OPTIMAL RESULTS AND TIGHT BOUNDS FOR THE MAXIMUM DIVERSITY PROBLEM

Roberto ARINGHIERI [*], Maurizio BRUGLIERI [†] Roberto CORDONE [‡]

Abstract. The *Maximum Diversity Problem* (*MDP*) consists in selecting a subset M of given cardinality out of a set N, in such a way that the sum of the pairwise diversities between the elements of M is maximum. New instances for this problem have been recently proposed in the literature and new algorithms are currently under study to solve them. As a reference for future research, this paper provides a collection of all best known results for the classical and the new instances, obtained by applying the state-of-the-art algorithms. Most of these results improve the published ones. In addition, the paper provides for the first time a collection of tight upper bounds, proving that some of these instances have been optimally solved. These bounds have been computed by a branch-and-bound algorithm based on a semidefinite formulation of the *Quadratic Knapsack Problem* (*QKP*), which is a generalization of the *MDP*.

Keywords: Maximum Diversity Problem, Tabu Search, Semidefinite Programming

1 Introduction

Given a set N of n objects, an integer number m and a matrix $D = \{d_{ij}\}$ expressing the *diversity* between each pair of objects i and j in N, the *Maximum Diversity Problem* (*MDP*) requires to find a collection $M \subset N$ of m objects such that the sum of their pairwise diversities is maximum. It is customary to assume, w.l.o.g., that the diversity matrix is symmetric ($d_{ij} = d_{ji}$ for all $i, j \in N$) and that $d_{ii} = 0$ for all $i \in N$ [11].

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

[†]Maurizio Bruglieri (maurizio.bruglieri@polimi.it) INDACO, Politecnico di Milano, via Durando 38/a, 20158 Milano, Italy

[‡]Roberto Cordone (roberto.cordone@unimi.it) DTI, Università degli Studi di Milano, via Bramante 65, 26013 Crema, Italy

This $\mathcal{NP}$-hard problem [13] has been recently tackled by a number of heuristic approaches, such as the Greedy Randomized Adaptive Search Procedure (*GRASP*) with path relinking proposed in [16] and [2], the hybrid *GRASP* proposed by [15], the Tabu Search algorithms developed in [8] and [4], the Scatter Search algorithms described in [9] and [3] and the Variable Neighborhood Search algorithm introduced in [3]. A few exact algorithms were proposed much earlier (e.g. [10]) to solve small instances. A combinatorial branch-and-bound algorithm [14] has been tested on a benchmark generated *ad hoc*, solving all instances from 10 to 50 elements and some instances up to 150 elements.

The present work is motivated by the following remarks:

1. Nearly all recent algorithms achieve identical results on the classical instances proposed in [1] and [17], whose size ranges from $n = 50$ to $n = 500$. This is a strong hint that the published results might be optimal, but no formal proof of that has ever been made available.

2. Four new sets of benchmark instances have been recently proposed [8]. The best results on these benchmarks are available only for some of the published algorithms and they refer to a limited computational time. Our experiments suggest that these results are far from being optimal and that the state-of-the-art algorithms can strongly improve them in a longer, but still reasonable, amount of time.

3. A tight upper bound on the available *MDP* benchmarks would certainly be useful to estimate the quality of the known results; to the best of our knowledge, no such bound has ever been published in the literature.

4. A frequently overlooked fact is that the *MDP* is a special case of the *Quadratic Knapsack Problem* (*QKP*), for which exact algorithms exist (mostly based on semidefinite relaxations [12]) and can provide optimal results or at least tight bounds.

Our purpose is to provide the reader with a complete collection of the best results obtained by the presently available algorithms for the *MDP*, with a collection of tight upper bounds and with some optimal results, in order to guarantee a sound reference for the evaluation and development of further algorithms on this problem. The purpose of this paper is not an experimental comparison of heuristic algorithms: such a comparison has already been done in [4] and [8], respectively on the classical and on the new instances, in both cases leading to conclude that Tabu Search has the strongest performance. The following results derive from a (truncated) branch-and-bound algorithm starting from the best known heuristic solution. The branch-and-bound algorithm described in Section 2 is an unpublished adaptation of a similar algorithm for the *QKP* [12]. It is initialized by the Tabu Search briefly introduced in Section 2.1 and described in detail in [4]. Section 3 presents the benchmark instances. All optimality proofs, all upper bounds and a large part of the heuristic solution values reported in Section 4 are unpublished.

