Tabu Search vs. GRASP for the Maximum Diversity Problem

4OR manuscript No.
(will be inserted by the editor)

Tabu Search vs. GRASP for the Maximum Diversity Problem

Roberto Aringhieri* Roberto Cordone and Yari Melzani

Dipartimento di Tecnologie dell'Informazione, Università di Milano, Crema, Italy.
e-mail: {aringhieri,cordone}@dti.unimi.it, ymelzani@crema.unimi.it

Received: December 2005 / Revised version: October 2006

Abstract The *Maximum Diversity Problem* (*MDP*) consists in determining a subset M of given cardinality from a set of elements N, in such a way that the sum of the pairwise differences between the elements of M is maximum. This problem, introduced by Glover Glover et al. (1977), has been deeply studied using the GRASP methodology Ghosh (1996); Andrade et al. (2003); Silva et al. (2004); Andrade et al. (2005). GRASPs are often characterized by a strong design effort dedicated to the randomized generation of high quality starting solutions, while the subsequent improvement phase is usually performed by a standard local search technique. The purpose of this paper is to explore a somewhat opposite approach, that is to refine the local search phase, by adopting a Tabu Search methodology, while keeping a very simple initialization procedure. Extensive computational results show that Tabu Search achieves both better results and much shorter computational times with respect to those reported for GRASP.

Keywords: Maximum Diversity, GRASP, Tabu Search

1 Introduction

Given a set N of n elements and a diversity measure d_{ij} between each pair of elements (i, j), with $d_{ij} > 0$ for $i \neq j$ and $d_{ij} = 0$ otherwise, the *Maximum Diversity Problem* (*MDP*) consists in determining a subset $M \subset N$ of given cardinality m, such that the sum of the pairwise differences between the

* Corresponding author: Roberto Aringhieri, Dipartimento di Tecnologie dell'Informazione, Università di Milano, Via Bramante 65, 26013 Crema, Italy, Tel. +39-0373-898051, Fax +39-0373-898074.

elements of M is maximum. Let $x_i = 1$ if element $i \in N$ belongs to the solution M, $x_i = 0$ otherwise. The *MDP* can be formulated as follows:

$$\max z = \frac{1}{2} \sum_{i \in N} \sum_{j \in N} d_{ij} x_i x_j \tag{1.1}$$

$$\sum_{i \in N} x_i = m \tag{1.2}$$

$$x_i \in \{0, 1\} \qquad i \in N \tag{1.3}$$

There are several different applications for this model. For example, while forming work teams, juries or student groups for project work, it is often desirable to gather a fixed number of individuals whose characteristics are as diversified as possible: work teams should include the largest possible range of skills, juries should represent the widest variety of points of view existing in a community, student groups should allow to share and exchange different backgrounds. In this framework, d_{ij} models the differences between individuals i and j with respect to some relevant characteristics.

Other interesting applications concern the allocation of available resources to preserve biological diversity Glover et al. (1995), medical treatments, the scheduling of final exams, VLSI design and data mining Kochenberger and Glover (1999).

The problem is *strongly* $\mathcal{NP}$-hard Kuo et al. (1993). This can be proved by reduction from the *k-Clique* problem. Given an instance of the latter, that is an undirected graph $G(V, E)$, build the following instance of the *MDP*: for each vertex of V, define an element of N; for each pair $(i, j) \in E$ set $d_{ij} = 1$, whereas $d_{ij} = 0$ when $(i, j) \notin E$; finally, set $m = k$. Graph G contains a clique of k vertexes if and only if the optimum of the *MDP* is equal to $k(k-1)/2$.

The *MDP* was introduced by Glover Glover et al. (1977), who presented an integer linear formulation, which can be solved only for small instances (less than 40 elements) because of the quadratic number of binary variables required. Other authors have applied the quadratic formulation reported above on instances of approximately the same size Ghosh (1996), in order to evaluate the performance of heuristic algorithms.

Apart from some early greedy and stingy heuristics Glover et al. (1996); Weitz and Lakshminarayanan (1998), nearly all heuristic approaches to the *MDP* adopt the Greedy Randomized Adaptive Search Procedure methodology (*GRASP*) Festa and Resende (2002). The first GRASP procedure for the *MDP* was proposed by Ghosh Ghosh (1996) obtaining good results just for instances up to 40 elements. Andrade et al. Andrade et al. (2003) developed a new GRASP able to solve instances up to 250 elements and to find better solutions on Ghosh's benchmark. Several different GRASP algorithms obtained by combining different construction and local search

heuristics were proposed by Silva et al. Silva et al. (2004). They were extensively tested over a benchmark set of instances randomly generated up to 500 elements: the results obtained were compared to the ones obtained by the previous GRASPs showing a better performance. In the end, a GRASP with path relinking has been described in Andrade et al. (2005).

