IRIS AperTO

UNIVERSITÀ DEGLI STUDI DI TORINO

AperTO - Archivio Istituzionale Open Access dell'Università di Torino

Comparing Local Search Metaheuristics for the Maximum Diversity Problem

This is the author's manuscript

Original Citation:

Availability:

This version is available http://hdl.handle.net/2318/75442 since 2015-12-11T00:41:14Z

Published version:

DOI:10.1057/jors.2010.104

Terms of use:

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

19 April 2024

UNIVERSITÀ DEGLI STUDI DI TORINO

This is an author version of the contribution published on:

R. Aringhieri and R. Cordone.

Comparing Local Search Metaheuristics for the Maximum Diversity Problem.

Journal of the Operational Research Society, 62(2):266-280, 2011.

DOI: 10.1057/jors.2010.104

The definitive version is available at:

http://www.palgrave-journals.com/jors/journal/v62/n2/full/jors2010104a.html

Comparing Local Search Metaheuristics for the Maximum Diversity Problem

Roberto Aringhieri* Roberto Cordone†

April 2, 2010

Abstract

The *Maximum Diversity Problem* (*MDP*) requires to extract a subset M of given cardinality from a set N, maximising the sum of the pairwise diversities between the extracted elements. The *MDP* has recently been the subject of much research and several sophisticated heuristics have been proposed to solve it. The present work compares four local search metaheuristics for the *MDP*, all based on the same Tabu Search procedure, with the aim to identify what additional elements provide the strongest improvement. The four metaheuristics are an Exploring Tabu Search, a Scatter Search, a Variable Neighbourhood Search and a simple Random Restart algorithm. All of them prove competitive with the best algorithms proposed in the literature. Quite surprisingly, the best ones are the simple Random Restart algorithm and a Variable Neighbourhood Search algorithm with an unusual parameter setting, which makes it quite close to random restart. Though this is probably related to the elementary structure of the *MDP*, it also suggests that, more often than expected, simpler algorithms might be better.

1 Introduction

This paper compares a number of local search metaheuristics for the Maximum Diversity Problem (*MDP*) [22]: given a set N of n elements, an integer number m and a matrix $D = \{d_{ij}\}$ expressing the *diversity* between each pair of elements i and j in N, find a collection $M \subset N$ of m elements such that the sum of their pairwise diversities is maximum. Without loss of generality, it is commonly assumed that the diversity matrix is symmetric ($d_{ij} = d_{ji}$ for all $i, j \in N$) and that $d_{ii} = 0$ for all $i \in N$.

The *MDP* can be formulated as the following quadratic programming problem, where $x_i = 1$ when element $i \in N$ belongs to M and $x_i = 0$ otherwise:

*Roberto Aringhieri (roberto.aringhieri@unito.it) Dipartimento di Informatica, Università degli Studi di Torino, Corso Svizzera 185, 10149 Torino, Italy

†Roberto Cordone (roberto.cordone@unimi.it) Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Via Comelico 39/41, 20135 Milano, Italy

$$\text{maximise } z, \text{ where } z = \frac{1}{2} \sum_{i \in N} \sum_{j \in N} d_{ij} x_i x_j \tag{1a}$$

$$\sum_{i \in N} x_i = m \tag{1b}$$

$$x_i \in \{0, 1\} \qquad i \in N \tag{1c}$$

Besides the specific applications of the *MDP*, which are discussed in Section 2, our interest in the problem is motivated by its peculiar combination of computational $\mathcal{NP}$-hardness [30] and extreme simplicity: its feasible solutions are generic subsets, which must only satisfy a cardinality constraint; all elements are equivalent with respect to this constraint and contribute to the objective function by interacting with all other elements. This leaves little foothold to devise problem-specific "tricks", such as *ad hoc* neighbourhoods, the use of unfeasible solutions as a bridge between feasible ones, the identification of specific relevant elements and so on (see, e.g., [34, 19, 20, 16, 1]). In our opinion, these features make the *MDP* an interesting environment to explore the influence of design decisions on performance. As we refer to local search metaheuristics, we mean to compare different general-purpose methodologies exploiting the same fundamental search procedure: in detail, we consider exploring Tabu Search (*XTS*) [13], Scatter Search (*SS*) [32], Variable Neighbourhood Search (*VNS*) [26] and Random Restart (*RR*), all of them based on a common Tabu Search module [25].

Indeed, all the heuristic approaches proposed in the literature for the *MDP* have a basic design much similar to our algorithms, mainly differing for the sophisticated strategies used to periodically reinitialise the search (see Section 2 for details). None of them exploits problem-specific features, thus supporting our previous remark on the difficulty to identify such features.

Section 2 surveys the state of the art algorithms for the *MDP*. Section 3 describes the algorithms here proposed. Section 4 compares their performances to the best published results and to each other. The first comparison aims to prove that these algorithms have been implemented to the best of our capabilities and that they are competitive with the state of the art; the second comparison aims to support the unexpected result that the more sophisticated strategies are dominated by those which periodically restart the search with a random subset of elements chosen trivially from the whole set N or simply excluding those in the current best known solution. The former approach corresponds to random restart, the latter to a degenerate parameter setting of *VNS*. Such a result, though probably due to the simple structure of the *MDP*, poses the following intriguing doubt: on how many "simple" problems a random or nearly random restart might outperform very refined and complex strategies?

2 Survey on the MDP

The *MDP* models several practical problems: when building work teams, student classes and juries it is a common aim to gather individuals with strongly diversified characteristics. Work teams take advantage from including the largest possible range of skills, classes should encourage the exchange between students with different backgrounds, juries should represent the widest variety of points of view existing in a community. Most of the time the number of individuals in a team is fixed, and the diversity between individuals i and j can be expressed by coefficient d_{ij}. Other interesting applications concern project financing [23], data mining [29], exam scheduling, medical treatment, the selection of diversified individuals from a population of solutions in evolutionary algorithms [36], the location of obnoxious facilities [10], and so on.

The *MDP* has been introduced by Glover [22], who proposed two integer linear programming formulations, which can be solved only for small instances ($n \leq 40$) because of the quadratic number of binary variables and the weakness of the continuous relaxation. The quadratic formulation reported in the introduction has been used to solve instances of approximately the same size in [21]. Recent exact approaches are a combinatorial branch-and-bound algorithm [35], which solves all instances up to 50 elements and some Euclidean instances up to 150 elements, and a branch-and-bound algorithm based on semidefinite bounds, which solves instances of approximately the same size, but provides a gap lower than 10% even on instances with $n = 2\,000$ [3].

The size m of the solution influences the computational hardness of the problem: lower ratios m/n yield harder instances, both for the exact and the heuristic algorithms. In fact, as reported in Section 4.1, the benchmark instances proposed in the literature have gradually shifted from larger ratios, with m/n up to 40%, to smaller ones ($m/n = 10\%$).

The first heuristic approaches to the *MDP* are greedy and stingy heuristics [24, 41]. The former start with an empty subset, choose an element at a time with a suitable criteria and add it to the current solution; the latter start with the whole set of elements, choose an element at a time with a suitable criterion and remove it. Both terminate as soon as the correct cardinality is reached.

All following heuristic approaches are based on local search. They nearly all adopt the straightforward neighbourhood provided by the swap of an element $i \in M$ with an element $j \in N \setminus M$, since the cardinality constraint makes it quite difficult to design different neighbourhoods. This neighbourhood includes exactly $m\,(n - m)$ solutions. All of the algorithms restrict the visit to feasible solutions. Notice that any feasible solution may be reached from any other one by repeated swaps, without visiting unfeasible solutions. Moreover, in general the optimal solution for a given value of m is not close to the optimal one with a larger or lower value, so that visiting unfeasible solutions does not appear to give any advantage. Most of these heuristics explore the whole neighbourhood and choose its best solution as the new one. As a consequence, most of the attention focuses on reinitialisation mechanisms of the basic local search.

The earlier algorithms adopt the Greedy Randomized Adaptive Search Procedure methodology (*GRASP*) [17]. They apply very sophisticated methods to build high quality randomised starting solutions and perform the subsequent improvement phase by a standard local search technique. Ghosh [21] proposes a greedy procedure based on a lower and an upper estimate of the contribution brought by each element to the solution. Andrade et al. [2] use a restricted candidate list and select the first and the last half of the elements with two different greedy criteria. Silva et al. [40] introduce a reactive mechanism to tune the *GRASP* parameters on the basis of the solutions found; one of their algorithms adopts a different neighbourhood, though similar, swapping two pairs of elements at a time. A hybrid *GRASP* with Data Mining, is proposed in [38]; this is based on the identification of good "patterns", i. e., subsets of elements frequently appearing in good solutions. A GRASP with path relinking is described in [39]. The instances solved range from 40 to 500 elements and the larger ones require hours of computing time.