2 A semidefinite-based branch-and-bound algorithm for the MDP

The *MDP* can be formulated in terms of an unknown symmetric square matrix $X = \{x_{ij}\}$ of size $n+1$, where $x_{ij} = 1$ if $i = j = 0$, or $i = 0$, $j > 0$ and element j belongs to solution M, or $i > 0$, $j = 0$ and element i belongs to M, or $i, j > 0$ and both elements i and j belong to M (in particular, $x_{ii} = 1$ if $i \in M$); $x_{ij} = 0$ otherwise.

$$
\begin{aligned}
z &= \max \frac{1}{2} \tilde{D} \bullet X && && (1)\\
\operatorname{rank}(X) &= 1 && && (2)\\
X &\succeq 0 && && (3)\\
x_{00} &= 1 && && (4)\\
x_{ii} - x_{0i} &= 0 && i \in N && (5)\\
\sum_{j \in N} x_{ij} &= m x_{ii} && i \in N && (6)\\
\sum_{j \in N} (x_{jj} - x_{ij}) &= m(1 - x_{ii}) && i \in N && (7)
\end{aligned}
$$

Here $\bullet$ denotes the Frobenius product between matrices and $\tilde{D}$ is the matrix of size $n+1$ obtained from D adding null vectors as row 0 and column 0, so that objective (1) is the total diversity. Constraints (2), (3) and (4) guarantee that $X = \binom{1}{x}\begin{pmatrix}1 & x^T\end{pmatrix}$ for a suitable column vector x, since $\operatorname{rank}(X)$ is the rank of matrix X and $X \succeq 0$ means that matrix X is positive semidefinite. Vector x may be interpreted as the incidence vector of solution M; in fact, constraints (5) guarantee that the entries of X are all binary.

Constraints (6) and (7) represent the cardinality requirement. A simpler expression of this requirement would be $I \bullet X = m+1$, where I is the identity matrix of size $n+1$. This expression clearly shows that the *MDP* is a special case of the *Quadratic Knapsack Problem* [12], in which the weight coefficient matrix is set equal to the identity matrix I, while the inequality knapsack constraint is replaced by an equality one without loss of generality (all diversities are ≥ 0). Constraints (6) and (7) are, however, much tighter than $I \bullet X = m + 1$. Most other valid inequalities for the *Quadratic Knapsack Problem* [12] are useless for this special case and the corresponding formulations collapse to (1)-(7).

An upper bound can be obtained by relaxing the non-convex rank constraint (2) and solving the resulting problem with a semidefinite programming solver. We used CSDP 5.0 [5].

Our branch-and-bound algorithm applies a best-bound visit strategy (always process the open node with the highest upper bound). Branching occurs by selecting the element i^* such that

$$i^* = \arg\min_{i \in N} \left| x_{ii} - \frac{1}{2} \right|$$

where x_{ij} is the solution of the relaxed problem.

Two subproblems are generated by forcing i^*, respectively, out of or into the solution. In the former case, the problem is trivially reduced by removing the row and column referring to element i^* from the data and from the variable matrix X. In the latter case, element i^* is again removed, but the following updates must also be performed:

- constant m is replaced by $m' = m - |M'|$, where M' is the set of elements currently forced into the solution;
- the diversities d_{ij} are replaced by auxiliary coefficients:

$$d'_{ij} \leftarrow d_{ij} + \frac{\sum_{k \in M'} (d_{ik} + d_{jk})}{m'}$$

- an offset term equal to $\frac{1}{2} \sum_{h,k \in M'} d_{hk}$ is added to the solution of the reduced problem.

Any feasible solution of the reduced problem corresponds to a solution of the original problem, which has the same value thanks to the cardinality constraint, to the auxiliary coefficients (which take into account the diversities with respect to the already fixed elements) and to the offset term (which takes into account the total diversity between the fixed elements).

The branch-and-bound starts with the best solution obtained by the *XTS* algorithm described in Section 2.1 (for further details see [4]), apart from a small number of cases in which a better value is known from the literature. In these cases, the best known value is used (see Section 4 for further details). A heuristic solution, which might improve the current lower bound, is also derived at each node from the solution X of the semidefinite relaxation, by selecting the m elements with the largest diagonal terms x_{ii} (except, of course x_{00}).