These GRASP algorithms are characterized by a strong design effort dedicated to build high quality randomized starting solutions. The subsequent improvement phase is usually performed by a standard local search technique. The purpose of this paper is to explore a somewhat opposite approach, that is to refine the local search phase, by adopting a Tabu Search methodology, while keeping a very simple initialization procedure. By using ad hoc memory mechanisms (both on a short and on a long term), it was possible to achieve both better results and much shorter computational times with respect to those reported in the literature.

Section 2 describes the Tabu Search algorithm proposed, while Section 3 discusses its performance in comparison to the best algorithms reported in the literature. Conclusions and future work close the paper.

2 The Tabu Search algorithm

After introducing some notation, we present very simple greedy and local search heuristics, which are the basic elements of our Tabu Search algorithm, based on ad hoc memory mechanisms. Then, we briefly describe the main ingredients for a standard Tabu Search algorithm (which are deeply discussed in Glover and Laguna (1997)), providing more details only about the components which have been specifically designed to solve the *MDP*.

Notation

The contribution of each element $i \in N$ to a given solution $M \subset N$ is defined as $D_i = \sum_{j \in M} d_{ij}$. Clearly, $z = (\sum_{i \in M} D_i)/2$.

Greedy Initialization

Let $M^{(0)} = \{i, j\} \subset N$ be the pair of elements of maximum diversity d_{ij} and $z^{(0)} = d_{ij}$. A feasible solution, i.e. a solution containing m elements, can be built from $M^{(0)}$ by consecutively adding one element k at a time. At the h-th iteration, the element $k^{(h)}$ to be added is chosen as

$$k^{(h)} = \arg\max_{i \in N \setminus M^{(h-1)}} D_i$$

giving rise to the following solution

$$M^{(h)} = M^{(h-1)} \cup \{k^{(h)}\} \text{ and } z^{(h)} = z^{(h-1)} + D_{k^{(h)}}.$$

After each iteration, we can easily update D_i for each $i \in N \setminus M^{(h)}$ by adding the value $d_{ik^{(h)}}$ to it. We actually update D_i also for $i \in M^{(h)}$, with the same formula, since these values are needed by the subsequent improvement phase. Each iteration requires $O(n)$ time, so that the overall procedure is $O(mn)$.

Neighborhood definition

The solution obtained by the previous greedy algorithm is the starting point for a local search improvement phase. This is based on the most natural neighborhood for *MDP*, that is the exchange between a single element s in the solution and a single element t out of it, that is $M' = M \cup \{t\} \setminus \{s\}$.

It is possible to efficiently evaluate such a move without recomputing the objective function from scratch. Let z be the value of solution M. The value z' of the new solution M' is obtained by subtracting the total contribution of the old element s (that is D_s) and adding the total contribution of the new element t (that is $D_t - d_{st}$), that is, more formally,

$$z' = z - D_s + D_t - d_{st}.$$

The move yielding the largest improvement in the objective function is selected and applied. After each move, the values of D_i are updated as follows:

$$D_i = D_i - d_{is} + d_{it} \qquad i \in N.$$

In particular, $D_s = D_s + d_{st}$ and $D_t = D_t - d_{st}$.

We exchange each of the m elements inside the solution with each of the $n - m$ elements outside of it. The evaluation of each move is done in constant time, and the update after a move takes $O(n)$ time. Therefore, the complexity of each local search iteration is $O(mn)$.

Tabu Search

Tabu Search is a well-known metaheuristic approach based on local search and on a mechanism to avoid looping over already visited solutions Glover and Laguna (1997). This mechanism consists in a finite-length list of forbidden moves, named tabu list.

As we want to avoid both the inclusion of a recently removed element and the removal of a recently included element, we define two independent tabu lists: list L_{in} forbids an element to enter the solution for ℓ_{in} iterations, whilst

list L_{out} forbids an element to exit for ℓ_{out} iterations. A move improving the best known solution is always performed, even if it is tabu (*aspiration criterion*).

