Tabu Search vs. GRASP for the Maximum Diversity Problem

Roberto Aringhieri[*] Roberto Cordone
Yari Melzani[†]
Università degli Studi di Milano (Polo di Crema)
Dipartimento di Tecnologie dell'Informazione
Via Bramante 65, 26013 Crema (CR), Italy
E–mail: {aringhieri,cordone}@dti.unimi.it, ymelzani@crema.unimi.it

20th December 2005

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 distances between the elements of M is maximum. This problem, introduced by Glover [6], has been deeply studied using GRASP methodologies [5, 1, 13, 2]. GRASP is often characterized by a strong design effort dedicated to build high quality randomized 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 Tabu Search methodologies, 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

Let be given a set N of n elements for which a diversity measure d_{ij} is defined between each pair of elements (i, j) such that $d_{ij} > 0$ for $i \neq j$, $d_{ij} = 0$ otherwise. The *Maximum Diversity Problem* (*MDP*) consists in determining a subset $M \subset N$ of given cardinality m, such that the sum of

[*]Corresponding author
[†]This work draws origin from [12]

the pairwise distances between the 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:

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

$$\sum_{i \in N} x_i = m \tag{2}$$

$$x_i \in \{0,1\} \qquad i \in N \tag{3}$$

There are several different applications for this model. For example, while forming work teams, or 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, function d_{ij} models the distance between individuals i and j with respect to some relevant characteristics.

Other interesting applications concern the allocation of available resources for preserving biological diversity [7], medical treatment, scheduling final exams, VLSI design and data mining [10].

The problem is *strongly* $\mathcal{NP}$-hard [11]. 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 *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 vertices if and only if the optimum of *MDP* is equal to $k(k-1)/2$.

The *MDP* was introduced by Glover [6], 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 to evaluate the performance of heuristic algorithms on instances of approximately the same size [5].

Apart from some early greedy and stingy heuristics [8, 14], nearly all heuristic approaches to the *MDP* adopt the Greedy Randomized Adaptive Search Procedure methodology (*GRASP*) [4]. The first GRASP procedure for the *MDP* was proposed by Ghosh [5] obtaining good results just for instances up to 40 elements. Andrade et al. [1] 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. [13]. 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 Ghosh and Andrade et al. showing a better performance. In the end, a GRASP with path relinking has been described in [2].

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 Tabu Search methodologies, 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 [9]), providing more details only about the components which have been specifically designed for solving MDP.

Notation

For each element $i \in N$, its contribution to a given solution $M \subset N$ is defined as $D_i = \sum_{j \in M} d_{ij}$. Clearly, $z = \frac{1}{2} \sum_{i \in M} D_i$.

Simple Greedy

Let the initial solution $M^{(0)} \subset N$ be composed by the pair of elements of maximum diversity d_{ij}, i.e. $M^{(0)} = \{i, j\}$ 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

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

After each iteration, we have to update D_i for each $i \in N \setminus M^{(h)}$. This can be easily done by adding the value $d_{ik^{(h)}}$ to D_i. 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 combine m elements inside the solution with $n - m$ 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 [9]. 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 criterium*).

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 in the most recent iterations: the length of the tabu list decreases after a suitable number of improving iterations, and it increases after a suitable number of worsening iterations. A detailed description of this mechanism can be found in [3].

So, ℓ_{in} varies in the range $\left[\ell_{in}^m, \ell_{in}^M\right]$, starting from its median point $\ell_{in}^{(0)} = \left(\ell_{in}^m + \ell_{in}^M\right)/2$. After T_w consecutive worsening iterations, ℓ_{in} 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 $\ell_{out}^{(0)} = \left(\ell_{out}^m + \ell_{out}^M\right)/2$.

In the literature, the amount $\Delta\ell$ is commonly fixed to 1. Preliminary experiments showed that, when the length of the tabu list increases up to the maximum value, thus pushing the search away from the current region of the solution space, it is often difficult to reduce it in order to intensify the search in the newly reached region, because the number of improving iterations is insufficient. A complementary behavior can be observed when the length of the tabu list decreases down to the 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.

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) [3]. 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'.

3 Computational results

In this section we report the computational results of our Tabu Search algorithm and then we compare them with those obtained by various GRASP [5, 1, 13, 2]. Before discussing the computational results, we describe the computational environment, the benchmark instances used and the tuning of algorithm parameters.

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: the first benchmark, say B_1, has been proposed in [1] whilst the second one, say B_2, in [13]. These sets are also available at website: `http://www.dti.unimi.it/~aringhieri`.