2.1 An eXploring Tabu Search (XTS)

A trivial greedy heuristic builds a starting solution by initializing M with the pair of elements of maximum diversity d_{ij} and iteratively adding the element i^* with the maximum contribution Δ_{i^*} to M. This solution is improved by a Tabu Search based on the exchange between an element s in the solution and an element t out of it ($M \leftarrow M \cup \{t\} \setminus \{s\}$) [4]. The non-tabu move yielding the largest improvement in the objective function is applied, but a move improving the best known solution is always performed, even if it is tabu (*aspiration criterion*).

We define two independent tabu lists: the former forbids a recently removed element to enter the solution for ℓ_{in} iterations, whilst the latter forbids a recently included element to exit for ℓ_{out} iterations. The lengths of both tabu lists (*tabu tenure*) decrease if the objective function has improved in the most recent iterations. This intensifies the search in the regions which provide improving solutions, and therefore

appear more promising. The tabu tenure increases if the objective function has worsened, in order to speed up the leaving from those regions which provide worsening solutions, and therefore probably surround an already visited local optimum.

The algorithm also adopts a long term memory mechanism known as eXploring Tabu Search (*XTS*) [6]. The basic idea is to maintain a set of good solutions $\mathcal{M}$ of fixed cardinality. When exploring a neighborhood, 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 search restarts from the best solution in $\mathcal{M}$ every time either the best known solution is not improved for a suitable number of iterations (assuming the currently explored region to be unpromising) or the length of one of the two tabu lists resides in the upper half of its range for a suitable number of consecutive iterations (assuming the short term mechanism to be insufficient to diversify the search). We adopted the same set of parameters published in [4], increasing the total number of iterations from 3 000 to 100 000.

3 Benchmark instances

At present, the following five sets of benchmark instances for the *MDP* are publicly available at web sites `http://www.uv.es/~rmarti/paper/results/mmdp%20instances.zip` and `http://www.di.unito.it/~aringhie/benchmarks.html`:

- benchmark APOM [1] 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 [17], 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 GKD [8] 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.
- benchmark DM1 [8], consists of 60 instances with real numbers generated from a $[0, 10]$ uniform distribution; 20 instances have $n = 500$ and $m = 200$, 20 instances have $n = 2000$ and $m = 200$, 20 instances have $n = 500$ and $m = 50$;
- benchmark DM2 [8], consists of 20 instances with real numbers generated from a $[0, 1000]$ uniform distribution; they have $n = 500$ and $m = 50$

4 Reference results

This section reports the results obtained by the branch-and-bound algorithm on all 160 publicly available instances of the *MDP* with a time limit of 24 hours of computation for each instance. The upper bounds reported correspond either to the optimum value (for the instances which could be solved exactly) or to the largest upper bound of the branching nodes still open at the end of the time limit. The lower bounds correspond to the heuristic solutions provided by algorithm *XTS*, with a few exceptions (specified in the following), in which they correspond to a better result available from the literature. In all instances of the benchmark, the branch-and-bound was unable to improve these starting solutions.

As for the 60 instances of the classical benchmarks *APOM* and *SOM*, the algorithm was able to prove for 15 of them (all belonging to benchmark *APOM*) the optimality of the previously best known result. For the other 45, it could only confirm these results and provide an upper bound (with an average gap equal to 3.58%). Tables 1 and 2 report the results on these benchmarks. The first column reports the name of each instance, the next two ones the upper and lower bound on the optimum obtained by the branch-and-bound algorithm (a single value when optimum), the last one the percent gap between them (symbol "−" means that the value is optimum, while "0.00%" means that the gap is lower than 0.005%). For all these instances, the starting solution (which coincides with the final one) was provided by algorithm *XTS*, with three exceptions. The values for instances *b200m80* and *b250m100* are drawn from [2], where they are attributed to previous algorithms. On these instances, both *XTS* and the branch-and-bound could only confirm the values obtained in [2], which are lower by 1 and 2, respectively, than the best known ones. The solution for instance *b250m50* was obtained by the Scatter Search and the Variable Neighbourhood Search described in [3], but also some variants of *XTS* could obtain it.

An interesting remark about these result is that the instances with a smaller ratio m/n between the number of chosen elements and the total cardinality of the set prove consistently harder to solve, and exhibit a larger gap.