Short term memory

The short term memory mechanism is a device which allows to intensify or diversify the search depending on the results of the search: the length of the tabu list (*tabu tenure*) decreases if the objective function has steadily improved in the most recent iterations, and it increases if the objective function has steadily worsened. The purpose of decreasing the tabu tenure is to intensify the search in those regions which provide improving solutions, and therefore appear more promising. On the contrary, increasing the tabu tenure speeds up the leaving 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.

More specifically, the tabu tenure ℓ_{in} varies in a given range $\left[\ell_{in}^{m}, \ell_{in}^{M}\right]$: at the beginning of the algorithm, it is set to the middle point of this range $\ell_{in}^{(0)} = \left(\ell_{in}^{m} + \ell_{in}^{M}\right)/2$; 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 ranges from ℓ_{out}^{m} to ℓ_{out}^{M}, starting at $\ell_{out}^{(0)} = \left(\ell_{out}^{m} + \ell_{out}^{M}\right)/2$. The values of parameters ℓ_{in}^{m}, ℓ_{in}^{M}, ℓ_{out}^{m}, ℓ_{out}^{M}, T_w and T_i must be tuned in advance: Table 3.1 provides the specific values adopted in our experimental campaign. On the contrary, the amount $\Delta\ell$ is commonly fixed to 1 in the literature. However, preliminary experiments showed that, when the tabu tenure increases up to its maximum value, 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 complementary behavior 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 $\Delta\ell$, instead of a fixed one: as the length of the tabu list approaches the lower or the upper limit of its range, $\Delta\ell$ becomes larger. A detailed description of this mechanism is given in Section 3.

Long term memory

Since we have observed that the value of the objective function may be very similar for many solutions in a given neighborhood, we decided to introduce

a long term memory mechanism, which is known in the literature as eXploring Tabu Search (XTS) Dell'Amico and Trubian (1998). The basic idea is to maintain a set of good solutions which were evaluated but not chosen, because they were worse than the best one in the neighborhood. These solutions could be a good starting point to diversify the search, leading it toward promising regions. Therefore, every time suitable conditions verify, the search restarts from one of these solutions.

To implement this mechanism, we use a list $\mathcal{M}$ of fixed length, composed of *second* solutions: when exploring the neighborhood of solution M, the best solution M^* becomes the incumbent, and the second best solution M' is inserted in $\mathcal{M}$, if its value is better than the worst in $\mathcal{M}$. The restart of the search is subject to two conditions: either the best known solution is not improved for I_{c_1} 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_{c_2} consecutive iterations. The first condition indicates that the currently explored region does not seem to be promising. The second condition indicates that the short term mechanism seems to be insufficient to diversify the search. When any of these conditions holds, the best solution in $\mathcal{M}$ is removed from the list and becomes the new incumbent. In order to replicate exactly the moment in which M' was found, it is required to save the whole state of the computation, that is the current solution M, the current tabu lists L_{in} and L_{out}, the parameters concerning the short term mechanism and the move which generates M'. After a limit value of $I_{\max}$ iterations the search is terminated. Once again, the specific values of $I_{\max}$, I_{c_1}, I_{c_2} and the length of list $\mathcal{M}$ must be tuned by experience, and the values used in our experiments are reported in Table 3.1.

3 Computational results

In this section we report the computational results of our Tabu Search algorithm, comparing them to those obtained by various GRASPs Ghosh (1996); Andrade et al. (2003); Silva et al. (2004); Andrade et al. (2005). Before discussing the computational results, we describe the computational environment, the benchmark instances used, the tuning of the algorithm's parameters and the main features of the competing algorithms.

Setting up the computational experiments

Our algorithm is coded using the C standard 2 and runs on a Linux machine with G++ 3.3.6 compiler. The PC is an Intel Pentium 4 Mobile 2.8Ghz with 512MB of main memory.

For our experiments, we have used two sets of benchmark instances available in the literature. Benchmark B_1, proposed in Andrade et al. (2003),

is composed of 40 instances such that $n = 50, 100, 150, 200, 250$ and m is equal to 20% or 40% of n. There are four different types of instances:

- type A: the elements are points on a plane; their coordinates are randomly extracted from $[1, 9]$ and d_{ij} is equal to the Euclidean distance between elements i and j;
- type B: all differences d_{ij} are random integers with a uniform distribution in $[1, 9999]$;
- type C: one half of the differences are random integers uniformly distributed in $[1, 9999]$, whilst the other half are random integers uniformly distributed in $[1, 4999]$;
- type C: one half of the differences are random integers uniformly distributed in $[1, 9999]$, whilst the other half are random integers uniformly distributed in $[5000, 9999]$;

Benchmark B_2, proposed in Silva et al. (2004), is composed of 20 instances such that $n = 100, 200, 300, 400, 500$, m is equal to 10%, 20%, 30%, 40% of n and the d_{ij} coefficients are random integers uniformly distributed in $[0, 9]$. These sets are also available at website: `http://www.dti.unimi.it/~aringhieri`.

