Computers & Operations Research 37 (2010) 498 – 508

Contents lists available at ScienceDirect

Computers & Operations Research

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

GRASP and path relinking for the max–min diversity problem

M.G.C. Resende[a], R. Martí[b,*], M. Gallego[c], A. Duarte[c]

[a]Algorithms and Optimization Research Department, AT&T Labs Research,180 Park Avenue, Room C241, Florham Park, NJ 07932, USA
[b]Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain
[c]Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Spain

ARTICLE INFO

Available online 28 May 2008

Keywords:
Max–min diversity problem
Metaheuristics
Adaptive memory programming
GRASP
Path relinking

ABSTRACT

The max–min diversity problem (MMDP) consists in selecting a subset of elements from a given set in such a way that the diversity among the selected elements is maximized. The problem is NP-hard and can be formulated as an integer linear program. Since the 1980s, several solution methods for this problem have been developed and applied to a variety of fields, particularly in the social and biological sciences. We propose a heuristic method—based on the GRASP and path relinking methodologies—for finding approximate solutions to this optimization problem. We explore different ways to hybridize GRASP and path relinking, including the recently proposed variant known as GRASP with evolutionary path relinking. Empirical results indicate that the proposed hybrid implementations compare favorably to previous metaheuristics, such as tabu search and simulated annealing.

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. As stated in [1], there are basically two approaches to formulate these problems: the max–sum and the max–min models. Both have received much attention in recent years. The former, also known as the *maximum diversity problem* (MDP) has been studied in [2–4]. For the *max–min diversity problem* (MMDP), both exact [5] and heuristic approaches, such as simulated annealing (SA) [6], tabu search (TS) [6], and GRASP [7] have been proposed. Because of the *flat landscape* of max–min problems, these papers agree that the MMDP presents a challenge to solution methods based on heuristic optimization.

The MMDP consists in selecting a subset M of m elements ($|M|=m$) from a set N of n elements in such a way that the minimum distance between the chosen elements is maximized. The definition of distance between elements is customized to specific applications. As mentioned in Kuo et al. [1] and Glover et al. [2], the MMDP has applications in plant breeding, social problems, and ecological preservation. In most of these applications, it is assumed that each element can be represented by a set of attributes. Let s_{ik} be the state or value of the k-th attribute of element i, where $k = 1, \ldots, K$. The distance between elements i and j can be defined as

$$d_{ij} = \sqrt{\sum_{k=1}^{K} (s_{ik} - s_{jk})^2}.$$

In this case, d_{ij} is simply the Euclidean distance between i and j. The distance values are then used to formulate the MMDP as a quadratic binary problem, where for $i = 1, \ldots, n$, variable x_i takes the value 1 if element i is selected and 0 otherwise:

$$\begin{aligned} (\text{MMDP})\ \max \quad & z_{MM}(x) = \min_{i<j} d_{ij} x_i x_j \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i = m, \\ & x_i = \{0, 1\}, \quad i = 1, \ldots, n. \end{aligned}$$

Erkut [5] and Ghosh [7] showed independently that the MMDP is NP-hard. The MDP can be formulated in a similar way by simply replacing the objective function, $z_{MM}(x)$, in the formulation above with

$$z_{MS}(x) = \sum_{i<j} d_{ij} x_i x_j.$$

Although the MDP and the MMDP are related, we should not expect a method developed for the MDP to perform well on the MMDP. The example in Fig. 1 illustrates that the correlation between the values of the solutions in both problems can be relatively low.

* Corresponding author. Tel.: +34 96 354 3090; fax: +34 96 386 4735.
E-mail addresses: mgcr@research.att.com (M.G.C. Resende), rafael.marti@uv.es (R. Martí), micael.gallego@urjc.es (M. Gallego), abraham.duarte@urjc.es (A. Duarte).

0305-0548/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2008.05.011

	1	2	3	4	5	6	7
1	-	4.6	6.2	2.1	3.5	3.6	4.4
2	4.6	-	6.6	7.1	8.2	2.4	5.3
3	6.2	6.6	-	7.3	3.3	2.4	3.8
4	2.1	7.1	7.3	-	5.5	1.1	2.3
5	3.5	8.2	3.3	5.5	-	6.4	3.4
6	3.6	2.4	2.4	1.1	6.4	-	5.4
7	4.4	5.3	3.8	2.3	3.4	5.4	-

Fig. 1. Distance matrix of an instance with $n = 7$.

Suppose we have seven elements of which we need to select five. Furthermore, the distances between each pair of elements are given by the matrix of Fig. 1. For such a small example, we can enumerate all possible solutions (selections of m out of n elements) and compute for each one the values $z_{MS}(x)$ and $z_{MM}(x)$. The correlation between both objective functions is 0.52, which can be considered relatively low. Moreover, we find that the optimal solution x^* of the MDP has a value $z_{MS}(x^*) = 54.4$ and a value $z_{MM}(x^*) = 2.1$. However, the optimal solution y^* of the MMDP has a value $z_{MM}(y^*) = 3.3$, which is relatively larger than $z_{MM}(x^*)$. Moreover, 30% of the solutions present a $z_{MM}(x)$ value larger than $z_{MM}(x^*)$. Therefore, we should not expect a method for the MDP to obtain good solutions for the MMDP. In this paper, we restrict our attention to solution methods specifically designed for the MMDP.

In Section 2, we describe previous work. In this paper, we explore the hybridization of the GRASP, path relinking (PR), and evolutionary path relinking (EvPR) methodologies to find optimal or near-optimal solutions to the MMDP. Then, we introduce our algorithms in Sections 3 and 4. Computational experiments are described in Section 5 and concluding remarks are made in Section 6.

2. Previous methods

Chandrasekaran and Daughety [8] introduced the MMDP under the name of *m-dispersion problem*. They proposed two simple polynomial-time heuristics for the special case of tree networks. Kuby [9] introduced the problem on general networks, proposing integer linear programming formulations with $O(n^2)$ constraints and n binary variables, and tested the formulations on instances of dimension $n = 25$ ($m = 5$ and 10).

Erkut [5] proposed a branch and bound algorithm and a heuristic method. The branch and bound method was able to solve problems with $n = 40$ in half an hour of CPU time (on an AT-compatible microcomputer with clock speed of 10 MHz). The heuristic method consists of construction plus local search. The construction starts with the infeasible solution where all n elements are selected. To reduce the set of selected elements to m, the procedure performs $n \cdot m$ steps. At each step, it de-selects the element i^* with an associated shortest distance. Note that given a shortest distance d_{ij} there are at least two elements with this associated distance. The method randomly selects one of them. The construction can be repeated, obtaining a different solution each time. The local search method scans the set of selected elements in search of the best exchange to replace a selected element with an unselected one. The method performs moves as long as the objective value increases and stops when no improving exchange can be found.

Kincaid [6] proposed two heuristics for the MMDP based on exchanges: a SA heuristic and a TS heuristic. In a given iteration, the SA method generates a random move (an exchange between a selected and an unselected element). If it is an improving move, it is automatically made; otherwise, it still may be made with positive probability. The so-called temperature and cooling schedule in the SA that manage the evolution of this acceptance probability are implemented according to Lundi and Mess [10]. The algorithm starts with an initial temperature equal to the largest distance value and it is reduced according to the factor `tfactr` = 0.89. For each temperature value, `sample_size` = $10n$ moves are generated. The SA method terminates when a maximum number of iterations `max_it` = 80 is reached (note that within this number of iterations the temperature value is still strictly positive).

The TS heuristic also performs exchange moves. At each iteration, `sample_size` = $10n$ moves are considered and the method performs the best admissible move among them. Admissible here refers to the tabu status. When a move is performed and a selected item i is exchanged with an unselected item j, the unordered pair (i,j) is recorded in the tabu list and is labeled tabu for `tabu_size` = 20 iterations. A selected move is admissible if it is not labeled tabu, or if its value improves upon the best known solution (aspiration criterion). After `max_it` = 65 iterations, the search is re-initiated from a new initial solution for diversification purposes. The author examines the performance of both methods on 30 instances of size $n = 25$ (in three groups of 10 with different characteristics) and m ranging from 5 to 15.