The best known results for the 20 instances of benchmark *GKD*, obtained by the Tabu Search described in [8] and confirmed by *XTS* could not be improved by the branch-and-bound algorithm. Table 3 reports them, together with the corresponding upper bounds. These show that they appear to be easy instances. In fact, though none of these instances could be solved exactly, the gap is always negligible. The table has the same structure as the previous ones, but it reports the absolute gap instead of the relative one, because the latter is always $< 0.01\%$.

For the new benchmarks *DM1* and *DM2*, *XTS* was able to improve the best known result in 65 cases out of 80 (in particular, 19 of the 20 instances of benchmark *DM1* with $n = 2000$) and to confirm it in 5 cases. In 10 cases out of 80, *XTS* could neither improve nor equal the best solution known from the literature, which always derives from the Tabu Search described in [8]. Since the aim of this work was specifically to provide the best possible reference results for further research on this problem, we have decided to initialize the branch-and-bound algorithm with the best known value from [8] in these 10 cases.

Instance	UB	LB	Gap (%)
a050m10	491.9		-
a050m20	1931.6		-
a100m20	2007.1		-
a100m40	7730.1		-
a150m30	4552.1		-
a150m60	17482.4		-
a200m40	8132.1		-
a200m80	31048.5		-
a250m50	12654.0		-
a250m100	48384.3		-
b050m10	334976		-
b050m20	1171416		-
b100m20	1313134	1267277	3.62%
b100m40	4586165	4544642	0.91%
b150m30	2870909	2758381	4.08%
b150m60	10078202	9960461	1.18%
b200m40	5004846	4788086	4.53%
b200m80	17733881	17544448	1.08%
b250m50	7737888	7389784	4.71%
b250m100	27560109	27168460	1.44%
c050m10	316409		-
c050m20	1094343		-
c100m20	1210319	1207522	0.23%
c100m40	4219641	4219476	0.00%
c150m30	2657611	2613286	1.70%
c150m60	9377502	9374611	0.03%
c200m40	4719609	4630545	1.92%
c200m80	16772509	16759895	0.08%
c250m50	7357973	7178043	2.51%
c250m100	26083700	26047022	0.14%
d050m10	381666	381379	0.08%
d050m20	1502908		-
d100m20	1620542	1570800	3.17%
d100m40	6082499	6067776	0.24%
d150m30	3664656	3502567	4.63%
d150m60	13673082	13611262	0.45%
d200m40	6500167	6207580	4.71%
d200m80	24244853	24133321	0.46%
d250m50	10163567	9685430	4.94%
d250m100	37917432	37753120	0.44%

Table 1: Results on the APOM instances

Instance	UB	LB	Gap (%)
n100m10	369	333	10.81%
n100m20	1228	1195	2.76%
n100m30	2465	2457	0.33%
n100m40	4176	4142	0.82%
n200m20	1475	1247	18.28%
n200m40	4679	4450	5.15%
n200m60	9605	9437	1.78%
n200m80	16425	16225	1.23%
n300m30	3136	2694	16.41%
n300m60	10220	9689	5.48%
n300m90	21265	20743	2.52%
n300m120	36313	35881	1.20%
n400m40	5405	4658	16.04%
n400m80	17833	16956	5.17%
n400m120	37157	36317	2.31%
n400m160	63346	62487	1.37%
n500m50	8200	7141	15.01%
n500m100	27353	26258	4.17%
n500m150	57615	56572	1.84%
n500m200	98411	97344	1.10%

Table 2: Results on the SOM instances

Instance	UB	LB	Gap
1	19 485.6	19 485.2	0.4
2	19 702.8	19 701.6	1.2
3	19 547.3	19 547.3	< 0.05
4	19 596.5	19 596.5	< 0.05
5	19 603.0	19 602.6	0.4
6	19 421.7	19 421.6	0.1
7	19 534.1	19 534.0	0.1
8	19 488.4	19 487.4	1.0
9	19 221.7	19 221.7	< 0.05
10	19 703.4	19 703.4	< 0.05
11	19 587.9	19 587.1	0.8
12	19 360.7	19 360.3	0.4
13	19 366.8	19 366.8	< 0.05
14	19 458.6	19 458.6	< 0.05
15	19 423.6	19 422.2	1.4
16	19 680.1	19 680.1	< 0.05
17	19 331.5	19 331.5	< 0.05
18	19 461.3	19 461.3	< 0.05
19	19 477.4	19 477.4	< 0.05
20	19 604.9	19 604.9	< 0.05