Preliminary computational experiments have been done in order to tune the parameters of our algorithm, i.e. the lengths of the two tabu lists, the total number of iterations and the values regulating both the short term and the long term mechanisms. Table 3.1 reports the parameters' values. These values have been used on all instances in the two benchmarks, though in some cases different parameters would have provided the same result in shorter time, especially on smaller instances. We remind that the initial

Tabu Search	$\ell_{in}^{(0)} = 11$	$\ell_{out}^{(0)} = 5$	$I_{\max} = 2000$
Short Term	$\ell_{in}^{m} = 8$ $\ell_{in}^{M} = 14$	$\ell_{out}^{m} = 3$ $\ell_{out}^{M} = 7$	$T_i = 3$ $T_w = 5$
Long Term	$\lvert\mathcal{M}\rvert = 15$	$I_{c_1} = 1000$	$I_{c_2} = 300$

Table 3.1. Parameters' values.

values of ℓ_{in} and ℓ_{out} are set to the middle point of the corresponding range and, therefore, they are equal to 11 and 5, respectively. Finally, the values of $\Delta\ell_{in}$ and $\Delta\ell_{out}$ depend on how far the current length of each list is from the median point of its range. In detail, we have:

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

In the experimental comparison, we will also consider the results obtained by two limited versions of the algorithm. The former is a tabu search with short term (but no long term) memory, which corresponds to setting $I_{c_1} = I_{c_2} = +\infty$. The latter corresponds to an even more limited standard tabu search, with fixed tabu tenures equal to $\ell_{in}^{(0)}$ and $\ell_{out}^{(0)}$, which corresponds to setting $\Delta\ell_{in} = \Delta\ell_{out} = 0$. All other parameters assume in the three algorithms the values above reported. The two limited versions are clearly less effective than the proposed one, but we take them into account because in two cases (out of 60 instances) the short term memory performs better than the long term one and in one case the standard Tabu Search proves the best of the three. Similar results can be obtained by considering only the long term memory Tabu Search and tuning ad hoc the value of the parameters or suitably increasing the maximum number of iterations.

Competing algorithms

The best known results in the literature for the two available benchmarks have been obtained by several different algorithms under distinct environment conditions. In detail, the competing algorithms are Ghosh's GRASP heuristic as implemented by Andrade et al. Ghosh (1996); Andrade et al. (2003), Andrade's GRASP heuristic Andrade et al. (2003), Silva's six GRASP heuristics (named from G3 to G8) Silva et al. (2004) and Andrade's six GRASP heuristics with path-relinking (named from T1E1 to T3E2) Andrade et al. (2005), that is 14 different algorithms. All of these algorithms are based on the GRASP paradigm.

Ghosh's GRASP Ghosh (1996). The constructive phase of this algorithm selects one element at a time and adds it to the current partial solution until this includes m elements. Since the contribution that the new element would give to the value of the objective is partly unknown, the algorithm computes a lower and an upper estimate, generates a random value uniformly distributed between them, and chooses the element for which this value is maximum.

The resulting solution is improved by a classical local search procedure, whose neighborhood is the same adopted by our Tabu Search: replacing one element in the solution with one out of it, so as to maximize the improvement. Once in a local optimum, the search terminates and a new constructive phase starts. The process goes on for a given number of iterations.

Andrade's GRASP Andrade et al. (2003). Andrade's constructive phase limits the choice of the new element to a *Restricted Candidate List* (*RCL*). This is made up by ordering the elements with respect to a suitable greedy ion, selecting the best m ones and removing those whose value presents a

difference larger than the average from the following element in the given order. The greedy criterion adopted changes during the construction: in the last $m/2$ iterations, it is the total diversity with respect to the elements already chosen (i.e. D_i); in the first $m/2$ iterations, it is a weighted sum of this criterion and the average diversity of the element with respect to the whole set. The improvement phase is the same as in the previous algorithm.