Kuo et al. [1] proposed the following improved integer linear programming formulation:

$$
\begin{aligned}
\max \quad & Z = w \\
\text{s.t.} \quad & \sum_{i=1}^{n} x_i = m, \\
& (C - d_{ij})y_{ij} + w \leqslant C, \quad 1 \leqslant i < j \leqslant n, \\
& x_i + x_j - y_{ij} \leqslant 1, \quad 1 \leqslant i < j \leqslant n, \\
& -x_i + y_{ij} \leqslant 0, \quad 1 \leqslant i < j \leqslant n, \\
& -x_j + y_{ij} \leqslant 0, \quad 1 \leqslant i < j \leqslant n, \\
& y_{ij} \geqslant 0, \quad 1 \leqslant i < j \leqslant n
\end{aligned}
$$

and illustrated it on small examples. The constant C takes an arbitrarily large value. The authors state that, for both exact and heuristic methods, the MMDP is harder to solve than the MDP. As reported in Section 5, we will use this formulation to solve small-size instances with an integer linear programming solver.

Ghosh [7] proposed a solution construction procedure and a local search method. Given a set N with n elements, the construction method performs m steps as follows. Let M_{k-1} be a partial solution with $k-1$ elements ($1 \leqslant k \leqslant m$). For any $i \in N \backslash M_{k-1}$, let $\Delta z_{MM}(i)$ be the contribution of i to the value of the solution. Let $\Delta z_{\mathrm{L}}(i)$ and $\Delta z_{\mathrm{U}}(i)$ be, respectively, a lower and an upper bound of $\Delta z_{MM}(i)$. $\Delta z_{\mathrm{L}}(i)$ is computed as the minimum between the value of the current solution and the minimum distance between i and the other elements in N. $\Delta z_{\mathrm{U}}(i)$ is computed by first sorting the distances between i and the elements in $N \backslash M_{k-1}$ and then computing the smallest among the largest $m-k$ elements. Then $\Delta z'(i) = (1-u)\Delta z_{\mathrm{L}}(i) + u\Delta z_{\mathrm{U}}(i)$ is an estimate of $\Delta z_{MM}(i)$ (where u is a random number from the U(0,1) uniform distribution). The element i^* with the largest value of the estimate is selected to be included in the partial solution:

$$M_k = M_{k-1} \cup \{i^*\}, \quad \Delta x'(i^*) = \max_{i \in N \backslash M_{k-1}} \{\Delta z'(i)\}.$$

Starting with a randomly selected element, this process is repeated until M_m is finally delivered as the output of the construction ($|M_m| = m$). The local search is similar to the one introduced in Erkut [5] and begins at the conclusion of the construction phase, attempting to improve upon an incumbent solution through neighborhood search. The neighborhood of a solution is the set of all solutions obtained by replacing one element by another. Given a solution M, for each $i \in M$ and $j \in N \backslash M$, we compute the move value $\Delta z_{MM}(i,j)$ associated with the exchange of i and j. The method scans the entire neighborhood and performs the move with the largest Δz_{MM} value, if it improves

```
begin GRC
1   Sel ← ∅;
2   Select i randomly from N;
3   Sel ← {i};
4   while |Sel| < m do
5       d_j ← min_{k∈Sel} d_jk, ∀j ∈ CL = N \ Sel;
6       d* ← max_{j∈CL} d_j;
7       RCL ← {j | d_j ≥ α · d*};
8       Select i* randomly in RCL;
9       Sel ← Sel ∪ {i*};
10  end-while;
end
```

Fig. 2. Constructive heuristic GRC.

the current solution. The current solution is updated as well as its corresponding value. The search stops when no move improves the current solution (i.e. when $\Delta z_{MM}(i,j) \leqslant 0$ for all i and j). The method performs 10 global phases (construction followed by improvement) and the best solution overall is returned as the output.

3. GRASP

The GRASP methodology was developed in the late 1980s [11,12] and the acronym was coined in Feo et al. [13]. We refer the reader to Resende and Ribeiro [14] for a recent survey of this metaheuristic. Each GRASP iteration consists in constructing a trial solution and then applying local search from the constructed solution. The construction phase is iterative, randomized greedy, and adaptive. In this section we describe our adaptation of the GRASP methodology for the MMDP.

3.1. Construction procedures

From the previous algorithms reviewed in Section 2, we can point to two construction procedures. ErkC, the method proposed in Erkut [5] is based on de-selecting elements, and GhoC, the one due to Ghosh [7], is based on an estimate of the contribution of the elements. In this section, we propose two new construction methods based on the GRASP methodology.

Given a set N with n elements, the construction procedure GRC performs m steps to produce a solution with m elements as shown in Fig. 2. The set *Sel* represents the partial solution under construction. At each step, GRC selects a candidate element $i^* \in CL = N \backslash Sel$ with a large distance to the elements in the partial solution *Sel*. Specifically, it first computes d_j as the minimum distance between element j and the selected elements. Then, it constructs the restricted candidate list, *RCL*, with all the candidate (unselected) elements j with a distance value d_j within a fraction α ($0 \leqslant \alpha \leqslant 1$) of the maximum distance $d^* = \max\{d_j | j \in CL\}$. Finally, GRC randomly selects an element in *RCL*.

GRC implements a typical GRASP construction in which first each candidate element is evaluated by a greedy function to construct the *RCL* and then an element is selected at random from the *RCL*. We now consider GRC2, an alternative construction procedure introduced in Resende and Werneck [15] as random plus greedy construction. In GRC2 we first randomly choose candidates and then evaluate each candidate according to a greedy function to make the greedy choice. GRC2 first constructs the restricted candidate list *RCL2* with a fraction β ($0 \leqslant \beta \leqslant 1$) of the elements in *CL* selected at random. Then, it evaluates all the elements in *RCL2*, computing d_j for all $j \in RCL2$, and selects the best one, i.e. the element j^* such that

$$d_{j^*} = \max_{j \in RCL2} d_j.$$

In the computational study, we discuss how search parameters α and β affect GRC and GRC2, respectively. We also test the *reactive* variants (Reactive-GRC and Reactive-GRC2) in which the value of the parameter is randomly determined according to an empirical distribution of probabilities [16].

During the initial constructions of Reactive-GRC (Reactive-GRC2), the value of α (β) is randomly selected from the set $S = \{0, 0.1, 0.2, \ldots, 0.9, 1\}$ with a uniform distribution. Twenty percent of the constructions sample from the uniform distribution while 80% sample according to the *hits* value. In each iteration, we test whether the constructed solution $x(a)$ obtained with $\alpha = a$ ($\beta = a$), has a value $z_{MM}(x(a))$ within a pre-established threshold of the best constructed solution so far.[1] In this case, we increment *hits*(a) by one unit (where *hits*(i) is initially set to zero for all $i \in S$). Otherwise, *hits*(a) remains unchanged. Therefore, initially all the values considered in S have the same opportunity to be selected for construction. However, as the algorithm progresses, those values better suited for a particular instance (those that produce better constructions) are more frequently selected. In this way, the reactive construction customizes the best value (or values) of the parameter for each instance. Note that in the non-reactive variants described earlier, the selection of the parameter is made offline and adjusted to a fixed value for all the instances considered.

3.2. Local search methods

Erkut [5] and Ghosh [7] propose the local search method BLS, based on the best-improvement strategy, in which at each iteration the method scans the entire neighborhood in search of the best exchange (between a selected and an unselected element). In what follows, we propose two new local search methods based on the first-improvement strategy (also known as *mildest ascent*). The first method, FLS, consists in a straightforward implementation of this strategy, while the second, called improved local search (IMLS), explores the neighborhood according to an evaluation function.