A second group of local search metaheuristics follow the Tabu Search framework. These solve instances from 500 to 2 000 elements in a time ranging from seconds to few minutes. The *Tabu_D-2* algorithm, proposed in [15], initialises the search with a stingy heuristic introduced in [24], but modified to take into account the average value and the number of all previously built solutions including each element. This adaptive mechanism has the purpose to guide the search towards regions surrounding good known solutions and away from the already visited solutions. *Tabu_D-2* explores a subset of the standard neighbourhood, including only $n - m$ solutions: in fact, for each element potentially added to the solution, only one element is tested for removal. Moreover, *Tabu_D-2* implements the first non tabu improving move found, instead of the best. The eXploring Tabu Search algorithm *XTS-MDP* described in [7] initialises the search with a simple greedy criteria. Contrary to *Tabu_D-2*, it adopts a sophisticated Tabu Search mechanism with two lists, adaptively tuned tenures (getting shorter when the solution improves, longer when it worsens) and a periodic reinitialisation based on a list of promising solutions found during the search.

The *Iterated Tabu Search* algorithm (*ITS*) proposed in [36] first turns the fractional solution $x_i = m/n$ for all $i \in N$ into a feasible integer one with a steepest ascent algorithm based on the mathematical programming formulation (1). Then, it applies a Tabu Search algorithm with a single tabu list, which forbids to revert the more recent swaps. The search restarts periodically by performing a small random number of swaps (at most $0.1n$), chosen at random from a short list which includes the $5 - 10$ most improving ones.

The Scatter Search algorithm presented in [18] applies an improved version of *Tabu_D-2* with two tabu lists instead of a single one, in order to build a *reference set* made up of best solutions and diverse solutions. The latter are defined as solutions strongly different from each other and from the best ones, where the difference between two solutions is measured by the number of elements belonging to only one of them. Periodically, all pairs of solutions in the reference set are combined to provide new starting solutions for *Tabu_D-2*.

Finally, the Variable Neighbourhood Search described in [10] initialises the

search with a random subset of elements and applies pure local search, adopting the standard neighbourhood and always accepting the first improving swap found. After reaching a local optimum, it builds a new starting solution by swapping k pairs of elements in the best known solution. The "shaking" parameter k is set to 1 at the beginning and it is increased by 1 at each restart. If k exceeds $\min(m, n-m)$ or if the best known solution improves, k is set back to 1.

3 Four local search metaheuristics for the *MDP*

Our experience on the *MDP* started with the implementation of a simple Tabu Search procedure, which is the basic local search module embedded into the algorithms described in the following. We chose Tabu Search because most previous algorithm adopted a very refined construction phase followed by an elementary improvement phase, consisting of a standard local search. On the other side, the fact that any feasible solution may be reached from any other one by repeated swaps suggested that a local search algorithm might be an effective solving approach, provided that it were refined enough to cope with the large number of local optima characterising the *MDP*.

Section 3.1 introduces some notation and the common elements of these algorithms: the greedy algorithm used to find the starting solution and the basic Tabu Search procedure. The differentiating elements of each approach are presented in Section 3.2 for *XTS*, Section 3.4 for *SS*, Section 3.3 for *VNS* and Section 3.5 for *RR*.

3.1 Greedy initialisation and basic Tabu Search

Given a subset $M \subset N$, a relevant role in all algorithms is played by coefficients $D_i = \sum_{j \in M} d_{ij}$ for $i \in N$, which can be interpreted as the (actual or potential) contribution of element i to the objective function. In fact, $z = \frac{1}{2} \sum_{i \in M} D_i$.

Greedy algorithm All the algorithms proposed build a feasible solution starting from the pair of elements (i, j) with the largest diversity d_{ij}. Let $M^{(0)} = \{i, j\} \subset N$ and $z^{(0)} = d_{ij}$. At each step, add the element with the largest diversity with respect to the already chosen ones: $k^{(h)} = \arg\max_{i \in N \setminus M^{(h-1)}} D_i$, so that $M^{(h)} = M^{(h-1)} \cup \left\{k^{(h)}\right\}$ and $z^{(h)} = z^{(h-1)} + D_{k^{(h)}}$. After the addition, all coefficients D_i can be easily updated by summing the value $d_{ik^{(h)}}$. Since each step requires $O(n)$ time, the overall procedure takes $O(mn)$.

Tabu Search Starting from a given feasible solution M of value z, the basic Tabu Search procedure tries to produce an improved solution M^{new} by exchanging a single element s in the current solution with a single element t out of it, so that $M^{new} = M \cup \{t\} \setminus \{s\}$. It is possible to efficiently evaluate the value z^{new} of M^{new} by simply subtracting from z the total contribution of the old

element s, that is D_s, and adding the total contribution of the new element t, that is $D_t - d_{st}$. More formally, $z^{new} = z - D_s + D_t - d_{st}$. The neighbourhood is completely explored before choosing one of its elements as the current solution. As each move is evaluated in constant time, a complete exploration of the neighbourhood takes $O(mn)$ time.

To avoid looping over already visited solutions, the procedure exploits a finite-length list of forbidden moves, named tabu list [25]. The distinguishing element of this Tabu Search with respect to the previous literature is the definition of two independent tabu lists, in order to avoid both the inclusion of a recently removed element and the removal of a recently included element [7]. This feature has been independently adopted by other recent algorithms [18]. To implement it, we store for each element $i \in N$ the last iteration in which i was swapped in or out of the solution. A move (s, t) is tabu, i. e., forbidden, when the outgoing element s has entered the solution less than ℓ_{out} iterations ago or the ingoing element t has exited less than ℓ_{in} iterations ago. The length of the prohibition (*tabu tenure*) is larger for the insertion than for the removal, in order to guarantee that the two tabus have a comparable strength. In fact, since m ranges from $0.1n$ to $0.4n$ in all benchmark instances, the number of elements out of the solution is from 1.5 to 9 times larger than the number of elements inside it. Correspondingly, the former elements have on average a lower probability to be involved in each move.

At each step, the procedure performs the non tabu move which yields the largest z^{new}. However, if the move providing the largest z^{new} improves the best known solution, it is performed even if it is tabu (*aspiration criteria*). After applying such a move, the value of D_i is updated as follows for all $i \in N$: $D_i = D_i - d_{is} + d_{it}$.

3.2 An eXploring Tabu Search algorithm

We extend the basic Tabu Search by adding a short-term and a long-term memory mechanism. The resulting algorithm is denoted as *XTS-MDP* [7].

Short term memory

The short-term memory mechanism tunes the two tabu tenures depending on the recent results of the search: it decreases them after T_i consecutive improving iterations and increases them after T_w consecutive worsening iterations. The purpose of the decrease is to intensify the search in those regions which provide improving solutions, and therefore appear more promising. The purpose of the increase is to speed up the departure from those regions which provide worsening solutions, and therefore probably surround an already visited local optimum.

Of course, the decrease and the increase rate must be tuned to avoid both over and under-reacting to the variations of the objective function. In detail, the tabu tenure ℓ_{in} varies in $\left[\ell_{in}^m, \ell_{in}^M\right]$: at the beginning of the algorithm, it is set to the middle point of this interval $\ell_{in}^{(0)} = \left(\ell_{in}^m + \ell_{in}^M\right)/2$; after T_w consecutive worsening iterations, it increases by $\Delta\ell_{in}$, whilst it decreases by $\Delta\ell_{in}$ after T_i

consecutive improving iterations. The same occurs for ℓ_{out}, which starts from $\ell_{out}^{(0)} = \left(\ell_{out}^{m} + \ell_{out}^{M}\right)/2$ and varies in $\left[\ell_{out}^{m}, \ell_{out}^{M}\right]$ by steps equal to $\Delta\ell_{out}$. For the specific values adopted in our computational experiments, see Section 4.2.

While the use of a variable tabu tenure is quite common in the literature, the amount $\Delta\ell$ is commonly fixed to 1. Our experience is that, when the tabu tenure reaches the maximum value of its range, thus pushing the search away from the current region of the solution space, the number of improving iterations is often insufficient to reduce it in order to intensify the search in the newly reached region. A similar behaviour can be observed when the tabu tenure decreases down to its minimum value: the number of worsening iterations is insufficient to increase it enough to diversify the search. To counterbalance this effect, we adopt a variable self-adapting variation step $\Delta\ell$, which becomes larger when the tabu tenure approaches the lower or the upper limit of its range. Once again, for the specific values adopted see Section 4.2.

Long term memory

The long-term memory mechanism allows the search to stop exploring unpromising regions of the solution space. This mechanism is known as eXploring Tabu Search [13]. Its basic idea is to focus on the *second-best* solution computed during each neighbourhood exploration and to maintain a set of the best ones. These solutions, which are not used by standard Tabu Search, could on the contrary be good starting points to guide the search toward promising regions. Therefore, under suitable conditions, the search restarts from one of them.