Silva's GRASPs G3-G8 Silva et al. (2004). These six heuristics combine three constructive methods (MDI, KLD and KLDv2) and two improvement algorithms (GhA and SOMA). The most distant insertion heuristic (MDI) adopts the two greedy criteria introduced by Andrade et al. (2003), but builds the *RCL* by simply selecting the best k elements. Parameter k is tuned by a sophisticated mechanism known as *reactive GRASP*: given a finite set of possible values, in a first block of iterations the algorithm tests each value for the same number of iterations; in the following blocks, the values which have provided the best average results are adopted for a larger number of iterations. The other two constructive methods build the *RCL* and tune its length in the same way, but the greedy criterion adopted by KLD to evaluate each element i is the sum of the k largest diversities d_{ij}, whereas KLDv2 employs an adaptive greedy criterion which corresponds to the upper estimate introduced by Ghosh (1996).

Only the best solution generated by the constructive heuristic during a given number of iterations undergoes the improvement phase. GhA is the basic local search adopted in all other algorithms; SOMA first reaches a local optimum with respect to the described neighborhood and then starts exchanging two elements, instead of a single one, until a local optimum is reached also with respect to this second neighborhood.

Andrade's GRASPs with path relinking Andrade et al. (2005). The constructive and improvement phases are the same as in Andrade et al. (2003). However, an elite set is maintained, which contains the best solutions found during the search. At each iteration, the current solution is combined with an elite solution, by exchanging (one at a time) the elements which belong to the current solution and not to the elite one with those which belong to the elite solution and not to the current one. The intermediate solutions thus obtained are evaluated, and possibly update the elite set. The relinking phase can move from the elite solution to the current one (backward relinking), in the opposite direction (forward relinking) or alternatively in the two directions (mixed relinking). The elite solution can be chosen at random or in a greedy way. This gives rise to the six different algorithms labelled from T1E1 to T3E2.

The main effort of these algorithms concerns the construction phase instead of the improvement one: most of them adopt restricted candidate lists

built with sophisticated, and in some cases auto-adaptive, mechanisms, in order to improve the starting solutions. Summarizing, most of them share a limited number of constructive procedures, and nearly all of them share the same standard improvement procedure. On the contrary, our Tabu Search is initialized by a trivial constructive procedure, it adopts the same neighborhood employed by the competing algorithms, but it adds the tabu mechanism and suitable intensification and diversification devices to enhance the search. We recall that the purpose of this paper is to compare these two different approaches. In the following, we report the computational results of this comparison.

Results for benchmark B_1

It is not easy to establish a comparison on this benchmark, since the best known values have been obtained from all the 14 competing algorithms, but detailed values are available only for the six GRASP algorithms with path-relinking, as also reported in Andrade et al. (2005). We therefore compare first our best results to the best known ones, then the results of our best performing algorithm (the tabu search with long term memory) to the results obtained by the best performing GRASP with path relinking, that is T3E2.

For 32 instances out of 40 we have equalled the best result reported in the literature. Table 3.2 discusses the remaining 8 instances. The first column contains the names of the instances. The following two columns report, respectively, our result and the best known one in the literature (the best between them is bolded). The last column reports the difference between the two values. In 5 cases out of 8, we improve the best known result; in 3 cases our performance is worse. Most of the time, the difference is quite small. Only 3 results are markedly different: in two of them, our results are better. We remind that this comparison opposes 3 slight variants of a single algorithm to 14 variants of 4 different algorithms. As we shall see in the following, the best known result for instance *B250m50* can actually be improved with respect to the literature by replacing the greedy initialization with a random one: the new best known value is 7 389 784 versus 7 388 997, with an improvement of 784.

If one compares our Tabu Search with long term memory to the best performing GRASP with path-relinking (T3E2), the two algorithms provide the same result for 29 instances out of 40, T3E2 proves better for instance *B250m50* and our Tabu Search proves better for the remaining 10 instances (and in 9 cases the difference is large). This suggests that our Tabu Search is more effective than T3E2. Table 3.3 compares the computational times in seconds and the best results obtained by the two algorithms. The table reports these data only for the instances of type B, C, D and size equal to

Instance	Tabu Search	Literature	Δ
A250m50	**12 654**	12 653	1
B200m80	17 544 447	**17 544 448**	-1
B250m50	7 379 797	**7 388 997***	-9 200
B250m100	**27 168 460**	27 162 906	5 554
C100m20	**1 207 522**	1 205 722	1 800
D150m60	**13 611 262**	13 611 261	1
D200m80	**24 133 321**	24 133 320	1
D250m100	37 753 118	**37 753 120**	-2

Table 3.2. Comparison between the best Tabu Search results and the best results in the literature on benchmark B_1 (when different). The best result on instance *B250m50* can be improved up to 7 389 784.