Given a set N with n elements, and a solution *Sel* with m selected elements, we compute the following values:

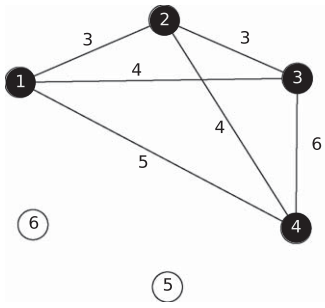
$$d_i = \min_{j \in Sel} d_{ij}, \quad d^* = \min_{j \in Sel} d_j,$$

where d_i is the minimum distance of element i to the selected elements (those in *Sel*), and d^* is the objective function of the current solution, i.e. $d^* = z_{MM}(Sel)$. It is clear that to improve a solution we need to remove (and thus replace) the elements i in the solution for which $d_i = d^*$.

The FLS method scans, at each iteration, the list of elements in the solution ($i \in Sel$) with minimum d_i value, i.e. for which $d_i = d^*$. It scans the list of elements in lexicographical order, starting with a randomly selected element. Then, for each element i with a minimum d_i-value, FLS examines the list of unselected elements ($j \in N \backslash Sel$) in search for the first improving exchange. The unselected elements are also examined in lexicographical order, starting with a randomly selected element. The method performs the first improving move ($Sel \leftarrow Sel \backslash \{i\} \cup \{j\}$) and updates d_i for all elements $i \in Sel$ as well as the objective function value d^*, concluding the current iteration. The algorithm repeats iterations as long as improving moves can be performed and stops when no further improvement is possible. As described below, the definition of "improving" is not limited to the objective function.

The example in Fig. 3 with $n = 6$ and $m = 4$ illustrates the performance of the local search procedure. Consider the solution

[1] We have empirically found that a conservative value of 90% for this threshold provides good results.



	1	2	3	4	5	6
1	-	3	4	5	7	5
2	3	-	3	4	8	1
3	4	3	-	6	4	7
4	5	4	6	-	5	9
5	7	8	4	5	-	8
6	5	1	7	9	8	-

Fig. 3. Local search performance.

$Sel = \{1,2,3,4\}$, depicted with dark circles, with a value of $d^* = 3$ in which we perform an iteration of the FLS method. For the sake of clarity, Fig. 3 only depicts some of the distances between the elements. The d_i values of the elements in the solution are $d_1 = 3$, $d_2 = 3$, $d_3 = 3$, and $d_4 = 4$. The FLS method selects an element with minimum d_i value, say for example $i = 1$. It then scans the list of unselected vertices in search for an improving move. Note, however, that when we remove element 1, elements 2 and 3 remain in the solution and therefore d^* will be equal to $d_{23} = 3$, regardless of the element that we introduce in the solution to replace element 1. Then, strictly speaking, we cannot find any improving exchange when we remove element 1. On the other hand, it is clear that in a certain sense the solution improves when we remove element 1, because the number of elements for which d^* is reached decreases and therefore we can say that we are closer to obtaining a better solution. This is why we consider an extended definition of improving for a given move, including not only when the move increases the value of d^*, but also when d^* remains fixed and the number of elements i with $d_i = d^*$ is reduced. In this example, when we replace element 1 with element 5 obtaining $Sel' = \{2,3,4,5\}$, we say that this is an improving move, because $d^* = 3$ and d_i only matches d^* in two elements (2 and 3), which compares favorably with the initial solution Sel (in which three elements matched $d^* = 3$).

The example in Fig. 3 also illustrates that when we select an element for exchange, it would be better to consider not only the distance with the closest element, but also the second closest, third closest, and so on. Given an element i, let $d_i^1, d_i^2, \ldots, d_i^k$ be its k lowest distance values between i and the m elements in the solution ($k < m$) sorted in increasing order ($d_i = d_i^1$). In the example, $d_1^1 = 3$, $d_1^2 = 4$, $d_2^1 = 3$, and $d_2^2 = 3$. Then it is better to remove element 2 instead of element 1 because by removing element 2 the objective d^* could increase to $d_2^3 = 4$. Therefore, we propose a new local search method, that we call IMLS, which is based on the evaluation of the value

$$e(i) = \sum_{j=1}^{k} \frac{d_i^j}{j}$$

for elements $i \in Sel$ with $d_i = d^*$, according to each element's lowest k distance values (where k is a search parameter).

The local search method IMLS selects, at each iteration, the element i^* with the lowest $e(i)$ value among the selected elements $i \in Sel$ with $d_i = d^*$. It then moves this element from the solution: $Sel \leftarrow Sel \backslash \{i^*\}$ to the unselected set, and computes the $e(s)$ value for all elements $s \in N \backslash Sel$. The method then scans the elements in $N \backslash Sel$ in decreasing order of $e(s)$ and performs the first improving move. If no improving move is found, the method selects the next element with lowest $e(i)$ value among the selected elements $i \in Sel$ with $d_i = d^*$ and tries to find an improving move. We also apply here the definition of improving move introduced in FLS (increasing the value of d^*, or keeping d^* fixed and reducing the number of elements i with $d_i = d^*$). The method stops when no further improvement is possible.

4. Path relinking

PR was suggested as an approach to integrate intensification and diversification strategies in the context of TS [17,18]. This approach generates new solutions by exploring trajectories that connect high-quality solutions—by starting from one of these solutions, called an *initiating solution*, and generating a path in the neighborhood space that leads toward the other solutions, called *guiding solutions*. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions, and incorporating them in an *intermediate solution* initially originated in the initiating solution.

Laguna and Martí [19] adapted PR in the context of GRASP as a form of intensification. The relinking in this context consists in finding a path between a solution found with GRASP and a chosen elite solution. Therefore, the relinking concept has a different interpretation within GRASP since the solutions found from one GRASP iteration to the next are not linked by a sequence of moves (as in the case of TS). Resende and Ribeiro [14] present numerous examples of GRASP with PR. In this section we explore the adaptation of GRASP with PR to the MMDP across different designs in which greedy, randomized, and evolutionary elements are considered in the implementation.

4.1. Greedy path relinking

Let x and y be two solutions of the MMDP, interpreted as the sets of m selected elements Sel_x and Sel_y, respectively ($|Sel_x| = |Sel_y| = m$). The path relinking procedure PR(x,y) starts with the first solution x, and gradually transforms it into the second one y, by swapping out elements selected in x with elements selected in y. The elements selected in both solutions x and y, Sel_{xy}, remain selected in the intermediate solutions generated in the path between them. Let Sel_{x-y} be the set of elements selected in x and not selected in y and symmetrically, let Sel_{y-x} be the set of elements selected in y and not selected in x, i.e.

$$Sel_{xy} = Sel_x \cap Sel_y, \quad Sel_{x-y} = Sel_x \backslash Sel_{xy}, Sel_{y-x} = Sel_y \backslash Sel_{xy}.$$

Let $p_0(x,y) = x$ be the initiating solution in the path P(x,y) from x to y. To obtain the solution $p_1(x,y)$ in this path, we unselect in $p_0(x,y)$ a single element $i \in Sel_{x-y}$, and select a single element $j \in Sel_{y-x}$, thus obtaining

$$Sel_{p_1(x,y)} = Sel_{p_0(x,y)} \backslash \{i\} \cup \{j\}.$$

In the *greedy path relinking* (GPR) algorithm, the selection of the elements i and j is made in a greedy fashion. To obtain $p_{k+1}(x,y)$ from $p_k(x,y)$, we evaluate all the possibilities for $i \in Sel_{p_k(x,y)-y}$ to be de-selected and $j \in Sel_{x-p_k(x,y)}$ to be selected, and perform the best swap. In this way, we reach y from x in $r = |Sel_{x-y}| = |Sel_{y-x}|$ steps, i.e. $p_r(x,y) = y$. The output of the PR algorithm is the best solution, different from x and y, found in the P(x,y) path (among $p_1(x,y), p_2(x,y), \ldots, p_{r-1}(x,y)$).