Table 3: Results on the GKD instances

Instance	UB	LB	Gap (%)
1	108 539.3	107 394.8*	1.07%
2	108 366.5	107 156.0*	1.13%
3	108 323.0	107 260.4	0.99%
4	108 256.7	107 010.9	1.16%
5	108 102.5	106 944.6	1.08%
6	108 364.6	107 164.5*	1.12%
7	108 211.7	107 079.4	1.06%
8	108 259.1	107 038.3	1.14%
9	108 497.0	107 482.7	0.94%
10	108 404.2	107 265.8	1.06%
11	108 305.8	107 181.6	1.05%
12	108 045.7	106 853.5	1.12%
13	108 771.5	107 647.3	1.04%
14	108 502.1	107 423.7	1.00%
15	108 265.1	107 038.3	1.15%
16	108 597.0	107 420.7	1.10%
17	108 317.0	107 101.9	1.13%
18	108 201.4	106 982.1	1.14%
19	108 221.7	107 036.9	1.11%
20	108 015.4	106 815.7	1.12%

Table 4: Results on the DM1 instances with $n = 500$ and $m = 200$

Tables 4 to 7 report the results on these benchmarks. Their structure is the same as in the previous tables. The lower bounds always correspond to the initial solution, and it is marked by an asterisk (*) in the 10 cases in which it derives from the literature [8] and by an apex ($'$) in the 5 cases in which the result found by *XTS* equalled the previous one. The other 65 best known results are new; they were all obtained by *XTS* and the branch-and-bound could not improve them. The computational time is always 24 hours and it includes the time required by *XTS*, which if of course negligible. The interested reader can easily compute a quite precise approximation of the computational time for *XTS* as $\gamma Im\,(n - m)$, where $I = 100\,000$ is the total number of iterations and γ is a machine-dependent coefficient. In our campaign, we used a 2.2 GHz 2 Dual Core AMD Opteron(tm) Processor 275, with 3 GB of memory and 250 *GB* of hard disk. For this machine, $\gamma \approx 1.05 \cdot 10^{-8}$ seconds, so that the time ranges from 24 seconds (for $n = 500$ and $m = 50$) to 380 seconds (for $n = 2\,000$ and $m = 200$). The time required to obtain the best result is on average much lower.

As well as the classical benchmarks *APOM* and *SOM*, the new benchmarks prove consistently harder when the ratio m/n is smaller: the gap is about 13% when $n = 500$ and $m = 50$ and 9% when $n = 2\,000$ and $m = 200$, versus a gap about 1% when $n = 500$ and $m = 200$.

Though the purpose of this paper is not an experimental comparison of heuristics, we can make some further remarks on the two Tabu Search algorithms which provide the starting solutions. The one proposed in [8] (see the conclusions in the original paper) implements a straightforward short-term memory function, with the aim to

Instance	UB	LB	Gap (%)
1	124 527.0	113 935	9.30%
2	124 516.9	113 910	9.31%
3	124 509.5	113 932	9.28%
4	124 477.9	113 539	9.63%
5	124 513.0	113 798	9.42%
6	124 545.0	113 961*	9.29%
7	124 512.7	113 873	9.34%
8	124 515.3	113 943	9.28%
9	124 537.4	114 070	9.18%
10	124 543.2	113 999	9.25%
11	124 551.2	113 994	9.26%
12	124 551.8	114 047	9.21%
13	124 485.2	113 849	9.34%
14	124 544.2	114 037	9.21%
15	124 522.8	114 036	9.20%
16	124 450.2	113 831	9.33%
17	124 578.5	114 078	9.20%
18	124 491.0	114 202	9.01%
19	124 473.9	113 967	9.22%
20	124 672.4	113 968	9.39%

