. Duarte, J.J. Pantrigo, E.G. Pardo, J. Sánchez-Oro, Parallel variable neighbourhood search strategies for the cutwidth minimization problem, IMA J. Manage. Math. (2013).
[10] T.A. Feo, M.G.C. Resende, A probabilistic heuristic for a computationally difficult set covering problem, Oper. Res. Lett. 8 (2) (1989) 67–71.
[11] T.A. Feo, M.G.C. Resende, S.H. Smith, A greedy randomized adaptive search procedure for maximum independent set, Oper. Res. 42 (1994) 860–878.
[12] M. Gallego, A. Duarte, M. Laguna, R. Martí, Hybrid heuristics for the maximum diversity problem, Comput. Optim. Appl. 44 (3) (2009) 411–426.
[13] F. Glover, Tabu search and adaptive memory programming: advances, applications and challenges, in: R. Barr, R.V. Helgason, J.L. Kennington (Eds.), Interfaces in Computer Science and Operations Research, Operations Research/Computer Science Interfaces Series, vol. 7, Springer, US, 1997, pp. 1–75.
[14] F. Glover, Exterior path relinking for zero-one optimization, Int. J. Appl. Metaheur. Comput. 5 (3) (2014).
[15] F. Glover, C.C. Kuo, K.S. Dhir, Heuristic algorithms for the maximum diversity problem, J. Inform. Optim. Sci. 19 (1) (1998) 109–132.
[16] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Norwell, MA, USA, 1997.
[17] F. Glover, C. McMillan, R. Glover, A heuristic programming approach to the employee scheduling problem and some thoughts on managerial robots, J. Oper. Manage. 4 (2) (1984) 113–128.
[18] P. Hansen, N. Mladenović, Variable neighborhood search, in: E.K. Burke, G. Kendall (Eds.), Search Methodologies, Springer, US, 2014, pp. 313–337.

[19] P. Hansen, N. Mladenović, J.A. Moreno-Pérez, Variable neighbourhood search: methods and applications, Ann. Oper. Res. 175 (1) (2010) 367–407.
[20] G. Kortsarz, D. Peleg, On choosing a dense subgraph, in: Proceedings of the 34th Annual Symposium on Foundations of Computer Science, 1993, pp. 692–701.
[21] M. Laguna, R. Martí, GRASP and path relinking for 2-layer straight line crossing minimization, INFORMS J. Comput. 11 (1) (1999) 44–52.
[22] M. Laguna, R. Martí, Scatter Search: Methodology and Implementations in C, Springer, 2003.
[23] M. Lozano, A. Duarte, F. Gortázar, R. Martí, Variable neighborhood search with ejection chains for the antibandwidth problem, J. Heurist. 18 (2012) 919–938.
[24] R. Martí, M. Gallego, A. Duarte, A branch and bound algorithm for the maximum diversity problem, Eur. J. Oper. Res. 200 (1) (2010) 36–44.
[25] R. Martí, M. Gallego, A. Duarte, E.G. Pardo, Heuristics and metaheuristics for the maximum diversity problem, J. Heurist. 19 (4) (2013) 591–615.
[26] N. Mladenović, P. Hansen, Variable neighborhood search, Comput. Oper. Res. 24 (11) (1997) 1097–1100.
[27] G. Palubeckis, Iterated tabu search for the maximum diversity problem, Appl. Math. Comput. 189 (1) (2007) 371–383.
[28] J.J. Pantrigo, R. Martí, A. Duarte, E.G. Pardo, Scatter search for the cutwidth minimization problem, Ann. Oper. Res. 199 (1) (2012) 285–304.
[29] O.A. Prokopyev, N. Kong, D.L. Martinez-Torres, The equitable dispersion problem, Eur. J. Oper. Res. 197 (1) (2009) 59–67.
[30] M.G.C. Resende, R. Martí, M. Gallego, A. Duarte, GRASP and path relinking for the max–min diversity problem, Comput. Oper. Res. 37 (3) (2010) 498–508.
[31] M.G.C. Resende, C.C. Ribeiro, Greedy randomized adaptive search procedures: advances, hybridizations, and applications, in: M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, International Series in Operations Research & Management Science, second ed., vol. 146, Springer, US, 2010, pp. 283–319.
[32] M.G.C. Resende, C.C. Ribeiro, GRASP: greedy randomized adaptive search procedures, in: E.K. Burke, G. Kendall (Eds.), Search Methodologies, Springer, US, 2014, pp. 287–312.
[33] M.G.C. Resende, R.F. Werneck, A hybrid heuristic for the *p*-median problem, J. Heurist. 10 (1) (2004) 59–88.
[34] J. Sánchez-Oro, J.J. Pantrigo, A. Duarte, Combining intensification and diversification strategies in VNS: an application to the vertex separation problem, Comput. Oper. Res. (2013).
[35] G.C. Silva, L.S. Ochi, S.L. Martins, Experimental comparison of greedy randomized adaptive search procedures for the maximum diversity problem, in: Experimental and Efficient Algorithms, Lecture Notes in Computer Science, vol. 3059, Springer, Berlin, Heidelberg, 2004, pp. 498–512.
[36] M.B. Teitz, Toward a theory of urban public facility location, Papers Reg. Sci. Assoc. 21 (1) (1968) 35–51.