The PR algorithm operates on a set of solutions, called *elite set* (*ES*), constructed with the application of a previous method. In this paper, we apply GRASP to build the elite set. If we only consider a quality criterion to populate the elite set, we could simply populate the elite set with the best $|ES|$ solutions generated with GRASP. However, previous studies [15] have empirically found that an application of PR to a pair of solutions is likely to be unsuccessful if the solutions are very similar. Therefore, to construct *ES* we will consider both quality and diversity.

Initially *ES* is empty, and we apply GRASP for $b = |ES|$ iterations and populate it with the solutions obtained. We order the solutions in *ES* from the best (x^1) to the worst (x^b). Then, in the following GRASP iterations, we test whether the generated (constructed and

```
begin StaticGRASP+PR
1   GlobalIter ← number of global iterations;
2   Apply GRASP (construction and local search) for b = |ES| iterations
    to populate ES ← {x^1, x^2, ..., x^b};
3   iters ← b + 1;
4   while iters ≤ GlobalIter do
5       x ← GRASP construction phase;
6       x' ← GRASP local search starting at x;
7       if z_MM(x') > z_MM(x^1) or (z_MM(x') > z_MM(x^b) and d(x', ES) ≥ dth) then
8           x^k ← closest solution to x' in ES with z_MM(x') > z_MM(x^k);
9           ES ← ES \ {x^k};
10          Insert x' into ES so that ES remains sorted from best x^1 to worst x^b;
11      end-if;
12      iters ← iters + 1;
13  end-while;
14  x^best ← x^1;
15  for (i = 1 to b − 1 and j = i + 1 to b do
16      Apply PR(x^i, x^j) and PR(x^j, x^i) and let x ← best solution found;
17      x' ← local search phase of GRASP starting from x;
18      if z_MM(x') > z_MM(x^best) then
19          x^best ← x';
20      end-if;
21  end-for;
22  return x^best;
end
```

Fig. 4. GRASP with PR in a static variant.

improved) solution x' qualifies to enter *ES*. Specifically, if x' is better than the best x^1, it is put in the set. Moreover, if it is better than the worst x^b and it is sufficiently different from the other solutions in the elite set ($d(x', ES) \geqslant dth$), it is also put in *ES*. The parameter *dth* is a distance threshold value that reflects the term "sufficiently different" and it is empirically adjusted (see Section 5). To keep the size of *ES* constant and equal to *b*, whenever we add a solution to this set, we remove another one. To maintain the quality and the diversity, we remove the closest solution to x' in *ES* among those worse than it in value. Fig. 4 shows pseudo-code of the GRASP with PR algorithm.

The design in Fig. 4 is called *static* since we first apply GRASP to construct the elite set *ES* and then we apply PR to generate solutions between all the pairs of solutions in *ES*. Given two solutions in *ES*, x and y, we apply PR in both directions, i.e. PR(x, y) from x to y and PR(y, x) from y to x. The best solution generated in both paths is subjected to the local search method for improved outcomes. As shown in Fig. 4, we always keep the best solution in the elite set (x^1) during the realization of the GRASP phase and we only replace it when the new solution generated improves it in quality. The algorithm terminates when PR is applied to all the pairs in *ES* and the best overall solution x^{best} is returned as the output.

As aforementioned, distance is used to measure how diverse one solution is with respect to a set of solutions. Specifically, for the MMDP, let x_i^r be the value of the i-th variable for the elite solution $r \in ES$. Also let x_i^t be the value of the i-th variable for the trial solution t. Then, the distance between the trial solution t and the solutions in the *ES* is defined as

$$d(t, ES) = b \cdot m - \sum_{r=1}^{b} \sum_{i:x_i^t=1} x_i^r.$$

The formula simply counts the number of times that each selected element in the trial solution t appears in the elite solutions and subtracts this value from the maximum possible distance (i.e., $b \cdot m$). The maximum distance occurs when no element that is selected in the trial solution t appears in any of the elite solutions in *ES*.

An alternative implementation of GRASP with PR consists in a *dynamic* update of the elite set as introduced in Laguna and Martí [19]. In this design, each solution x' generated with GRASP is directly subjected to the PR algorithm, which is applied between x' and a solution x^j selected from *ES*. The selection is probabilistically made according to the value of the solutions. As in the *static* design, the local search method is applied to the output of PR, but now, the resulting solution is directly tested for inclusion in *ES*. If successful, it can be used as guiding solution in later applications of PR. Fig. 5 shows pseudo-code for this dynamic variant.

In our computational experience, described in Section 5, we compare the static variant versus the dynamic variant with respect to both quality and speed.

4.2. Greedy randomized path relinking

Faria et al. [20] introduced *greedy randomized path relinking* (GRPR) where instead of moving between the initiating and the guiding solutions in a greedy way, the moves are done in a greedy randomized fashion.

As described above for the GPR algorithm, at each step in the path from the initiating solution x to the guiding solution y, the selection of the elements i (to be de-selected) and j (to be selected) is made in a greedy fashion. In the GRPR algorithm, we construct a set of good candidates i and j for swapping and randomly select one among them. This procedure mimics the selection method employed in a GRASP construction.

To obtain $p_{k+1}(x, y)$ from $p_k(x, y)$, we evaluate all the possibilities for $i \in Sel_{p_k(x,y)-y}$ to be de-selected and $j \in Sel_{y-p_k(x,y)}$ to be selected. The candidate set C contains all these swaps, i.e.

$$C_k(x, y) = \{(i, j) | i \in Sel_{p_k(x,y)-y}, j \in Sel_{y-p_k(x,y)}\}.$$

Let $z(i, j)$ be the value of the move associated with de-select i and select j in the current solution $p_k(x, y)$ to obtain $p_{k+1}(x, y)$. Then,

$$z(i, j) = z_{MM}(p_{k+1}(x, y)) - z_{MM}(p_k(x, y)).$$

```
begin DynamicGRASP+PR
1   GlobalIter ← number of global iterations;
2   Apply GRASP (construction and local search) for b = |ES| iterations
    to populate ES ← {x^1, x^2, ..., x^b};
3   iter ← b + 1;
4   while iters ≤ GlobalIter do
5       x ← GRASP construction phase;
6       x' ← GRASP local search starting at x;
7       Randomly select x^j from ES;
8       Apply PR(x', x^j) and PR(x^j, x') and let y be the best solution found;
9       y' ← GRASP local search starting at y;
10      if z_MM(y') > z_MM(x^1) or (z_MM(y') > z_MM(x^b) and d(y', ES) ≥ dth) then
11          x^k ← closest solution to y' in ES with z_MM(y') > z_MM(x^k);
12          Add y' to ES and remove x^k;
13          Sort ES from best x^1 to worst x^b;
14      end-if;
15  end-while;
16  x^best ← x^1;
17  return x^best;
end
```

Fig. 5. GRASP with PR in a dynamic variant.

In step k of the path from x to y, the restricted candidate list $RCL_k(x,y)$ of good candidates for swapping is

$$RCL_k(x,y) = \{(i,j) \in C_k(x,y) | z(i,j) \geqslant \delta z^*\},$$

where z^* is the maximum of $z(i,j)$ in $C_k(x,y)$ and δ $(0 \leqslant \delta \leqslant 1)$ is a search parameter. A pair $(i,j) \in RCL_k(x,y)$ is randomly selected and the associated swap is performed.

In the application of PR in the GRASP with PR algorithm, we can apply the greedy variant (GPR) described in Section 4.1 or the randomized variant (GRPR) described in this subsection. Specifically, in the static variant we only need to apply GPR or GRPR in step 16 of the pseudo-code shown in Fig. 4, and similarly in the dynamic variant we apply one or the other in step 8 of the pseudo-code in Fig. 5.