To implement this mechanism, when exploring the neighbourhood of M, its best solution M' becomes the incumbent, while the second-best solution M'' is inserted into a list $\mathcal{M}$. This list also saves the tabu lists and all the values of the parameters for the short-term mechanism which would result by choosing M'' as the incumbent. The purpose is to exactly reproduce the whole state of the computation, so that, if necessary, the algorithm might backtrack to this point and proceed as if a different choice had been made. List $\mathcal{M}$ has a limited length: when that length is exceeded, its worst element is removed. Duplicate solutions are removed, as well.

The search restarts from the best solution in $\mathcal{M}$ every time any of the following conditions is verified: either the best known solution does not improve for I^* iterations or the length of one of the two tabu lists resides in the upper half of its range, that is $\left[\left(\ell^m + \ell^M\right)/2; \ell^M\right]$, for I_{up} consecutive iterations. The first condition suggests that the currently explored region is not promising, the second one that the short term mechanism is insufficient to diversify the search. When any of these conditions holds, the best solution is extracted from $\mathcal{M}$, it becomes the new incumbent and the corresponding state of the computation is fully retrieved. The whole algorithm terminates after a limit number I of neighbourhood explorations.

The pseudocode in Figure 1 summarises Algorithm *XTS-MDP*, denoting as $z(M)$ the value of solution M. The greedy procedure builds a starting best

```
XTS-MDP(N, m, d, ℓ_in^m, ℓ_in^M, ℓ_out^m, ℓ_out^M, I*, I_up, I)
M* := Greedy(N, m, d); M := {M*};
L_i := −∞ for all i ∈ N;                                  {Initialise the tabu list}
ℓ_in := (ℓ_in^m + ℓ_in^M)/2; ℓ_out := (ℓ_out^m + ℓ_out^M)/2;
T_i := 0; T_w := 0; i := 0; i* := 0; i_up := 0;
while i ≤ I do
  M := ExtractBest(M);
  {Short-term Tabu Search procedure}
  while i ≤ I and i ≤ i* + I* and i_up ≤ I_up do
    (M', M'') := FindBestNeighbour(M, N, m, d, L, ℓ_in, ℓ_out, i, z(M*));
    {Update the number of consecutive improvements or worsenings}
    if z(M') > z(M) then T_i := T_i + 1; T_w := 0; end if
    if z(M') < z(M) then T_i := 0; T_w := T_w + 1; end if
    {Update the tabu tenures}
    (ℓ_in, ℓ_out) := UpdateTenures(ℓ_in, ℓ_out, T_i, T_w, ℓ_in^m, ℓ_in^M, ℓ_out^m, ℓ_out^M);
    {Update the number of iterations with longer tabu tenures}
    if ℓ_in ≥ (ℓ_in^m + ℓ_in^M)/2  or  ℓ_out ≥ (ℓ_out^m + ℓ_out^M)/2 then
      i_up := i_up + 1  else  i_up := 0;
    end if
    {Update the best known solution}
    if z(M') > z(M*) then M* := M'; i* := i; end if
    {Update the current solution}
    M := M';
    {Update the list of the second-best solutions}
    M̄ := ExtractWorst(M);
    if z(M'') > z(M̄)  and  M'' ∉ M then
      M := M ∪ {M''}
    else
      M := M ∪ {M̄}
    end if
    i := i + 1;
  end while
end while
return M*;
```

Figure 1: Pseudocode of Algorithm *XTS-MDP*.

known solution M^*, which also becomes the only candidate starting solution of list $\mathcal{M}$. All parameters are initialised, and in particular the iteration L_i of the last swap for each element i is set to $-\infty$ in order to label all moves as non tabu. Then, we draw from $\mathcal{M}$ the best solution M to initialise the short-term Tabu Search procedure, which will terminate when reaching the maximum total number of iterations or when satisfying either of the two conditions discussed above. At each step, the Tabu Search explores the neighbourhood of the current solution M and returns its best and second-best solutions which are non tabu or satisfy the aspiration criteria, respectively denoted as M' and M''. Solution M' determines the update of the number of improving or worsening consecutive iterations, of the tabu tenures and of the number i_{up} of consecutive iterations

with a long tabu tenure. It possibly updates the best known solution M^* and replaces the current one M. Solution M'' possibly updates the list $\mathcal{M}$ of candidate starting solutions.

Notice that the basic Tabu Search can be easily obtained from *XTS-MDP* by keeping the tabu tenures to fixed values ($\ell^m_{\text{in}} = \ell^M_{\text{in}}$ and $\ell^m_{\text{out}} = \ell^M_{\text{out}}$) and enabling only the global termination condition ($I^* = I_{up} = I$). This allows to ignore most of the update operations.

Our computational experience with Algorithm *XTS-MDP*, though impressive with respect to the previously published algorithms [7], revealed the existence of some hard instances, which could not be solved to the optimum with the standard parameter setting, even in a huge number of iterations, but required specific settings. A certain degree of randomisation often improves the robustness of an algorithm, as it avoids the deterministic repetition of mistakes forbidding to reach the optimum [28]. This led us to experiment with alternative approaches in which randomisation plays a role.

3.3 A Variable Neighbourhood Search algorithm

The *VNS* approach systematically exploits the idea of neighbourhood change, to overcome the limits posed by the use of a single neighbourhood. It requires a hierarchy of size-increasing neighbourhoods. Our implementation, denoted as *VNS-MDP*, defines the k-th neighbourhood $N_k(M)$ of solution M as the set of all solutions obtained from M by replacing k elements in the solution with k elements out of it. An equivalent definition would be

$$N_k(M) = \{M' \subset N : \rho(M, M') = k\}$$

where function $\rho(M, M') = |M \setminus M'|$ expresses the distance between solutions M and M' as the number of elements included in one and not in the other. Notice that, since M and M' have the same cardinality, function ρ is symmetric: $\rho(M, M') = \rho(M', M)$. Also notice that larger values of k correspond to larger neighbourhoods and that, though these neighbourhoods are clearly related to one another, they are all pairwise disjoint, since swapping a larger number of elements simultaneously cannot yield the same solutions obtained by swapping a smaller number.

A pseudocode of *VNS-MDP* is given in Figure 2. At the beginning, the greedy procedure described in Section 3.1 generates a starting best known solution M^*, which is also used as the current one M, and the neighbourhood index k is set to its minimum value $k_{\min}$. Starting from M, the basic Tabu Search runs with fixed tabu tenures for I_{TS} iterations, and returns the best solution $\tilde{M}$ found in the current run. Then, if $\tilde{M}$ is judged more promising than M^* as a starting solution, index k is set back to $k_{\min}$ and procedure *Shaking* selects a new current solution M from its k-th neighbourhood; otherwise, k increases and procedure *Shaking* selects a new current solution M from the k-th neighbourhood of M^*. In the end, $\tilde{M}$ possibly updates the best known solution M^*. When index k exceeds a threshold $k_{\max}$, it is forced back to $k_{\min}$. The algorithm

terminates after a given total number of iterations I. Several variants of this algorithm have been tested by tuning its numerical parameters and by suitably defining procedures *PromisingStartingSolution* and *Shaking*.

VNS-MDP$(N, m, d, \ell_{\text{in}}, \ell_{\text{out}}, I_{TS}, \alpha, I)$
$M^* :=$ Greedy(N, m, d); $i := 0$; $M := M^*$;
while $i \leq I$ **do**
  $k := k_{\min}$;
  while $i \leq I$ **and** $k \leq k_{\max}$ **do**
    $\tilde{M} :=$ TabuSearch$(M, N, m, d, \ell_{\text{in}}, \ell_{\text{out}}, I_{TS})$;
    {Update the starting solution}
    if PromisingStartingSolution$(\tilde{M}, M^*, \alpha)$ **then**
      $k := k_{\min}$; $M :=$ Shaking$(\tilde{M}, k)$;
    else
      $k := k + \delta k$; $M :=$ Shaking(M^*, k);
    end if
    {Update the best known solution}
    if $z(\tilde{M}) > z(M^*)$ **then** $M^* := \tilde{M}$; **end if**
    $i := i + I_{TS}$;
  end while
end while
return M^*;

Figure 2: Pseudocode of Algorithm *VNS-MDP*.

Variants of Algorithm *VNS-MDP*

In a first phase, we considered the three versions of *VNS* denoted in the literature as *basic VNS*, *skewed VNS* and *guided VNS* [27]. All of these algorithms set $k_{\min}$ and δk to 1 for the sake of simplicity, and tune $k_{\max}$ by experiment in a range of small values. These are common assumptions in the literature of *VNS*, based on the idea that the restart solution M should be close to a promising solution, either $\tilde{M}$ or M^*, suggested by local search.

The *basic VNS* implements procedure *Shaking* by swapping k elements in the solution and k elements out of it, chosen at random with a uniform distribution. Procedure *PromisingStartingSolution* simply checks condition $z(\tilde{M}) > z(M^*)$. Thus, if the current run of the Tabu Search improves the best known solution, k gets back to 1 and $\tilde{M}$ is selected; in other words, the search restarts from a solution very close to the new best known one. If it fails, k increases and M^* is selected: the search restarts from a solution neighbour to the old best known one, but farther and farther away as the failure repeats.