Benchmark B_1 is composed of 40 instances such that n = 50, 100, 150, 200, 250 and m equals to 20% or 40% of n. There are 4 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 distances d_{ij} are random integers with a uniform distribution in $[1, 9999]$;
- type C: 50% of the distances are random integers uniformly distributed in $[1, 9999]$, whilst the remaining are random integers uniformly distributed in $[1, 4999]$;
- type C: 50% of the distances are random integers uniformly distributed in $[1, 9999]$, whilst the remaining are random integers uniformly distributed in $[5000, 9999]$;

Benchmark B_2 is composed of 20 instances such that n = 100, 200, 300, 400, 500 and m is equal to 10%, 20%, 30%, 40% of n and d_{ij} are random integers uniformly distributed in $[0, 9]$.

Preliminary computational experiments have been done in order to tune the parameters of our algorithm, i.e. the lengths of the two tabu lists and the values concerning both the short term and the long term mechanisms.

Table 1 reports the parameters' values. We remind that the initial values

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

Table 1: Parameters' values.

of ℓ_{in} and ℓ_{out} are set to the median 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:

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

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 not 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 the same values in the three algorithms. 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. [5, 1], Andrade's GRASP heuristic [1], Silva's six GRASP heuristics (named from G3 to G8) [13] and Andrade's six GRASP heuristics with path-relinking (named from T1E1 to T3E2) [2], that is 14 different algorithms.

All of these algorithms are based on the GRASP paradigm. They select one element at a time (on the basis of a greedy criterium, partly randomized) and add it to the current partial solution until this includes m elements, as required. Then, they apply some local search procedure to the resulting solution, in order to improve it. This process goes on for a suitable number of iterations. Ghosh's heuristic is a classical GRASP. Andrade's heuristic limits the choice of the new element to a suitable *Restricted Candidate List* (*RCL*). Both of them apply the basic local search technique also adopted in our algorithm. Silva's algorithms G3-G8 combine three different constructive heuristics with two different local search procedures. They tune the length of the *RCL* by a sophisticated mechanism known as *reactive GRASP*. Moreover, not all solutions generated by the constructive heuristic, but only the best out of a given number undergoes the improvement phase. The two local search techniques are the basic local search also adopted in our algorithm and an enhanced version, which first reaches a local optimum with respect to the first neighborhood and then starts exchanging two elements instead of a single one until a second local optima is reached. The six GRASP algorithms with path-relinking described in [2] combine three relinking strategies (forward, backward or mixed) with two selection strategies from the elite set (random or greedy).

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 [2]. 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 equaled the best result reported in the literature. Table 2 discusses the remaining 8 instances. The first column contains the name of the instances. The following two columns report, respectively, our result and the best known 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 is better. We remind that this comparison opposes 3 slight variants of a single algorithm to 14 variants of 4 different algorithms.

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 robust. Table 3 compares the computational times. Unfortunately,

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 2: Comparison between the best Tabu Search results and the best results in the literature (when different) on benchmark B_1.

these are available only for instances of type B, C, D and sizes 150, 200 and 250.

Instance	Long Term Tabu Search	T3E2	Instance	Long Term Tabu Search	T3E2
B150m30	1.76	318	C200m80	14.68	2 052
B150m60	6.14	1 323	C250m50	11.12	1 602
B200m40	5.78	858	C250m100	32.96	6 774
B200m80	18.90	3 684	D150m30	1.42	204
B250m50	15.99	2 577	D150m60	4.70	759
B250m100	44.61	9 519	D200m40	4.62	603
C150m30	1.34	195	D200m80	14.24	2 856
C150m60	3.90	693	D250m50	13.10	1 545
C200m40	4.73	630	D250m100	33.81	8 067

Table 3: Comparison between the computational times of the Tabu Search with long term memory and of the best performing GRASP with path-relinking T3E2 on (part of) benchmark B_1 (time in seconds).

However, these are actually the most relevant instances in the benchmark, since they are the hardest ones. The first column reports the name of the instance and the two following report the computational times in seconds for the competing algorithms. Since the best results just discussed were obtained during 3 runs of algorithm T3E2, we have multiplied by 3 the computational times reported in [2], which refer to the average of the 3 runs. It can be remarked that the Tabu Search is from 100 to 250 times faster. Of course, it is difficult to compare the computational times of algorithms running on different machines (T3E2 runs on a 550 MHz Intel Pentium III PC with 384 MB of RAM), but such a ratio is certainly not entirely due to 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 [13], and both the values of the best solutions found and the corresponding computational times are available. As for benchmark B_1, 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.