4.3. Truncated PR

As aforementioned, PR explores a path in the solution space from an initiating solution $x = p_0(x,y)$ to a guiding solution $y = p_r(x,y)$, where $r = |Sel_{x-y}| = |Sel_{y-x}|$ is the number of steps from x to y.

At each intermediate solution $p_k(x,y)$, a restricted neighborhood of the solution is searched for the next solution in the path from $p_k(x,y)$ to y. The neighborhood is restricted because only moves that remove element $i \in Sel_{p_k(x,y)-y}$ and put in its place an element $j \in Sel_{y-p_k(x,y)}$ are allowed. As the procedure moves from one intermediate solution to the next, the cardinalities of sets $Sel_{p_k(x,y)-y}$ and $Sel_{y-p_k(x,y)}$ decrease by one element each. Consequently, as the procedure nears the guiding solution, there are fewer allowed moves to explore and the search tends to be less effective. This suggests that PR tends to find good solutions near the initiating solution since it can explore the solution space more effectively around that solution. If this happens, then the effort made by PR near the guiding solution is fruitless.

In truncated PR, a new stopping criterion is used. Instead of continuing the search until the guiding solution is reached, only κ steps are allowed, i.e. the resulting path in the solution space is $p_1(x,y), p_2(x,y), \ldots, p_\kappa(x,y)$ and the best of these solutions is returned as the PR solution.

4.4. Evolutionary path relinking

Resende and Werneck [15] introduced EvPR as a post-processing phase for GRASP with PR (see also [21]). In EvPR, the solutions in the elite set (*ES*) are evolved in a similar way that the reference set evolves in scatter search (SS) [24].

As in the dynamic variant of GRASP with GPR, in GRASP with EvPR we apply in each iteration the construction and the improvement phase of GRASP as well as the PR method to obtain the elite set (see steps 5–9 in the pseudo-code shown in Fig. 5). After a pre-established number of iterations the GRASP with GPR stops. However, in GRASP with EvPR, a post-processing phase based on PR is applied to each pair of solutions in *ES*. The solutions obtained with this latter application of PR are considered to be candidates to enter *ES*, and PR is again applied to them as long as new solutions enter *ES*. This way we say that *ES* evolves. Fig. 6 shows the pseudo-code of GRASP with EvPR in which this process is repeated for *GlobalIter* iterations.

GRASP with EvPR and SS are evolutionary methods based on evolving a small set of selected solutions (elite set in the former and reference set in the latter). We can, therefore, observe similarities between them. In some implementations of SS, GRASP is used to populate the reference set, but note that other constructive methods can be used as well. Similarly, PR can be used to combine solutions in SS, but we can use any other combination method [22]. From an algorithmic point of view, we may find two main differences between these methods. The first one is that in SS we do not apply PR to the solutions obtained with GRASP (as we do in steps 7 and 8 in pseudo-code of GRASP with EvPR shown in Fig. 6), but rather, we only apply PR as a combination method between solutions already in the reference set. The second difference is that in SS when none of the new solutions obtained with combinations are admitted to the reference set (elite set), it is rebuilt, removing some of its solutions, as specified in the *reference set update method* [22]. In GRASP with EvPR we do not remove solutions from *ES*, but rather, we again apply GRASP (starting from step 5) and use the same rules for inclusion in the *ES*.

5. Computational experiments

This section describes the computational experiments that we performed to test the efficiency of our GRASP with PR procedures as well as to compare them with the previous methods identified to be the state-of-the-art for the MMDP. We implemented the methods

```
begin GRASP+EvPR
1   GlobalIter ← number of global iterations;
2   Apply GRASP (construction and local search) for b = |ES| iterations
    to populate ES ← {x^1, x^2, ..., x^b};
3   for iter = 1, ..., GlobalIter do
4       for i = 1, ..., LocalIter do
5           x ← GRASP construction phase;
6           x' ← GRASP local search starting at x;
7           Randomly select x^j from ES;
8           Apply PR(x', x^j) and PR(x^j, x') and let y be the best solution found;
9           y' ← GRASP local search starting at y;
10          if z_MM(y') > z_MM(x^1) or (z_MM(y') > z_MM(x^b) and d(y', ES) ≥ dth) then
11              x^k ← closest solution to y' in ES with z_MM(y') > z_MM(x^k);
12              Add y' to ES and remove x^k;
13              Sort ES from best x^1 to worst x^b;
14          end-if;
15      end-for;
16      NewSol ← 1;
17      while NewSol do
18          NewSol ← 0;
19          Apply PR(x, x') and PR(x', x) for every pair (x, x') in ES
            not combined before. Let y be the best solution found;
20          y' ← GRASP local search starting at y;
21          if z_MM(y') > z_MM(x^1) or (z_MM(y') > z_MM(x^b) and d(y', ES) ≥ dth) then
22              x^k ← closest solution to y' in ES with z_MM(y') > z_MM(x^k);
23              Add y' to ES and remove x^k;
24              Sort ES from best x^1 to worst x^b;
25              NewSol ← 1;
26              x^best ← x^1;
27          end-if;
28      end-while;
29  end-for;
30  return x^1;
end
```

Fig. 6. GRASP with EvPR.

in Java SE 6 and solved the integer linear programming formulation described in Section 2 with Cplex 8.0. All the experiments were conducted on a Pentium 4 computer running at 3 GHz with 3 GB of RAM. We have employed three sets of instances in our experiments:

Glover: This data set consists of 75 matrices for which the values were calculated as the Euclidean distances from randomly generated points with coordinates in the 0–100 range. The number of coordinates for each point is also randomly generated between 2 and 21. Glover et al. [2] developed this test problem generator and constructed instances with $n=10, 15$, and 30. The value of m ranges from $0.2n$ to $0.8n$.

Geo: This data set consists of 60 matrices constructed with the same test problem generator employed in the *Glover* set. We generated 20 instances with $n=100, 250$, and 500. For each value of n we consider $m=0.1n$, $0.3n$ (generating 10 instances for each combination of n and m). These instances are similar to the *geometric* instances introduced in Erkut [5].

Ran: This data set consists of 60 matrices with random numbers. These instances are based on the generator introduced by Silva et al. [3]. As for the *Geo* set, we generated 20 instances with $n=100, 250$, and 500 (and for each value of n we consider $m=0.1n$, $0.3n$). The integer random numbers are generated between 50 and 100 in all the instances except when $n=500$ and $m=150$ in which they are generated between 1 and 200 (to make them harder in terms of comparison among heuristics).

In each experiment, we compute for each instance the overall best solution value, *BestValue*, obtained by all executions of the methods considered. 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 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*. We also report the statistic *Score* achieved by each method, as described in Ribeiro et al. [23]. For each instance, the *n_score* of a method *M* is defined as the number of methods that found a better solution than *M*. In case of ties, all the methods receive the same *n_score*, equal to the number of methods strictly better than all of them. We then report Score as the sum of the *n_score* values across all the instances in the experiment. Thus, the lower the Score the better the method.

Table 1
New constructive methods on *Geo* and *Ran* instances with $n=100, 250$

	GRC(α)				GRC2(β)			
	0.75	0.90	0.95	Reactive	0.75	0.90	0.95	Reactive
Dev. (%)	9.23	2.51	1.09	0.81	0.70	0.58	0.66	1.07
#Best	0	5	10	16	21	21	18	15
Score	277	184	101	60	41	43	51	90

In our preliminary experimentation we consider the set of 40 instances formed with 10 instances from the *Geo* set with $n=100$ and 10 with $n=25$ and similarly from the *Ran* set (half with $m=0.1n$ and half with $m=0.3n$). In the first preliminary experiment, we study the parameter α in constructive method GRC as well as the parameter β in the constructive method GRC2. We run GRC and GRC2 100 times, thus obtaining 100 solutions for each method and instance pair. Table 1 reports, for this set of 40 instances and each value of α, the values of Dev., #Best, and Score described above.