The *skewed VNS* applies the same shaking procedure used by the basic *VNS*, but it restarts from $\tilde{M}$ also in case of failures, provided that $\tilde{M}$ is sufficiently distant from M^* to avoid falling back into it. This is obtained by implementing procedure *PromisingStartingSolution* with the more complex and weaker condition:

$$z(\tilde{M}) + \alpha\rho(\tilde{M}, M^*) > z(M^*)$$

where $\alpha > 0$. If this condition fails, not only $\tilde{M}$ is worse than M^*, because $z(\tilde{M}) \leq z(M^*)$, but $\tilde{M}$ is also close to M^*, because $\rho(\tilde{M}, M^*)$ is small. In other words, solution $\tilde{M}$ is rejected only when it is neither improving nor diversifying. The rationale of this approach is that, since Tabu Search is more effective than standard local search as a basic procedure, the local optima closer to the best known result have probably been already explored and therefore it might be more effective to restart from solutions farther away.

The *guided VNS* adopts the simple condition $z(\tilde{M}) > z(M^*)$ used by the basic *VNS*, but its *Shaking* procedure is deterministic and based on the number of iterations during which each element i has been included in the current solution (*frequency*). The procedure removes the k elements of the solution with the highest frequencies and replaces them with the k elements out of the solution with the lowest frequencies. The rationale of this approach is to focus on diversification by forcing into the initial solution the elements which have been less frequently considered so far. It is also an attempt to use deterministic adaptive tools instead of simple randomisation.

The very good performance of the *VNS* algorithm proposed in [10], published when we were completing these experiments, led us to perform some additional analysis. This algorithm, in fact, adopts the unusual setting $k_{\max} = \min(m, n-m)$, which is the maximum possible value, much larger than the ones tested in our first experiments and commonly used in the literature. Such a value makes the restart mechanism gradually evolve from a random generation in the standard neighbourhood to a nearly random generation in the whole solution space. The difference is that, when $k = k_{\max}$, the shaking procedure removes all current elements, whereas a pure random restart would probably keep some of them. With such a parameter setting, the common sense assumption that the restart solutions should be close to the best known one is satisfied only by a few first restarts. To pursue this line of research, we introduced other two variants of *VNS*.

The *accelerated VNS* modifies the basic *VNS* by setting $k_{\max} = \min(m, n-m)$. Moreover, the use of Tabu Search as the core search procedure makes restarts much less frequent in our algorithm, so that the increase of k would be much slower. To obtain a comparable behaviour, we decided to correspondingly increase δk and $k_{\min}$.

In the end, the *random VNS* sets $k_{\min} = k_{\max} = \min(m, n-m)$, thus always restarting the search in the largest neighbourhood, instead of gradually enlarging the neighbourhood used. This approach corresponds to a nearly random restart, which only forbids the elements belonging to the current best known solution. The rationale of this approach is to specifically test whether the use of the largest neighbourhood might explain the effectiveness of the *VNS* algorithm proposed in [10].

The numerical values assigned to all parameters are given in Section 4.2.

3.4 A Scatter Search algorithm

Scatter Search is an evolutionary method which uses strategies for search intensification and diversification [32]. A pseudocode of our Scatter Search algorithm, *SS-MDP*, is given in Figure 3.

SS-MDP$(N, m, d, \ell_{\text{in}}, \ell_{\text{out}}, I_{TS}, I)$
$(M^*, R, i) :=$ GenerateReferenceSet(N, m, d);
while $i \leq I$ **do**
  $R' := R$;
  for each $(M_1, M_2) \in R'$ **do**
    $M :=$ CombineSolutions(M_1, M_2);
    $\tilde{M} :=$ TabuSearch$(M, N, m, d, \ell_{\text{in}}, \ell_{\text{out}}, I_{TS})$;
    {Update the reference set}
    $R :=$ UpdateReferenceSet$(R, \tilde{M})$;
    {Update the best known solution}
    if $z(\tilde{M}) > z(M^*)$ **then** $M^* := \tilde{M}$; **end if**
    $i := i + I_{TS}$;
  end for
  if $R' = R$ **then** $(R, i) :=$ ModifyReferenceSet(R, i, N, m, d); **end if**
end while
return M^*;

Figure 3: Pseudocode of Algorithm *SS-MDP*.

The algorithm first generates a set of solutions R, named *reference set*, by suitably applying, as explained in the following, the greedy procedure and the basic Tabu Search procedure. The reference set is composed of two subsets B and D: B contains the best solutions found, whereas D contains those solutions which have the largest sum of distances with respect to the other solutions in R. The distance function ρ is the same used in *VNS-MDP* and defined in Section 3.3. The reference set evolves by combining each pair of its members to obtain a new starting solution M: the combination of two solutions in B intensifies the search, while the diversification is given by combining two solutions in D or one in B and one in D. Starting from M, the basic Tabu Search procedure returns an improved solution $\tilde{M}$, which might enter either the best or the diverse subset of the reference set and which might update the best known solution. Notice that the combinations are always performed on the original reference set, which is replaced by the new one only in the end. If the reference set is unchanged by a whole loop of combinations, the diverse subset D must be regenerated. The algorithm terminates after a given total number of iterations I.

Management of the reference set

The initial reference set is generated repeating the following procedure until R reaches the desired size. First, the greedy procedure selects at random a pair of elements and builds a solution starting from them. The used pair is marked

to avoid selecting it again. Then, the basic Tabu Search procedure with fixed tabu tenures returns an improved solution. We try to insert this solution first in B, then in D, provided that it is not a duplicate. If B has reached its desired size and the solution is worse than all the ones in B, it is rejected. As well, it is rejected from D when D has reached its desired size and the solution is not different enough from the ones in the reference set. This means that the sum of the distances between the solution and all current elements of R is lower than that between each of those elements and the other ones. If both insertions fail, the solution is neither good nor different enough from the reference set. Then, it is randomly modified in order to improve the diversification, by exchanging the elements which have so far belonged more frequently to a solution with the less frequent ones. The resulting solution is once again tested for insertion into R.

The update of R is based on the combination of two solutions M_1 and M_2 drawn from R into a new single one M. Although it would be possible to combine three or more solutions, the combination of pairs usually proves sufficient to obtain good results while keeping a reasonable running time [31]. Let $D_i^{(1)}$ and $D_i^{(2)}$ denote the D_i coefficients for solutions M_1 and M_2. We compute the auxiliary coefficients:

$$\bar{D}_i = \begin{cases} D_i^{(1)} + D_i^{(2)} & \text{if } i \in M^{(1)} \text{ and } i \in M^{(2)} \\ D_i^{(1)} & \text{if } i \in M^{(1)} \text{ and } i \in N \setminus M^{(2)} \\ D_i^{(2)} & \text{if } i \in N \setminus M^{(1)} \text{ and } i \in M^{(2)} \\ 0 & \text{if } i \in N \setminus M^{(1)} \text{ and } i \in N \setminus M^{(2)} \end{cases}$$

and we build the new initial solution M by selecting the m elements with the largest values of $\bar{D}_i$.

Notice that the reference set R in general is only partly modified by the combination phase, because each combined solution might be accepted or not. For the sake of efficiency, the first combination phase involves all pairs of solutions, whereas the following ones involve only the pairs with at least one new solution.

3.5 A random restart algorithm

Our random restart algorithm *RR-MDP* (see Figure 4) builds a starting solution with the greedy procedure and improves it with I_{TS} iterations of the basic Tabu Search procedure, with fixed tenures. As long as the limit number I of neighbourhood explorations has not yet been reached, the algorithm restarts periodically the search by simply building a new current solution M with m elements chosen at random and improving it with the basic Tabu Search.

Algorithm *RR-MDP* derives by an experimental application to the *MDP* of the *Ant Colony Optimization* approach [14]. Our preliminary results were quite encouraging, as the algorithm performed remarkably well on all benchmark instances [6]. While tuning its parameters, however, we realised that the best parameter set corresponded to a very high evaporation rate, that is to a pure random restart. The direct implementation of this algorithm confirmed this unexpected remark.

RR-MDP$(n, m, d, \ell_{\text{in}}, \ell_{\text{out}}, I_{TS}, I)$
$M :=$ Greedy(N, m, d);
$M^* :=$ TabuSearch$(M, N, m, d, \ell_{\text{in}}, \ell_{\text{out}}, I_{TS})$;
$i := I_{TS}$
while $i \leq I$ **do**
  $M :=$ RandomRestart(N, m);
  $\tilde{M} :=$ TabuSearch$(M, N, m, d, \ell_{\text{in}}, \ell_{\text{out}}, I_{TS})$;
  {Update the best known solution}
  if $z(\tilde{M}) > z(M^*)$ **then** $M^* := \tilde{M}$; **end if**
  $i := i + I_{TS}$;
end while
return M^*;

Figure 4: Pseudocode of Algorithm *RR-MDP*.