For 14 instances out of 20 we have equaled the best result reported in the literature. Table 4 discusses the remaining 6 instances. The first column contains the name of the instances. The following two columns report the result and computational time referring to our 3 versions of Tabu Search. Computational time are reported in seconds. The following two columns report the result and computational time referring to the 6 algorithms G3-G8. The best result for each instance is bolded. Since the computational times reported in [13] refer to the average of 3 runs, and we consider here the best result over 3 runs, we have multiplied those times by 3 before reporting them in Table 4. The last column reports the difference between our result and the best known one.

	Tabu Search		Literature		
Instance	z	CPU	z	CPU	Δ
matrizn300m90	**20 743**	69,89	20 733	120 554,7	10
matrizn400m120	**36 317**	208,22	36 315	391 434,0	2
matrizn400m160	**62 487**	295,32	62 483	608 614,5	4
matrizn500m50	**7 141**	94,33	7 131	110 013,6	10
matrizn500m100	**26 258**	272,45	26 254	458 166,3	4
matrizn500m150	56 572	446,43	**58 605**	1 087 398,6	-2 033

Table 4: Comparison between the best Tabu Search results and the best results in the literature (when different) on benchmark B_2 (time in seconds).

In 5 cases out of 6, we improve the best known result, though by small amounts; in a single case our performance is worse. The computational times show a huge difference: the six GRASP algorithms are, all together, from 1000 to 2500 times slower than our three Tabu Search algorithms, both for the instances considered in the table and for the other ones. Once again, it is difficult to compare the computational times of algorithms running on different machines, but such ratios are beyond doubt meaningful.

Table 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 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), while G5 is the fastest. For both algorithms, we consider the best results obtained in 3 runs and multiply by 3 the average computational times reported in [13]. 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.

	Long Term Tabu Search		G3		G5	
Instance	z	CPU	z	CPU	z	CPU
n200m40	**4 450**	5.35	4 448	1 901.7	4 448	952.2
n200m80	**16 225**	17.39	**16 225**	5 537.4	16 207	2 407.8
n300m30	**2 694**	13.09	**2 694**	2 934.9	2 691	1 239.6
n300m60	**9 689**	39.40	9 681	8 849.7	**9 689**	5 062.5
n300m90	**20 743**	69.89	20 728	17 719.8	20 640	9 829.2
n300m120	**35 881**	93.18	**35 881**	30 836.4	35 871	13 358.4
n400m40	4 651	45.20	4 648	8 344.5	**4 653**	3 624.9
n400m80	16 935	128.28	**16 956**	30 484.8	16 925	15 774.0
n400m120	**36 317**	208.22	36 315	57 662.7	36 175	30 293.7
n400m160	**62 487**	295.32	62 470	90 289.2	62 313	47 052.9
n500m50	**7 141**	94.33	7 131	16 984.8	7 130	8 806.8
n500m100	**26 258**	272.45	26 224	68 466.3	26 201	35 748.6
n500m150	56 572	446.43	56 563	149 058.0	**58 605**	82 638.0
n500m200	**97 344**	627.11	97 327	227 430.0	97 213	116 833.3

Table 5: Comparison between the computational times of the Tabu Search with long term memory and of the best performing GRASP algorithms on benchmark B_2 (time in seconds).

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 the remaining 3 cases its performance is worse than one competitor, but better than the other. The computational time is, once again, extremely smaller: from 80 to 350 times smaller, and it remains so for the instances not included in the table. Though these GRASP algorithms run on a slower machine (a PC AMD Athlon 1.3 GHz with 256 MB of RAM), the difference in the running times cannot be due to the hardware employed.

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 contrary, adopted an extremely simple initialization procedure to focus our attention on a more effective local search. A certain number of devices, aiming 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 equaled or improved by our algorithm, which can be therefore considered more robust than the competing GRASP algorithms. Moreover, the computational time required to obtain these results is orders of magnitude lower than that required by those GRASP. 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 observing the behavior of those specific tools able to enrich the improvement phase in order to better explore the solution space.

References

[1] 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.

[2] 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.

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

[4] 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.

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

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

[7] 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.

[8] 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.

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

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

[11] 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.

[12] Y. Melzani. Un algoritmo di tabu search per il maximum diversity problem. Master's thesis, DTI - Università di Milano, Luglio 2005.

[13] 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.

[14] 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.