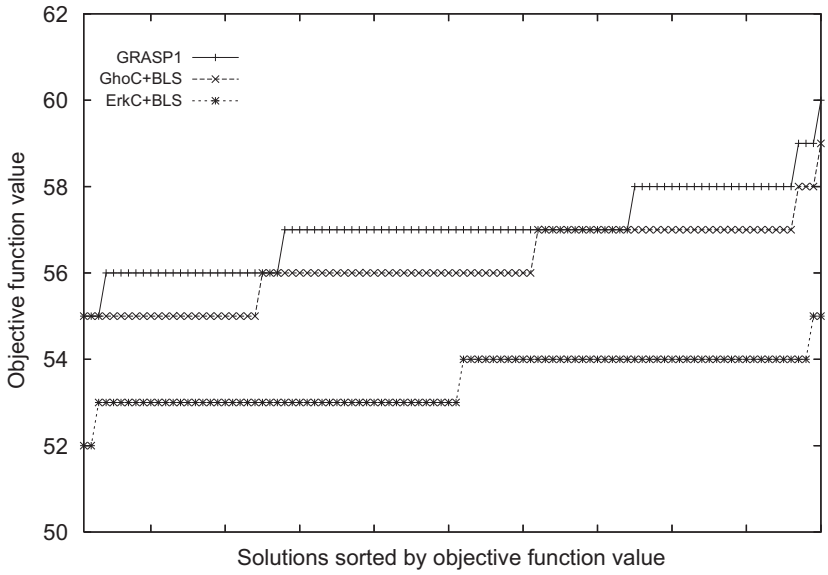


Fig. 7. Solution values with construction + local search.

Table 2
Constructive methods on *Geo* and *Ran* instances with $n = 100, 250$

Method	RanC	TD2	GD2	ErkC	GhoC	GRC2
Dev. (%)	22.85	23.21	17.10	6.08	1.68	0.12
#Best	0	1	0	4	15	37
Score	168	151	117	80	26	4

Table 3
Local search method on *Geo* and *Ran* instances with $n = 100, 250$

Method	ErkC + BLS	GhoC + BLS	GRASP1	GRASP2
Dev. (%)	2.40	0.82	0.24	0.38
#Best	8	22	29	29
Score	81	85	16	22

Table 4
Greedy path relinking methods on *Geo* and *Ran* instances with $n = 100, 250$

	Static GPR				Dynamic GPR			
dth	4	8	10	12	4	8	10	12
Dev. (%)	0.65	0.63	0.54	0.76	0.21	0.32	0.49	0.47
#Best	20	21	24	20	30	29	22	22
Score	86	71	62	85	35	29	69	52
Time (s)	11.0	11.3	10.7	11.0	18.1	18.6	18.3	18.2

The results in Table 1 report that the best outcomes are obtained when the constructive method GRC2 is run with a value of $\beta = 0.90$. Therefore, we use this method in the rest of our experimentation.

In our second preliminary experiment we compare the constructive method GRC2(0.9) with the two previous constructive methods for the MMDP: ErkC [5] and GhoC [7]. We also consider a random construction (RanC) in which the m elements in the solution are randomly selected as a baseline for comparison. In addition, we include in this experiment two previous algorithms developed for the MDP considered to be the best constructive methods for this variant of the problem: TD2 and GD2 [4]. We generate 100 solutions with each method on each instance and report the three statistics described above.

Results in Table 2 report the superiority of the proposed method (GRC2) on these instances. The method is able to obtain 37 best solutions out of 40 instances. Moreover, this experiment confirms that the methods developed for the MDP provide low-quality solutions when employed to solve the MMDP. Specifically, TD2 and GD2 obtain, respectively, 23.21 and 17.10 relative percentage deviation on average, while ErkC, GhoC, and GRC2 obtain 6.08, 1.68, and 0.12, respectively. As expected, RanC obtains low-quality solutions.

In the third preliminary experiment, we compare the constructive with the local search methods for the MMDP. Specifically we target the constructive and improvement methods ErkC + BLS [5] and GhoC + BLS [7]. We consider the two local search methods proposed in Section 3.2, FLS and IMLS in combination with the constructive method GRC2. We denote by GRASP1 the constructive method GRC2(0.9) coupled with the local search FLS, and by GRASP2 the GRC2(0.9) method with IMLS. We construct and improve 100 solutions in each instance with these four methods and report the statistics of the best solutions found in Table 3. We do not report the solution methods for the MDP since, as in the previous experiment, they provide low-quality results.

Table 3 reports that our two new approaches based on the GRASP methodology are able to improve upon previous methods also based on construction plus local search. Specifically, GRASP1 and GRASP2 present an average percent deviation from the best solutions obtained in this experiment of 0.24 and 0.38, respectively, while ErkC + BLS and GhoC + BLS obtain 2.4 and 0.82, respectively. To complement the information presented in Table 3, Fig. 7 shows the values of the 100 solutions obtained with GRASP1, GhoC + BLS, and ErkC + BLS, ordered from lowest to highest, on a *Ran* instance of dimension $n = 250$ and $m = 25$. The results shown in Fig. 7 indicate that GRASP1 systematically obtains better solutions than the other two methods tested. Although, for the sake of simplicity, this figure only shows the results for one instance, we have found that it is representative of the evolution of the tested methods in the entire set.

In the following experiment we compare the two variants of the GPR algorithm described in Section 4.1. We consider both the static version in which the PR is applied after GRASP1 (pseudo-code shown in Fig. 4), and the dynamic version in which the PR is executed within each iteration of GRASP1 (pseudo-code shown in Fig. 5). The PR method depends on the parameter *dth* that specifies the minimum distance for a solution to enter the elite set. Table 4 reports, for the set of 40 instances considered in our preliminary experiments

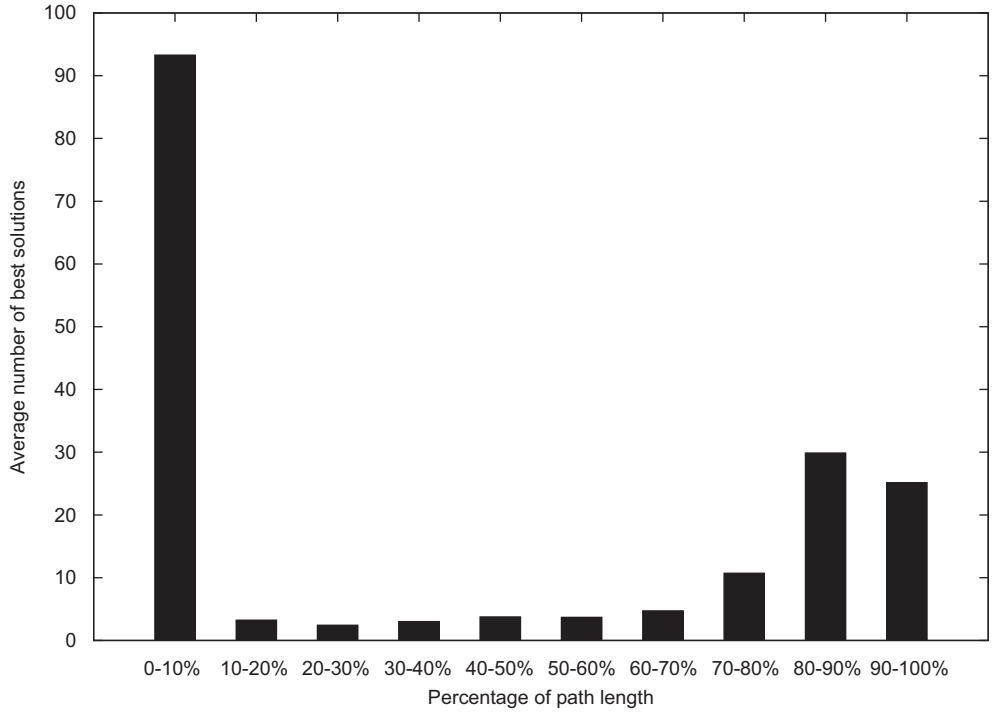


Fig. 8. Average number of best solutions in the path.