4 Computational results

In this section we discuss the computational results of the proposed algorithms. First, we introduce the computational environment and the benchmark instances (Section 4.1). Then, we introduce the specific values used for their parameters. Then, we compare the results of the algorithms to each other, in order to draw some conclusions about the most effective design decisions, and to the best known results, in order to show that they are all competitive with the state of the art. Note that, in the following, when we refer to best known result we always consider the one obtained from the literature.

4.1 Computational environment

All the algorithms proposed are coded in C standard and compiled by *gcc*. They run on a 2.2 GHz 2 Dual Core AMD Opteron(tm) Processor 275, with 3 GB of memory.

For our experiments, we have drawn from the literature five sets of benchmark instances, which are all available at website `http://www.di.unito.it/~aringhie/benchmarks.html`:

- benchmark APOM [2] includes 40 instances with n ranging from 50 to 250 and m from $0.2n$ to $0.4n$, subdivided into:
  - type A: the elements are points on a plane, with random coordinates extracted from $[1, 9]$; the d_{ij} values are Euclidean distances;
  - type B: all d_{ij} values are random integers uniformly drawn from $[1, 9999]$;
  - type C: the values are random integers; half of them are uniformly distributed in $[1, 9999]$, the other half in $[1, 4999]$;
  - type D: the values are random integers; half of them are uniformly distributed in $[1, 9999]$, the other half in $[5000, 9999]$;

- benchmark SOM [40], consists of 20 instances with n ranging from 100 to 500 and m from $0.1n$ to $0.4n$; the d_{ij} coefficients are random integers uniformly distributed in $[0, 9]$;
- benchmark DM1 [15], consists of 60 instances with real numbers generated from a $[0, 10]$ uniform distribution; 20 instances have $n = 500$ and $m = 200$ (DM1A), 20 instances have $n = 2000$ and $m = 200$ (DM1B), 20 instances have $n = 500$ and $m = 50$ (DM1C);
- benchmark DM2 [15], consists of 20 instances with real numbers generated from a $[0, 1000]$ uniform distribution; they have $n = 500$ and $m = 50$
- benchmark GKD [15] consists of 20 instances with $n = 500$ and $m = 50$, in which the elements are points in a 10-dimensional space with coordinates randomly generated in $[0, 10]$; the d_{ij} values are Euclidean distances.

4.2 Parameter tuning

In this section, we briefly list the values adopted for the parameters of the proposed algorithms, which were obtained in a preliminary experimental campaign. It must be remarked that the range of values tested for most of these parameters was suggested by common sense and base on our experience [12, 8, 9, 4]: a small number of alternative settings were investigated and the ones with the best average performance were applied to all benchmark instances, without any fine tuning to improve the results on specific classes of instances.

With the explicit purpose to guarantee a fair comparison among the alternative approaches, the basic Tabu Search procedure embedded in all algorithms adopts the same parameters; in the case of *XTS-MDP*, which applies variable tabu tenures, instead of fixed ones, the range of the variable tenure is centred on the value of the fixed one.

To generate pseudo-random numbers for all randomised algorithms, we have tested both the classical *ran1* generator described in [37] and the *RngStream* generator proposed in [33]. Since the results only exhibited slight differences, we decided to use the more recent generator, *RngStream*. This has been initialised with its default predefined random seed, in order to obtain a deterministic version of the algorithm. A preliminary test, consisting of 10 runs with different random seeds, confirmed that the differences are statistically insignificant according to *Wilcoxon's matched-pairs signed-ranks test* [42], even when considering the two runs with the best and the worst average performance.

In order to compare all the approaches in the fairest way, we have imposed the same number $I = 100\,000$ of basic iterations (i. e., neighbourhood explorations) to all of them. This value is largely sufficient to obtain the best known result on most of the benchmark instances, while requiring a reasonable computational time.

XTS-MDP In our computational experiments, the tabu tenure ℓ_{in} varies between $\ell_{in}^m = 8$ and $\ell_{in}^M = 14$, while ℓ_{out} varies between $\ell_{out}^m = 3$ and $\ell_{out}^M = 7$.

These values were obtained after some tuning in the range between 3 and 20, by keeping into account, as observed in Section 3.1, that the tabu tenure for insertion should be larger than that for removal, because the elements out of the solution exceed those inside. The starting values for the tenures are trivially set to the middle point of the corresponding range: $\ell_{in}^{(0)} = 11$ and $\ell_{out}^{(0)} = 5$. Both tenures are updated after $T_i = 3$ improving iterations or $T_w = 5$ worsening iterations with the following variable step:

$$\Delta\ell_{in} = \begin{cases} 2 & \text{if } \ell_{in} = \ell_{in}^m \text{ or } \ell = \ell_{in}^M \\ 1 & \text{if } \ell_{in}^m < \ell_{in} < \ell_{out}^M \end{cases} \text{ and } \Delta\ell_{out} = \begin{cases} 2 & \text{if } \ell_{out} = \ell_{out}^m \text{ or } \ell = \ell_{out}^M \\ 1 & \text{if } \ell_{out}^m < \ell_{out} < \ell_{out}^M \end{cases}.$$

which simply corresponds to a larger update when the tenure hits an extreme value of its range. Finally, the parameters tuning the long-term exploring restart mechanism are $|\mathcal{M}| = 15$, $I^* = 1\,000$ and $I_{up} = 300$. With few exceptions, most of the restarts are triggered by parameter I^*, so that each of them performs at least 1 000 iterations.

VNS-MDP The two tabu tenures of the basic Tabu Search have been fixed to $\ell_{in} = 11$ and $\ell_{out} = 5$, that are the starting values and the middle points of the ranges adopted in *XTS-MDP*. The number of basic Tabu Search iterations has been set to $I_{TS} = 2\,000$ for all variants of *VNS-MDP*, because larger values excessively limited the number of restarts, thus reducing *VNS* to the basic Tabu Search procedure. On the other side, smaller values yielded consistently dominated results according to Wilcoxon's test, probably because they terminate prematurely the search while it is still vital. Moreover, the same value of $I_{TS} = 2\,000$ has been adopted for *RR-MDP*, which exhibited a similar dependency, and we have already remarked that each restart of *XTS-MDP* terminates after I^* non-improving iterations, that is at least 1 000 overall iterations. Notice that the literature on *VNS* usually adopts standard local search as the core search procedure, so that the number of iterations before restart is not a parameter and is often much smaller than the value chosen here. This is true in particular for the *MDP*, whose instances typically exhibit a huge number of local optima.

As introduced in Section 3.3, the *basic VNS*, the *skewed VNS* and the *guided VNS* assume that the restart solutions should be close to the best known one M^* or to a promising solution $\tilde{M}$ provided by local search. Consequently, $k_{\min} = 1$ and $\delta k = 1$, while $k_{\max}$, has been tuned by experiment in the range between 5 and 20. Slightly better results suggested to set $k_{\max} = 10$. For reasons discussed in detail in Section 4.3, we unexpectedly found out that the large value of I_{TS} makes parameter α almost uninfluential on the performance of the *skewed VNS*. We remind that α allows to accept non improving solutions as new starting points, provided that they are far away from the current best known one.

The *accelerated VNS* sets $k_{\max} = \min(m, n - m)$. In order to achieve a speed in the increase of k similar to that of the *VNS* algorithm by Brimberg et al. [10], while at the same time considering a hierarchy of exactly 10 different neighbourhoods as in the first three variants, we set $k_{\min} = k_{\max}/10$ and $\delta k =$

$k_{\max}/10$. In other words, the neighbourhood used gradually increases in 10 steps up to the largest possible one.

The *random VNS* has a trivial degenerate parameter setting, with $k_{\min} = k_{\max} = \min(m, n-m)$ and $\delta k = 0$. In practice, its only difference with respect to *RR-MDP* is that, while building a random restart solution, it forbids the elements of the current best known one.

SS-MDP After some tests with lower values (5, 10, 15), which consistently provided worse results, we set the size of the best subset and the diverse subset composing the reference set to $|B| = |D| = 20$. Larger values would excessively increase the number of restarts, since each of the $40 \cdot 39/2 = 780$ pairs of solutions in the reference set must be used to generate a new starting point. Given a total of $I = 100\,000$ iterations, we set the number of basic Tabu Search iterations to $I_{TS} = 50$ to allow some repetitions of the whole mechanism on the updated reference set. As usual, the tabu tenures of the basic Tabu Search have been fixed to $\ell_{in} = 11$ and $\ell_{out} = 5$.

RR-MDP The tabu tenures of the basic Tabu Search have been fixed to $\ell_{in} = 11$ and $\ell_{out} = 5$ as in the other algorithms. As in the *VNS* algorithms and for the same reasons, the number of basic Tabu Search iterations for each restart has been set to $I_{TS} = 2\,000$.