150, 200 and 250, because the computational times for the other instances are not available in the literature. However, these are actually the most relevant instances in the benchmark, since they are the hardest ones. The columns labelled "Instance" report the name of the instance, the columns labelled "XTS" and "T3E2" report the computational times in seconds for the competing algorithms. The columns labelled "Δ" provide the difference between the best result obtained by the Tabu Search and the best one obtained by T3E2. Since the best results just discussed were obtained during three runs of algorithm T3E2, we have multiplied by 3 the computational times reported in Andrade et al. (2005), which refer to the average of the three runs. Then, we have divided them by 7.3, which is the ratio of the CPU frequencies of the different machines employed (T3E2 runs on a 550 MHz Intel Pentium III PC with 384 MB of RAM), in order to make the computational times at least approximately comparable. Even after this correction, the Tabu Search is from 16 to 32 times faster than T3E2, and similar ratios hold for the other algorithms proposed in Andrade et al. (2005).

Instance	XTS	T3E2	Δ	Instance	XTS	T3E2	Δ
B150m30	1.76	43.56	0	C200m80	14.68	281.10	0
B150m60	6.14	181.23	3524	C250m50	11.12	219.45	0
B200m40	5.78	117.53	0	C250m100	32.96	927.95	0
B200m80	18.90	504.66	0	D150m30	1.42	27.95	352
B250m50	15.99	353.01	-9223	D150m60	4.70	103.97	2111
B250m100	44.61	1303.97	14766	D200m40	4.62	82.60	4680
C150m30	1.34	26.71	0	D200m80	14.24	391.23	1661
C150m60	3.90	94.93	0	D250m50	13.10	211.64	4712
C200m40	4.73	86.30	0	D250m100	33.81	1105.07	10790

Table 3.3. Comparison between the computational times and the best results of the Tabu Search with long term memory (XTS) and of the best performing GRASP with path-relinking T3E2 on (part of) benchmark B_1 (time in seconds, corrected to account for the different machines employed).

Results for benchmark B_2

The results on benchmark B_2 can be compared in a more complete way. In fact, all the best known results on them have been obtained by the six GRASP algorithms G3-G8 Silva et al. (2004), and both the values of the best solutions found and the corresponding computational times are available. We first compare our best results with the best in the literature, then the results of our long term Tabu Search with the best of the six competitors (G3) and with the fastest (G5).

For 15 instances out of 20, the best result reported in the literature is equalled; for the remaining 5 instances, it is improved. Notice that in Silva et al. (2004) a best known result equal to 58 605 is reported for instance *n500m150*. The rest of the literature, including more recent papers such as Andrade et al. (2005), report a best known result equal to 56 572, which we also obtain. Since our ongoing work on semidefinite upper bounds for the *MDP* proves that the optimum cannot be higher than 57 663, we interpret that as a spurious value.

The first column of Table 3.4 contains the names of the instances. The following two columns report the result and the computational time in seconds referring to our three versions of Tabu Search. The following two columns report the result and the computational time in seconds referring to the 6 algorithms G3-G8. The best result for each instance is bolded. Since the computational times reported in Silva et al. (2004) refer to the average of three runs, and we consider here the best result over three runs, we have multiplied those times by 3 before reporting them in Table 3.4. Moreover, in order to make them comparable to our computational times, we have divided them by 2.15, which is the ratio of the CPU frequencies of the different machines employed (G3-G8 run on a PC AMD Athlon 1.3 GHz with 256 MB of RAM). The last column in the table reports the difference between our result and the best known one. The computational times show a huge difference: the six GRASP algorithms are, all together, from 180 to 350 times slower than our three Tabu Search algorithms, even if they have been corrected to keep into account the slower machine employed. Similar, and often larger, ratios hold for the instances not considered in the table.

Table 3.5 compares the long term Tabu Search to algorithm G3 and G5. The first column reports the name of the instances. The following two columns provide the result and the computational time in seconds of our long term memory Tabu Search. The two following couples of columns provide the same information for algorithms G3 and G5. The choice is motivated by the fact that G3 is the algorithm achieving the largest number of best results (12 over 20), whereas G5 is the fastest. For both algorithms, we provide the best results obtained in three runs and multiply by 3 the average computational times reported in Silva et al. (2004). Then, we divide them by 2.15, as above.