Table 5
Truncated GPR and GRPR methods on *Geo* and *Ran* instances with $n = 100, 250$

	GPR				GRPR			
depth	50%	70%	90%	100%	50%	70%	90%	100%
Dev. (%)	0.26	0.23	0.25	0.20	0.37	0.34	0.43	0.58
#Best	31	33	32	34	28	30	28	23
Score	18	20	14	14	39	33	43	91
#Paths	3268.95	3115.10	3026.58	3094.68	3277.18	3142.88	3111.90	3208.15
PR time (s)	0.28	0.30	0.31	0.33	0.26	0.30	0.30	0.30

and four different values of *dth*, the average percentage deviation from the best solution obtained (Dev.), the number of best solutions (#Best), the sum of the *n_score* values (Score), and the average CPU time (Time) in seconds.

Table 4 clearly reports that the GPR in its dynamic variant obtains better solutions than the GPR in the static variant, although it consumes more running time (about 18 s on average compared with the 11 s of the static version). Moreover, this table also reports that the best value of *dth* in the dynamic version is 4, since the method obtains an average percentage deviation of 0.21 and 30 best solutions, which compares favorably with the other values shown. We therefore set the value of *dth* to 4 in the following PR algorithms and restrict our attention to the dynamic variant.

In the next preliminary experiment we undertake to compare the GPR algorithm considered above with the GRPR described in Section 4.2. We first study the effect of the parameter δ in the performance of the GRPR algorithm, considering $\delta = 0.1, 0.3, 0.5, 0.7$, and 0.9. We do not reproduce the results of this experiment in a table, but we simply report that we obtain slightly better solutions with δ set to 0.9 than with the other values (an improvement of 0.2% for the average deviation from the best solutions in this experiment). We then compare the performances of GPR and GRPR with $\delta = 0.9$ both running for 2 min, and consider in this experiment the truncated strategy described in Section 4.3 in which the path is truncated when a portion *depth* of the solutions is explored. When *depth* is set to 100%, the entire path is explored (and therefore no truncation at all is applied). Alternatively, when *depth* is set to 10%, for example, only the first 10% solutions in the path are explored. Table 5 reports the statistics Dev., #Best, Score, as well as the number of paths explored (#Paths) and the average running time in seconds (PR time) that each method dedicates to the PR algorithm.

Table 5 reports that the GPR method provides better solutions than the GRPR, since the average percentage deviation values obtained with the former range from 0.20% to 0.26% while with GRPR these values range from 0.34% to 0.58%. As expected, as the value of the parameter *depth* increases, the time dedicated to the PR (PR time) also increases. Moreover, given that the total running time is set to 2 min in all the cases, the number of explored paths (#Paths) is reduced as *depth* increases. However, these variations (PR time and #Paths) are small since the time saved when the path is truncated is very small in this implementation because the cardinality of the neighborhood explored in the path reduces as the path approaches the guiding solution. Therefore, variations in the parameter *depth* have a small effect on the quality of the final solution obtained with the method.

Fig. 8 complements the information presented in Table 5, plotting the number of best solutions found in each part of the path. The figure shows the average number of best solutions found in the first 10% of the path, the number of best solutions in the second 10% of the path (from 10% to 20%), and so on. The figure confirms the hypothesis that the best solutions are mainly obtained at the beginning of the path. However, good solutions are also obtained in the final part of the path. This fact, together with the small time saving associated with truncated the path lead us to consider in the following experiments the GPR method with *depth* set to 100%.

In our final experiment, we compare our best methods with the state-of-the-art methods for the MMDP. Specifically, we consider

Table 6
Best methods—*Glover* instances

	GhoC + BLS	SA	TS	GRASP1	GPR	GRASP + EvPR
Dev. (%)	0.00	0.00	0.00	0.00	0.00	0.00
#Opt	75	75	75	75	75	75
Score	0	0	0	0	0	0
Time (s)	0.03	0.98	1.56	0.02	0.02	0.04

Table 7
Best methods—*Geo* instances

		GhoC + BLS	SA	TS	GRASP1	GPR	GRASP + EvPR
$n = 100$	Dev. (%)	0.75	0.00	0.00	0.76	0.11	0.09
	#Best	10	19	20	10	16	17
	Score	42	1	0	44	13	8
	Time (s)	2.45	20.96	33.64	0.68	1.68	3.76
$n = 250$	Dev. (%)	1.00	0.68	1.75	1.11	0.19	0.16
	#Best	0	6	2	1	7	14
	Score	65	36	73	71	18	11
	Time (s)	30.50	220.57	439.68	5.58	33.44	65.57
$n = 500$	Dev. (%)	2.36	3.48	9.27	2.39	0.25	0.04
	#Best	0	0	0	0	7	16
	Score	56	62	100	61	13	4
	Time (s)	282.37	1449.85	3633.36	34.99	788.31	1465.44

Table 8
Best methods—*Ran* instances

		GhoC + BLS	SA	TS	GRASP1	GPR	GRASP + EvPR
$n = 100$	Dev. (%)	1.71	2.89	3.28	1.37	0.61	0.49
	#Best	4	9	10	7	14	15
	Score	51	41	44	40	16	9
	Time (s)	1.37	10.82	33.11	0.84	2.96	7.36
$n = 250$	Dev. (%)	2.01	3.73	7.49	1.34	0.81	0.26
	#Best	3	0	0	5	11	17
	Score	34	78	90	20	9	3
	Time (s)	15.98	115.10	430.07	19.22	101.57	271.05
$n = 500$	Dev. (%)	2.95	41.62	41.62	1.70	0.18	0.27
	#Best	13	0	0	14	18	17
	Score	11	80	80	8	2	3
	Time (s)	93.05	868.00	3606.49	99.02	2172.38	6349.20

the following six algorithms (all run for 100 global iterations except GRASP + EvPR):

GhoC + BLS: Multi-start method [7].
SA: Simulated annealing [6].
TS: Tabu search [6].
GRASP1: Constructive method GRC2(0.9) coupled with the local search FLS.
GPR: Dynamic greedy path relinking in which the PR is executed within each iteration of GRASP1 with dth=4 and $depth = 100\%$.
GRASP + EvPR: Evolutionary path relinking with $GlobalIter = 5$ (generation with GPR and evolution with PR of the elite set) and $LocalIter = 20$.

Tables 6–8 report, for each method on each set of instances, the average relative percentage deviation (Dev.) between the best solution values obtained with each method and the best known, the number of instances (#Best) in which the value of the best solution obtained with each method matches the best known, the statistic Score where the lower the Score the better the method, and the average CPU time in seconds. Experiments with Cplex 8.0 (with the Kuo et al. [1] formulation) confirm that the best known solutions in the 75 *Glover* instances reported in Table 6 are the optimal solutions.

The problem instances in the *Glover* set (Table 6) do not provide a way of differentiating the performances of the methods that we are comparing. They are easy to solve and all the methods are capable of quickly finding the optimal solutions.