4.3 An "equal-effort" comparison

The use of an equal number I of basic iterations (i. e., neighbourhood explorations) for all the approaches has the purpose to compare them in the fairest way, focusing on the effectiveness of their distinctive elements, rather than on the implementation details. On one side, the algorithms applying the more refined mechanisms will have an advantage, since the additional time required, for example, to select and combine solutions from the reference set in *SS-MDP* or to generate starting solutions based on frequency in the *guided VNS*, will be ignored. On the other side, the results are unaffected by the more or less careful implementation of such mechanisms, thus providing a clearer estimation of their contribution to the overall quality of the results.

Since $I = 100\,000$, algorithms *RR-MDP* and all versions of *VNS-MDP* restart exactly $I/I_{TS} = 100\,000/2\,000 = 50$ times. As for *XTS-MDP*, since the restart condition on I_{up} seldom occurs and since any improving solution affects the condition on I^*, the number of restarts is $\leq I/I^* = 100\,000/1000 = 100$. On the other side, *SS-MDP* restarts $I/I_{TS} = 100\,000/50 = 2\,000$ times.

Comparing different design of VNS

First, we compare the results obtained by the five different variants of *VNS-MDP*. Unexpectedly, the *skewed VNS* obtained exactly the same results as the *basic VNS* on all instances for all tested values of parameter α. We remind that the *skewed VNS* accepts the best solution $\tilde{M}$ found in the last Tabu Search run

as a promising starting point not only if it improves the best known solution M^*, but also if it is sufficiently distant from M^*. Parameter α tunes the weight of distance in this decision: with a sufficiently large value of α, $\tilde{M}$ is always accepted. However, the combination of a large value for I_{TS}, that is a strong exploitation of Tabu Search, and a small value for $k_{\max}$, that is a restart not far away from M^*, implies that in practice $\tilde{M}$ is either better or identical to M^*, and therefore the basic and the skewed variants behave in the same way. When I_{TS} is small (e. g., $I_{TS} \approx 100$), the two variants differ, but their performance degrades. As well, they differ when k is large, but then the *skewed VNS* would first select a promising solution $\tilde{M}$ far away from M^*, and then produce with the shaking procedure a starting point far away from $\tilde{M}$, thus making the first choice rather meaningless.

		basic VNS			guided VNS		
Benchmark	(n, m)	GapBK	#BK	CPU	GapBK	#BK	CPU
APOM	$(250, 100)$	-	**40**	5.96	-	**40**	5.76
SOM	$(500, 200)$	-	**20**	19.78	-	**20**	19.45
GKD	$(500, 50)$	-	**20**	22.92	-	**20**	22.55
DM1a	$(500, 200)$	0.00%	16	59.78	0.00%	15	60.96
DM1b	$(2000, 200)$	0.26%	0	375.22	0.26%	0	372.62
DM1c	$(500, 50)$	0.02%	18	22.37	0.02%	19	22.55
DM2	$(500, 50)$	0.00%	18	22.19	0.00%	17	22.92

		accelerated VNS			random VNS		
Benchmark	(n, m)	GapBK	#BK	CPU	GapBK	#BK	CPU
APOM	$(250, 100)$	-	**40**	5.77	-	**40**	5.78
SOM	$(500, 200)$	-	**20**	19.43	-	**20**	19.51
GKD	$(500, 50)$	-	**20**	22.16	-	**20**	22.67
DM1a	$(500, 200)$	0.00%	19	60.18	-	**20**	60.56
DM1b	$(2000, 200)$	0.11%	0	398.10	0.08%	4	403.56
DM1c	$(500, 50)$	0.00%	19	22.40	-	**20**	22.20
DM2	$(500, 50)$	-	**20**	22.27	-	**20**	22.40

Table 1: Comparison between the results of four variants of Algorithm *VNS-MDP* on seven benchmarks for the *MDP* (average gap, number of best known results found and computational time).

Table 1 therefore reports only results for the other four variants described in Section 3.3. Each row refers to one of the seven benchmark sets; benchmark DM1 is divided into three subbenchmarks including instances of homogeneous size. The first column reports the names of the benchmarks, the second column the size of their instances (for the first two, the size of the largest ones). The following triplets of columns report for each of the variants the following information: the average percent gap with respect to the best known solution (*GapBK*), the number of best known results found (*#BK*) and the average computational time in seconds. Concerning the gap, label 0.00% represents an average gap lower than 0.005%, whereas a dash marks the benchmarks on which all best known results have been found; in this case, the number of best known results is also marked in bold. We remind that all benchmarks include 20 in-

stances, apart from APOM, which contains 40 instances. As it will be discussed in the next section, all best known results derive from the Iterated Tabu Search algorithm *ITS* proposed by [36] and confirmed by the *VNS* algorithm reported in [10]; they were also confirmed by our experiments. As for the computational times, they correspond to the whole run: the time strictly required to find the best known solution is often much smaller.

Notice that the computational times are more or less the same for all variants, with an 8% increase for those in which $k_{\max}$ is large, corresponding to the larger number of elements to select and swap. They can also be approximated quite precisely with the expression $\gamma I m\,(n-m)$, where $I = 100\,000$ is the total number of explorations and γ is a machine-dependent coefficient; in our case, $\gamma \approx 1.05 \cdot 10^{-8}$ seconds. This supports the hypothesis that most of the time is devoted to the neighbourhood exploration, whose complexity is proportional to $m\,(n-m)$, while the other elements of the algorithms introduce a secondary, if not negligible, overhead.

	basic VNS			guided VNS		
Benchmark	W^+	W^-	p	W^+	W^-	p
APOM	0	0	-	0	0	-
SOM	0	0	-	0	0	-
GKD	0	0	-	0	0	-
DM1a	10	0	-	**15**	0	6.25%
DM1b	**185**	5	0.03%	**178**	12	0.09%
DM1c	3	0	-	1	0	-
DM2	3	0	-	6	0	-

	accelerated VNS			random VNS		
Benchmark	W^+	W^-	p	W^+	W^-	p
APOM	0	0	-	0	0	-
SOM	0	0	-	0	0	-
GKD	0	0	-	0	0	-
DM1a	1	0	-	0	0	-
DM1b	97	113	-	53	**157**	5.45%
DM1c	1	0	-	0	0	-
DM2	0	0	-	0	0	-

Table 2: Statistical analysis with Wilcoxon's test of the comparison between *RR-MDP* and four variants of *VNS-MDP* on seven benchmarks: the algorithm is worse than *RR-MDP* when W^+ is in bold, better when W^- is in bold; p is the probability that the observed difference be due to random sampling.

The *random VNS* outperforms the other three variants: it finds the best known result on all instances up to $n = 500$ and on four of the largest instances. On benchmark DM1B it achieves a better percent gap: 0.08% as opposed to 0.11% for the *accelerated VNS* and 0.26% for the other two competitors. In order to estimate the validity of these conclusions, we have used the *Wilcoxon matched-pairs signed-ranks test* [42]. This is a well-known nonparametric test to compare two paired populations of values, such as the results provided by two competing algorithms on different instances of a problem. It analyses the differ-

ences between the paired measurements on a sample of instances and computes the probability p that, assuming the median difference of the two populations to be zero, i. e., the two algorithms to be equally effective, random sampling would lead to a sum of ranks as far apart as observed, or more so. If p is small, the difference is probably not a coincidence and a large difference between W^+ and W^- indicates that the first population is significantly larger or smaller than the second, i. e., that the first algorithm is better or worse than the second. We have decided to compare the performance of all our algorithms to that of Algorithm *RR-MDP*, which is the simplest one, as it adopts pure random restart.

Table 2 reports W^+, W^- and p for this comparison between *RR-MDP* and the four variants of *VNS-MDP* on the seven benchmarks. Label "-" corresponds to values of p larger than 10%, that is to statistically insignificant differences; a bold value for W^+ indicates that the algorithm performs worse than *RR-MDP*, a bold value for W^- that it performs better. It can be observed that all variants are equivalent to each other and to random restart on most of the benchmarks. The *basic VNS* and the *guided VNS* are more or less equivalent to each other and both dominated by random restart on the largest instances, whereas the *accelerated VNS* is approximately equivalent to random restart and the *random VNS* is slightly better, with a 5.45% probability that such a result might be due to sampling. This supports the conclusion that randomisation and a strong diversification are effective tools while solving the *MDP*.

Comparing our algorithms for MDP

Table 3 summarises the internal comparison between the four approaches considered in the paper, namely *XTS-MDP*, *VNS-MDP*, *SS-MDP* and *RR-MDP*. By *VNS-MDP* we denote in the following the *random VNS* variant, which proved to be the best performing one. It has the same structure as Table 1, reporting for each benchmark and for each algorithm the average percent gap with respect to the best known solution, the number of best known results found and the average computational time in seconds.