Instance	Tabu Search		G3-G8		Δ
	z	CPU	z	CPU	
n300m90	**20 743**	166.71	20 733	120 554.7	10
n400m120	**36 317**	570.84	36 315	391 434.0	2
n400m160	**62 487**	797.90	62 483	608 614.5	4
n500m50	**7 141**	278.48	7 131	110 013.6	10
n500m100	**26 258**	718.42	26 254	458 166.3	4

Table 3.4. Comparison between the best Tabu Search results and the best results in the literature on benchmark B_2 (when different).

Instance	XTS		G3		G5	
	z	CPU	z	CPU	z	CPU
n200m40	**4 450**	5.35	4 448	884.5	4 448	442.9
n200m80	**16 225**	17.39	**16 225**	2 575.5	16 207	1 119.9
n300m30	**2 694**	13.09	**2 694**	1 365.1	2 691	576.6
n300m60	**9 689**	39.40	9 681	4 116.1	**9 689**	2 354.7
n300m90	**20 743**	69.89	20 728	8 241.8	20 640	4 571.7
n300m120	**35 881**	93.18	**35 881**	14 342.5	35 871	6 213.2
n400m40	4 651	45.20	4 648	3 881.2	**4 653**	1 686.0
n400m80	16 935	128.28	**16 956**	14 179.0	16 925	7 336.7
n400m120	**36 317**	208.22	36 315	26 819.9	36 175	14 090.1
n400m160	**62 487**	295.32	62 470	41 995.0	62 313	21 885.1
n500m50	**7 141**	94.33	7 131	7 899.9	7 130	4 096.2
n500m100	**26 258**	272.45	26 224	31 844.8	26 201	16 627.3
n500m150	**56 572**	446.43	56 563	69 329.3	(?)	38 436.3
n500m200	**97 344**	627.11	97 327	105 781.4	97 213	54 341.1

Table 3.5. Comparison between the computational times in seconds of the Tabu Search with long term memory (XTS) and of the best performing GRASP algorithms on benchmark B_2 (when different solutions are obtained).

The instances in which all three algorithms achieve the same solution are neglected. Therefore, the table presents only 14 of 20 instances. The best result on each row is bolded. In 7 cases out of 14, our algorithm performs better than both competitors, in 4 cases it equals the result of one competitor and outperforms the other, in 2 cases its performance is worse than one competitor, but better than the other. Instance *n500m150* has already been commented: the Tabu Search performs better than G3, whereas the result reported in Silva et al. (2004) for G5 must be considered as spurious. The computational time is, once again, much smaller (from 40 to 90 times smaller than G5, from 80 to 170 times smaller than G3), and it remains such for the instances not included in the table.

Robustness of the improvement phase

In order to gain further insight on the behavior of the algorithm, and in particular on the robustness of the improvement phase, we have replaced the greedy initialization procedure described in Section 2 with a purely random selection of m elements from set N.

The results have been processed by TTTplots Aiex et al. (2005), a Perl tool specifically designed to estimate the probability distribution of an algorithm to achieve a certain target value in a certain computational time. Instead of the computational time, we took into account the number of iterations, which is a more significant and machine-independent parameter. The computational time for each iteration can be easily deduced from the total computational time and the total number of iterations, since all iterations have the same complexity and the initialization phase is negligible, being a trivial random selection of m elements.

We have run the Tabu Search with long term memory (XTS) on 1 000 independently generated random solutions. For each run, we have recorded the number of iterations required to achieve a target value at least equal to 99% of the best known result. All runs on all tested instances reached the target within the limit of 2 000 iterations used in the experimental campaign. After sorting these numbers of iterations in ascending order, it is easy to determine for each positive number i the fraction of runs in which the target result has been achieved within at most i iterations. This can be interpreted as an empirical probability distributions for the number of iterations required to achieve the target value.

Figures 3.1 and 3.2, produced by TTTplots, provide this distribution, respectively, for instance *B250m50* from benchmark B_1 and instance *n400m80* from benchmark B_2. We have chosen these instances because they are among the few on which the Tabu Search with long term memory does not find the best known result. The behavior for the other instances is similar, and it is quite in accordance with the theoretical exponential distribution commonly observed in such cases. The graphs show that a large majority of the runs reach the target value much earlier than the established iteration limit, even if initialized at random. This suggests that the improvement phase is quite robust, and the initialization phase nearly irrelevant to the final result.