Tables 7 and 8 present the merit of the proposed procedures. Our GPR and GRASP + EvPR implementations consistently produce the best solutions with percent deviations smaller than those of the competing methods (and with number of best solutions found larger than the others). GRASP + EvPR presents a marginal improvement when compared with GPR but requires longer running times (especially for large instances). On the other hand, the GRASP1 algorithm is able to obtain relatively good solutions in short computational time, with a performance very similar to the GhoC+BLS method. The SA and TS methods perform well on small *Geo* instances (n=100) but are clearly inferior to the others reported in our comparison when target large sized instances ($n = 500$). The ranking of the methods

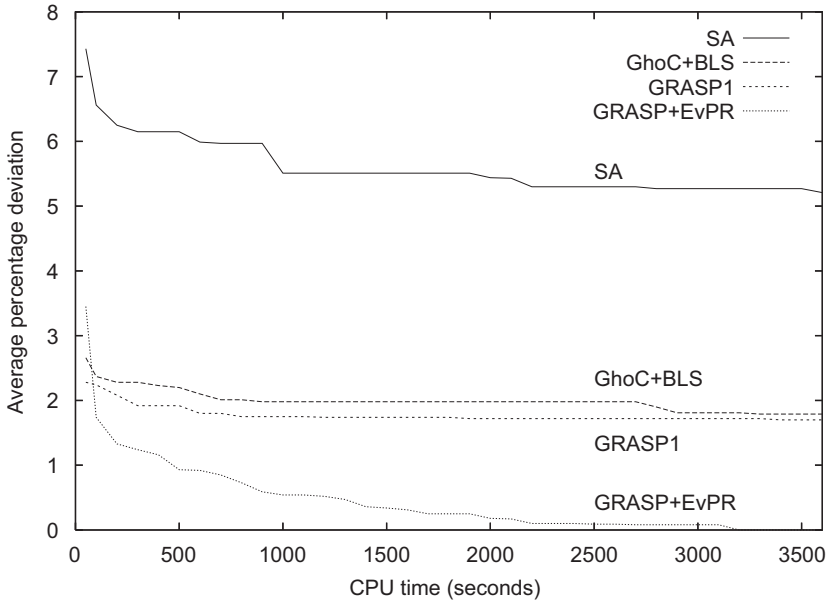


Fig. 9. Search profiles on large *Geo* instances.

with respect to the best solutions found in the 120 *Geo* and *Ran* instances is: GRASP+EvPR(96), GPR(73), GRASP1(37), SA(34), TS(32), and GhoC + BLS(30).

Fig. 9 shows the typical search profile for the methods that we compared. This run corresponds to the largest *Geo* instances ($n=500$, $m=150$) with a time limit of 60 min per instance and method.

Fig. 9 clearly shows that GRASP + EvPR outperforms the other methods over a long term horizon (3600 s in this experiment). Moreover, it is worthwhile noting that GRASP + EvPR obtains high-quality solutions (better than the competing methods) from the first iterations (100 s). On the other hand, SA presents a low performance when comparing with the other three methods in this experiment.

The GRASP1 method obtains the best solutions within the first 50 s. However, GRASP1 by itself (without the PR post-processing) is not able to improve these initial solutions and presents a flat profile during the entire search.

6. Conclusions

The objective of this study has been to advance the current state of knowledge about implementations of path relinking (PR) procedures for combinatorial optimization. Unlike other evolutionary methods such as genetic algorithms or scatter search, PR has not yet been extensively studied.

In this paper, we studied the generation of solutions with GRASP and their combination with PR. We also tested four different variants of PR known as greedy PR, greedy randomized PR, truncated PR, and evolutionary PR, as well as two search strategies: static and dynamic. We performed several experiments with previously reported instances. Our experiments show that the dynamic variants of GRASP with greedy PR and GRASP with evolutionary PR are the best methods for the MMDP instances tested in this paper. Moreover, the results indicate that the proposed hybrid heuristics compare favorably to previous metaheuristics, such as tabu search and simulated annealing.

Obviously, the results that we obtained with our implementation are not all due to the strategies that we wanted to test and that we describe in Section 4. Performance was definitely enhanced by the context-specific methods that we developed for the MMDP. However, our preliminary experiments do show the merit of the mechanisms in Section 4 that we hope could become standard in future PR implementations.

Acknowledgments

This research has been partially supported by the Ministerio de Educación y Ciencia of Spain (Grant Nos. TIN2005-08943-C02-02 and TIN2006-02696), by the *Comunidad de Madrid—Universidad Rey Juan Carlos* project (Ref. URJC-CM-2006-CET-0603).

References

[1] Kuo CC, Glover F, Dhir KS. Analyzing and modeling the maximum diversity problem by zero-one programming. Decision Sciences 1993;24:1171–85.
[2] Glover F, Kuo CC, Dhir KS. Heuristic algorithms for the maximum diversity problem. Journal of Information and Optimization Sciences 1998;19:109–32.
[3] G.C. Silva, L.S. Ochi, S.L. Martins, Experimental comparison of greedy randomized adaptive search procedures for the maximum diversity problem. In: Lecture Notes in Computer Science, vol. 3059; 2004. p. 498–512.
[4] Duarte A, Martí R. Tabu Search and GRASP for the maximum diversity problem. European Journal of Operational Research 2007;178:71–84.
[5] Erkut E. The discrete *p*-dispersion problem. European Journal of Operational Research 1990;46:48–60.
[6] Kincaid RK. Good solutions to discrete noxious location problems via metaheuristics. Annals of Operations Research 1992;40:265–81.
[7] Ghosh JB. Computational aspects of the maximum diversity problem. Operations Research Letters 1996;19:175–81.
[8] Chandrasekaran R, Daughety A. Location on tree networks: *p*-centre and *n*-dispersion problems. Mathematics of Operations Research 1981;6:50–7.
[9] Kuby MJ. Programming models for facility dispersion: the *p*-dispersion and maximum dispersion problems. Geographical Analysis 1987;19:315–29.
[10] Lundi M, Mess A. Convergence of an annealing algorithm. Mathematical Programming 1986;34:111–24.
[11] Feo TA, Resende MGC. A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters 1989;8:67–71.
[12] Feo TA, Resende MGC. Greedy randomized adaptive search procedures. Journal of Global Optimization 1995;6:109–33.
[13] Feo TA, Resende MGC, Smith SH. A greedy randomized adaptive search procedure for maximum independent set. Operations Research 1994;42:860–78.
[14] Resende MGC, Ribeiro CC. Greedy randomized adaptive search procedures. In: Glover F, Kochenberger G, editors. Handbook of metaheuristics. Dordrecht: Kluwer Academic Publishers; 2003. p. 219–50.
[15] Resende MGC, Werneck RF. A hybrid heuristic for the *p*-median problem. Journal of Heuristics 2004;10:59–88.
[16] Prais M, Ribeiro CC. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. INFORMS Journal on Computing 2000;12:164–76.
[17] Glover F. Tabu search and adaptive memory programming—advances, applications, and challenges. In: Barr RS, Helgason RV, Kennington JL, editors. Interfaces in computer science and operations research. Dordrecht: Kluwer Academic Publishers; 1996. p. 1–75.
[18] Glover F, Laguna M. Tabu search. Dordrecht: Kluwer Academic Publishers; 1997.
[19] Laguna M, Martí R. GRASP and path relinking for 2-layer straight line crossing minimization. INFORMS Journal on Computing 1999;11:44–52.
[20] Faria Jr. H, Binato S, Resende MGC, Falcão DJ. Transmission network design by a greedy randomized adaptive path relinking approach. IEEE Transactions on Power Systems 2005;20:43–9.
[21] Andrade DV, Resende MGC. GRASP with evolutionary path-relinking. In: Proceedings of the seventh metaheuristics international conference (MIC 2007); 2007.
[22] Martí R, Laguna M, Glover F. Principles of scatter search. European Journal of Operational Research 2006;169:359–72.
[23] Ribeiro CC, Uchoa E, Werneck RF. A hybrid GRASP with perturbations for the Steiner problem in graphs. INFORMS Journal on Computing 2002;14:228–46.
[24] Laguna M, Martí R. Scatter search: methodology and implementations in C. Dordrecht: Kluwer Academic Publishers; 2003.