The computational times confirm the previous remark on the predominance of neighbourhood exploration over all other elements of the algorithms and consequently on the good approximation provided by expression $\gamma Im\,(n-m)$. It is remarkable that Algorithm *SS-MDP*, though requiring a much larger number of restarts, actually has a lower computational time on average. This is probably due to the efficiency of the procedure which recombines the reference solutions.

Algorithm *VNS-MDP* outperforms the other three, with the partial exception of *RR-MDP*: it finds the best known result on all instances up to $n = 500$ and on four of the largest ones; the average percent gap on the largest instances is 0.08% as opposed to 0.12% for *RR-MDP*, 0.23% for *SS-MDP* and 0.24% for *XTS-MDP*. As in the previous discussion, we have used Wilcoxon's test to estimate the validity of these conclusions. Table 4 reports W^+, W^- and p for the comparison between Algorithm *RR-MDP* and the three other ones on the seven benchmarks. The structure of the table is the same as for Table 2 and

Benchmark	(n, m)	XTS-MDP GapBK	XTS-MDP #BK	XTS-MDP CPU	VNS-MDP GapBK	VNS-MDP #BK	VNS-MDP CPU
APOM	$(250, 100)$	0.00%	39	5.51	-	**40**	5.78
SOM	$(500, 200)$	0.02%	19	19.07	-	**20**	19.51
GKD	$(500, 50)$	-	**20**	21.90	-	**20**	22.67
DM1a	$(500, 200)$	0.00%	15	61.43	-	**20**	60.56
DM1b	$(2000, 200)$	0.24%	0	432.17	0.08%	4	403.56
DM1c	$(500, 50)$	0.15%	9	21.88	-	**20**	22.20
DM2	$(500, 50)$	0.08%	12	21.90	-	**20**	22.40

Benchmark	(n, m)	SS-MDP GapBK	SS-MDP #BK	SS-MDP CPU	RR-MDP GapBK	RR-MDP #BK	RR-MDP CPU
APOM	$(250, 100)$	-	**40**	5.81	-	**40**	5.26
SOM	$(500, 200)$	-	**20**	19.68	-	**20**	18.10
GKD	$(500, 50)$	-	**20**	23.48	-	**20**	21.06
DM1a	$(500, 200)$	0.00%	19	60.61	-	**20**	58.70
DM1b	$(2000, 200)$	0.28%	0	390.70	0.12%	0	406.29
DM1c	$(500, 50)$	0.00%	19	23.41	-	**20**	21.03
DM2	$(500, 50)$	0.01%	18	23.47	-	**20**	20.97

Table 3: Comparison between the results of algorithms *XTS-MDP*, *VNS-MDP*, *SS-MDP* and *RR-MDP* on seven benchmarks for the *MDP* (average gap, number of best known results found and computational time).

Algorithm *RR-MDP* is once again assumed as a reference, so that the results in the two tables are homogeneous. The four algorithms are equivalent on the smaller benchmarks, but on the larger ones and especially on DM1B Algorithm *XTS-MDP* and *SS-MDP* are significantly worse than random restart, whereas *VNS-MDP* is better, though with a 5.45% probability that such a result might be due to sampling. Once again, the simpler strategies, based on randomisation and diversification exhibit a stronger effectiveness.

Benchmark	XTS-MDP W^+	XTS-MDP W^-	XTS-MDP p	VNS-MDP W^+	VNS-MDP W^-	VNS-MDP p	SS-MDP W^+	SS-MDP W^-	SS-MDP p
APOM	1	0	-	0	0	-	0	0	-
SOM	1	0	-	0	0	-	0	0	-
GKD	0	0	-	0	0	-	0	0	-
DM1a	**15**	0	6.25%	0	0	-	1	0	-
DM1b	**204.5**	5.5	0.02%	53	**157**	5.45%	**207**	3	0.02%
DM1c	**66**	0	0.10%	0	0	-	1	0	-
DM2	**36**	0	0.78%	0	0	-	3	0	-

Table 4: Statistical analysis with Wilcoxon's test of the comparison between *RR-MDP* and the other three algorithms on seven benchmarks: the algorithm is worse than *RR-MDP* when W^+ is in bold, better when W^- is in bold; p is the probability that the observed difference be due to random sampling.

4.4 Comparison with the state of the art

In this section, we will compare the performance of the proposed algorithms to the best ones introduced in the literature. The aim of the comparison is not to claim their predominance on the other algorithms, but mainly to support our claim that they were carefully implemented, and that general conclusions can be drawn from the study of their performance. Indeed, they all perform worse than the *VNS* algorithm reported in [10], but comparably to or better than the other algorithms recently proposed in the literature, which are *Tabu_D2* [15], *ITS* [36] and the Scatter Search algorithm presented in [18].

Benchmark	XTS-MDP	VNS-MDP	SS-MDP	RR-MDP	CPU
SOM	0.2	0.0	0.0	0.0	$\approx$ 20 sec
DM1a	4.3	0.0	0.1	0.0	$\approx$ 60 sec
DM1b	277.5	86.4	320.2	132.9	$\approx$ 400 sec
DM1c	11.5	0.0	0.2	0.0	$\approx$ 20 sec
DM2	595.9	0.0	44.2	0.0	$\approx$ 20 sec

	VNS		ITS		Tabu_D2	SS
Benchmark	20 sec	200 sec	20 sec	200 sec		
SOM	0.0	0.0	0.0	0.0	4.2	6.8
DM1a	0.1	0.0	4.2	0.8	39.8	-
DM1b	115.3	30.6	264.8	104.3	599.8	695.4
DM1c	0.0	0.0	1.0	0.0	7.4	-
DM2	25.9	0.0	97.4	0.0	516.2	126.3

Table 5: Comparison between the results of the proposed algorithms and the four best performing ones in the literature on five benchmarks for the *MDP* (average absolute gap between the best known result and the best found by each algorithm).

The use of different programming languages, different compilers and different computers makes the fair comparison of algorithms a very hard task. Table 5 attempts to provide a meaningful comparison along the lines of [10]. Each row corresponds to one of the five benchmarks for which results are available for the algorithms compared; benchmarks APOM and GKD have not been used, as they are indeed easier than the others. The upper half of the table refers to our four algorithms: a column reports the average absolute gap between the best known value and the best solution found by each of them; the last column reports the average computational time, which is much similar for all of them. The lower half of the table refers to the four best algorithms proposed in the literature. The results for *VNS* and *ITS*, taken from [10], refer both to a single run, with a time limit of 20 seconds, and to 10 independent runs, with a total time of 200 seconds. As for *Tabu_D2* and *SS*, no computational time is reported in [10] together with the results. Finally, we remark also that the use of randomness makes difficult a fair comparison among different algorithms since different random number generators can show different accuracy and computational efficiency. For instance, this is the case of the two generators employed in our tests, i.e., *ran1* [37] and *RngStream* [33].

A comparison referring to different time limits and different machines is not

completely fair. A comparison performed gathering all the competing algorithms and running them on the same machine for the same computational time, both with shorter and longer runs, is an ongoing work of other authors, accessible online [11]. We have cooperated to this work by making our algorithms available. However, the machines here involved are not strongly different: a 2.2 GHz AMD Opteron for our algorithms, a 1.8 GHz for *VNS*, a Pentium M 1.733 GHz for *ITS* and a 3 GHz for *Tabu_D2* and *SS*. Hence, we feel at least authorised to draw the rough conclusion that all algorithms tested are competitive with the state of the art, though they are not the best. In particular, the *VNS* algorithm proposed in [10] is the best of these algorithms; *VNS-MDP* and *RR-MDP* are comparable to *ITS*, the performance of *SS-MDP* and *XTS-MDP* is between *ITS* and *SS*, while *Tabu_D2* is worse.

Benchmark	(n, m)	Gap UB - BK
APOM	$(250, 100)$	1.18%
SOM	$(500, 200)$	5.69%
GKD	$(500, 50)$	-
DM1a	$(500, 200)$	1.08%
DM1b	$(2000, 200)$	8.99%
DM1c	$(500, 50)$	13.46%
DM2	$(500, 50)$	13.45%

Table 6: Average percent gap between the best known results and the best known upper bounds reported in the literature of the *MDP*.

For reference purposes, Table 6 reports for each of the seven benchmarks the average percent gap between the best known results reported in [36] and [10] and the upper bounds computed by Semidefinite Programming and branch-and-bound in [3]. This information expresses, at least partly, the relative hardness between the benchmarks.

A common point between our best performing algorithms (the random variant of *VNS-MDP* and *RR-MDP*) and the *VNS* algorithm proposed in [10] is a restart mechanism based on solutions generated most of the time nearly at random and, in the case of *VNS-MDP* and partly of *VNS*, far away from the current best known solution. Our statistical tests seems to support the conclusion that most refinements of the restart mechanism have a negative influence on the performance, with the remarkable exception of diversification.

A relevant difference between the best algorithm proposed in the literature and ours is the use of pure local search as the core procedure, instead of Tabu Search. Consequently, each restart is certainly less effective, but far more efficient: it terminates in local optima, but the number of restarts is much higher. It is certainly hard to analyse the interaction between these two conflicting effects.