4 Conclusions

In this paper, we have presented a Tabu Search algorithm for the MDP, a problem with applications in a wide range of different fields. All previously proposed algorithms of some effectiveness are GRASP procedures, with very refined initialization phases and a sophisticated management of the solutions, but with a rather simple improvement phase. We have, on the

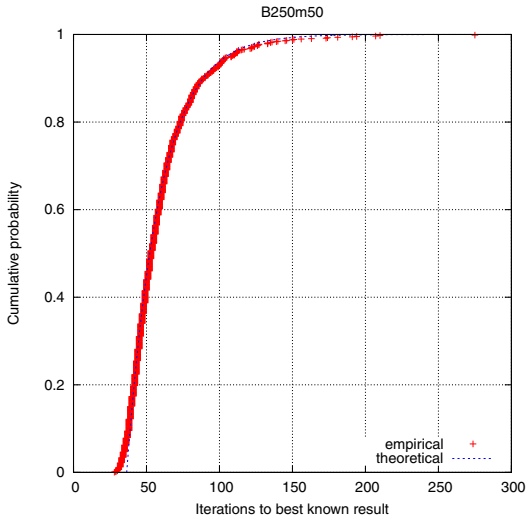

Fig. 3.1. Empirical distribution of the probability to reach the 99% of the best known result from a random starting solution on instance *B250m50* with the Tabu Search with a long term memory

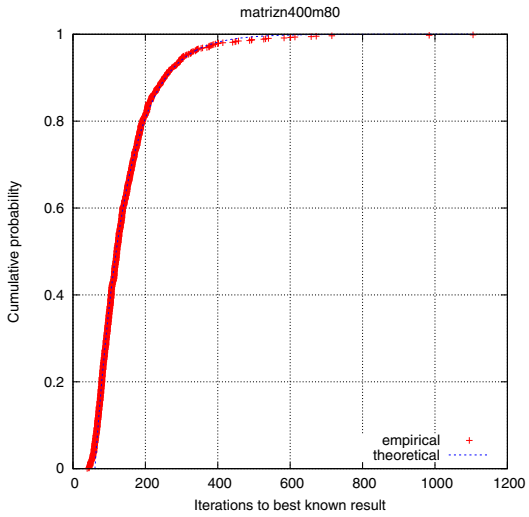

Fig. 3.2. Empirical distribution of the probability to reach the 99% of the best known result from a random starting solution on instance *n400m80* with the Tabu Search with a long term memory

contrary, adopted an extremely simple initialization procedure to focus our attention on a more effective local search. A certain number of devices, aim-

ing at a careful balance between intensification and diversification, has been added to a simple neighborhood search. Namely, a short term memory mechanism tunes the length of two tabu lists, respectively forbidding the removal of a newly added element and the inclusion of a newly removed element, and a long term memory mechanism restarts, under suitable conditions, the search from a set of promising solutions previously taken into account but not already visited. The computational experiments prove that almost all the best known results in the literature can be equalled or improved by our algorithm, which can be therefore considered more robust than the competing GRASP algorithms. Such a conclusion is also supported by a statistical analysis of the performance of the algorithm when randomly initialized. Moreover, the computational time required to obtain these results is orders of magnitude lower. Ongoing work is dedicated to develop and test further local search metaheuristics for the MDP, such as Scatter Search, Variable Neighborhood Search and Iterated Local Search. We are particularly interested in all tools specifically devoted to enrich the improvement phase, yielding a better exploration of the solution space.

References

R. M. Aiex, M. G. C. Resende, and C. C. Ribeiro. Tttplots: A perl program to create time-to-target plots. Technical Report TD-6HT7EL, AT&T Labs, 2005.

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.

P. M. D. Andrade, L. S. Plastino, and S. L. Martins. GRASP with path-relinking for the maximum diversity problem. In S. Nikoletseas, editor, *Proceedings of the 4th International Workshop on Efficient and Experimental Algorithms (WEA 2005)*, volume 3539 of *Lecture Notes in Computer Science (LNCS)*, pages 558–569. Springer–Verlag, 2005.

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

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.

J. B. Ghosh. Computational aspects of maximum diversity problem. *Operation Research Letters*, 19:175–181, 1996.

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

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.

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.

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

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

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.

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 (LNCS)*, pages 498–512. Springer–Verlag, 2004.

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.