5 Conclusions

This paper presents four local search metaheuristics for the *MDP*: an Exploring Tabu Search, a Variable Neighbourhood Search, a Scatter Search and a Random Restart algorithm. All these algorithms are competitive with the state of the art, as they hit most of the best known results published in the literature, while keeping a small gap even in the worst cases. All these algorithms are based on the same basic Tabu Search mechanism with the same parameter setting, with a slight exception for *XTS-MDP*, which allows the tabu tenures to vary around the values adopted by the others. When run for the same number of iterations (i. e., neighbourhood explorations), the best performing algorithm are the ones based on random restart (*RR-MDP*) or a nearly random restart in which the elements of the current best known solution are forbidden (the *random VNS* variant of *VNS-MDP*). In our opinion, this is related to the extremely simple structure of the problem, which has no specific feature to be exploited by a sophisticated strategy. Of course, we do not suggest that this result holds on all problems, but it is intriguing to wonder for how many problems complex algorithms might be dominated by much simpler ones, largely based on random restart. The best algorithm available in literature for the *MDP* is the VNS proposed in [10]: though we could not fully reproduce their results on the hardest instances of the benchmark, our computational experience reported for both VNS and random restart algorithms, suggests that the good quality of their results strongly might depend on the use of a large neighbourhood to restart the search – which exploits the advantages of a nearly random restart – instead of using the VNS methodology per se.

Acknowledgements

The authors wish to thank Yari Melzani, Gian Paolo Ghilardi, Alberto Ghilardi, Andrea Beretta and Marco Tadini for their help in the computational experience reported in [5] and [6].

References

[1] R.K. Ahujaa, O. Ergun, J.B. Orlin, and A.P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123:75–102, 2002.

[2] P. M. D. Andrade, A. Plastino, L. S. Ochi, and S. L. Martins. GRASP for the Maximum Diversity Problem. In *Proceedings of the Fifth Metaheuristics International Conference (MIC 2003)*, 2003.

[3] R. Aringhieri, M. Bruglieri, and R. Cordone. Optimal results and tight bounds for the Maximum Diversity Problem. *Foundations of Computing and Decision Sciences*, 34(2):73–85, 2009.

[4] R. Aringhieri, M. Bruglieri, F. Malucelli, and M. Nonato. Metaheuristics for a vehicle routing problem on bipartite graphs with distance constraints. In *Proc. of 6th Metaheuristics International Conference*, August 2005.

[5] R. Aringhieri and R. Cordone. Better and faster solutions for the Maximum Diversity Problem. Note del Polo 93, Università degli Studi di Milano, Crema, April 2006.

[6] R. Aringhieri, R. Cordone, and Y. Melzani. An Ant Colony Optimization approach to the Maximum Diversity Problem. Note del Polo 109, Università degli Studi di Milano, Crema, October 2007.

[7] R. Aringhieri, R. Cordone, and Y. Melzani. Tabu search vs. GRASP for the Maximum Diversity Problem. *4OR: A Quarterly Journal of Operations Research*, 6(1):45–60, 2008.

[8] R. Aringhieri and M. Dell'Amico. Comparing Metaheuristic Algorithms for the SONET Network Design Problems. *Journal of Heuristics*, 11(1):35–57, january 2005. ISI impact factor 2006-2007: 0.740-0.644.

[9] R. Aringhieri and M. Dell'Amico. Solution of the SONET Ring Assignment Problem with capacity constraints. In C. Rego and B. Alidaee, editors, *Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search*, pages 93–116. Kluwer Academic Publisher, 2005.

[10] J. Brimberg, N. Mladenović, D. Urošević, and E. Ngai. Variable neighbourhood search for the heaviest k-subgraph. *Computers & Operations Research*, 36:2885–2891, 2009.

[11] V. Campos, A. Duarte, M. Gallego, F. Gortazar, R. Martì, and E. Piñana. Optsicom. `http://heur.uv.es/optsicom/`.

[12] R. Cordone and R. Wolfler Calvo. A heuristic for the vehicle routing problem with time windows. *Journal of Heuristics*, 7(2):107–129, March 2001.

[13] M. Dell'Amico and M. Trubian. Solution of large weighted equicut problems. *European Jurnal of Operational Research*, 106:500–521, 1998.

[14] M. Dorigo and T. Stuetzle. *Ant Colony Optimization*. MIT Press, 2004.

[15] A. Duarte and R. Martì. Tabu search and GRASP for the Maximum Diversity Problem. *European Journal of Operational Research*, 178:71–84, 2007.

[16] A. Esposito, M.S. Fiorenzo Catalano, F. Malucelli, and L. Tarricone. A new matrix bandwidth reduction algorithm. *Operations Research Letters*, 23:99–107, 1998.

[17] P. Festa and M. G. C. Resende. GRASP: An annotated bibliography. In C. C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.

[18] M. Gallego, A. Duarte, M. Laguna, and R. Martì. Hybrid heuristics for the Maximum Diversity Problem. *Computational Optimization and Applications*, 44(3):411–426, December 2009.

[19] M. Gendreau, A. Hertz, and G. Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40:1086–1094, 1992.

[20] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40:1276–1290, 1994.

[21] J. B. Ghosh. Computational aspects of Maximum Diversity Problem. *Operation Research Letters*, 19:175–181, 1996.

[22] F. Glover, G. Hersh, and C. McMillian. Selecting subset of maximum diversity. MS/IS 77-9, University of Colorado at Boulder, 1977.

[23] F. Glover, C. C. Kuo, and K. S. Dhir. A discrete optimization model for preserving biological diversity. *Appl. Math. Modelling*, 19(11):696–701, November 1995.

[24] F. Glover, C. C. Kuo, and K. S. Dhir. Integer programming and heuristic approaches to the Minimum Diversity Problem. *Journal of Business and Management*, 4(1):93–111, 1996.

[25] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[26] P. Hansen and N. Mladenović. Variable neighbourhood search: Principles and applications. *European Journal of Operations Research*, 130:449–467, 2001.

[27] P. Hansen and N. Mladenovic. Variable neighborhood search. In F. Glover and G. Kochenagen, editors, *Handbook of Metaheuristics*, pages 145–184. Kluwer Academic Publishers, 2003.

[28] J. Kleinberg and É. Tardos. *Algorithm Design*. Addison-Wesley, 2006.

[29] G. Kochenberger and F. Glover. Diversity data mining. Working Paper Series HCES-03-99, The University of Mississipi, 1999.

[30] C. C. Kuo, F. Glover, and K.S. Dhir. Analyzing and modeling the Maximum Diversity Problem by zero-one programming. *Decision Science*, 24:1171–1185, 1993.

[31] M. Laguna and V.A. Armentano. Lessons from applying and experimenting with scatter search. In C. Rego and B. Alidaee, editors, *Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search*, chapter 10, pages 229 – 246. Kluwer Academic Publishers, 2005.

[32] M. Laguna and R. Martí. *Scatter Search: methodology and Implementations in C.* Kluwer Academic Publishers, 2003.

[33] P. L'Ecuyer. Good parameters and implementations for combined multiple recursive random number generators. *Operations Research*, 47(1):159–164, 1999.

[34] S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21:498–516, 1973.

[35] R. Martì, M. Gallego, and A. Duarte. A branch and bound algorithm for the Maximum Diversity Problem. *European Journal of Operational Research*, 200:36–44, 2010.

[36] G. Palubeckis. Iterated tabu search for the Maximum Diversity Problem. *Applied Mathematics and Computation*, 189:371–383, 2007.

[37] W. H. Press. *Numerical Recipes in C.* Cambridge University Press, Cambridge, 1990.

[38] L.F. Santos, M.H. Ribeiro, A. Plastino, and S.L. Martins. A Hybrid GRASP with Data Mining for the Maximum Diversity Problem. In M.J. Blesa, C. Blum, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics, Second International Workshop*, volume 3636 of *Lecture Notes in Computer Science*, pages 116–127. Springer Berlin / Heidelberg, 2005.

[39] G. C. Silva, M. R. Q. de Andrade, L. S. Ochi, S. L. Martins, and A. Plastino. New heuristics for the Maximum Diversity Problem. *Journal of Heuristics*, 13:315–336, 2007.

[40] G. C. Silva, L. S. Ochi, and S. L. Martins. Experimental comparison of greedy randomized adaptive search procedures for the Maximum Diversity Problem. In *Proceedings of the 3rd International Workshop on Efficient and Experimental Algorithms (WEA 2004)*, volume 3059 of *Lectures Notes on Computer Science*, pages 498–512. Springer Berlin / Heidelberg, 2004.

[41] R. Weitz and S. Lakshminarayanan. An empirical comparison of heuristic methods for creating maximally diverse group. *Journal of the Operational Research Society*, 49:635–646, 1998.

[42] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1:80–83, 1945.