Table 5: Results on the DM1 instances with $n = 2000$ and $m = 200$

Instance	UB	LB	Gap (%)
1	8 846.4	7 833.8′	12.92%
2	8 790.3	7 754.9*	13.35%
3	8 813.8	7 757.1	13.62%
4	8 841.4	7 765.1	13.86%
5	8 785.7	7 755.2	13.29%
6	8 808.6	7 773.7	13.31%
7	8 807.4	7 752.7*	13.60%
8	8 780.4	7 741.0	13.43%
9	8 810.8	7 760.7	13.53%
10	8 844.7	7 778.7*	13.70%
11	8 822.8	7 771.0	13.54%
12	8 809.8	7 757.7*	13.56%
13	8 837.8	7 785.7	13.51%
14	8 802.2	7 795.6	12.91%
15	8 844.8	7 725.9	14.48%
16	8 840.0	7 792.8′	13.44%
17	8 815.6	7 787.2	13.21%
18	8 819.3	7 756.3′	13.71%
19	8 826.9	7 755.4′	13.82%
20	8 787.6	7 733.9′	13.63%

Table 6: Results on the DM1 instances with $n = 500$ and $m = 50$

Instance	UB	LB	Gap (%)
1	884 050.7	778 030.6	13.63%
2	881 235.1	779 963.8*	12.98%
3	879 991.5	776 471.4*	13.33%
4	884 152.0	775 394.5	14.03%
5	881 461.8	775 611.4*	13.65%
6	878 124.3	775 153.6	13.28%
7	882 925.0	776 716.3	13.67%
8	886 052.8	779 168.6	13.72%
9	879 887.5	774 801.8*	13.56%
10	883 543.4	774 393.9	14.09%
11	881 098.8	777 468.8	13.33%
12	879 054.3	775 492.9	13.35%
13	879 878.1	780 191.9*	12.78%
14	888 410.8	782 232.7	13.57%
15	884 916.8	780 300.6*	13.41%
16	879 427.5	775 436.2	13.41%
17	881 565.9	775 292.9	13.71%
18	881 016.8	775 735.5	13.57%
19	884 906.8	778 802.8*	13.62%
20	878 128.9	778 644.7	12.78%

Table 7: Results on the DM2 instances

produce high quality solutions in short computation time. The published results have been obtained in 10 seconds of computation on a Pentium IV at 3GHz with 1GB of RAM. The purpose of *XTS* is somewhat complementary: it exploits more refined memory mechanisms to produce nearly optimal solutions in a reasonable time. Consequently, its computational time is larger (from 24 to 380 seconds, though on a different machine). In an attempt to compare the two algorithms on an equal-time basis, we ran *XTS* for a fixed time determined by multiplying the time limit used in [8] by a coefficient derived from Dongarra [7] in order to roughly account for the different performance of the two machines employed. The results turned out to be scarcely significant: as a matter of fact, in a large number of instances, different runs of *XTS* obtained significantly different results. This is due to the fact that the number of iterations performed in a fixed time limit cannot be rigorously fixed and that a time limit comparable to the one used in [8] terminates *XTS* when it has still a relevant potential for improvement. Though this is not explicitly stated in the original paper, it is a rational assumption that the 10-second time limit for the simpler Tabu Search exceeds, as it is usual, the time of the last improvement, and therefore that algorithm is not affected by such a phenomenon. The comparison is further impoverished by the fact that Dongarra's coefficient is necessarily approximated and therefore the the correct time limit could be different. Our conclusion is that, given the different time scales for which the two algorithms are conceived, they should be considered as different tools for different purposes, hardly comparable at all.

Acknowledgments

The authors wish to thank Mr. Alberto Ghilardi for his contribution to the organization of the experimental campaign.

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*, pages 558–569. Springer Berlin / Heidelberg, 2005.

[3] 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.

[4] 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.

[5] B. Borchers. *CSDP 5.0 User Guide*, September 2005.

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

[7] J. J. Dongarra. Performance of various computers using standard linear equations software. Report CS-89-85, Computer Science Department, University of Tennesse, 2008.

[8] A. Duarte and R. Martì. Tabu search and GRASP for the maximum diversity problem. *European Journal of Operational Research*, 178(1):71–84, April 2007.

[9] M. Gallego, A. Duarte, M. Laguna, and R. Martì. Hybrid heuristics for the maximum diversity problem. Technical report, University of Valencia, June 2006.

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

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

[12] C. Helmberg, F.Rendl, and R. Weismantel. A semidefinite programming approach to the quadratic knapsack problem. *Journal of Combinatorial Optimization*, 4(2):197–215, 2000.

[13] 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.

[14] R. Martì, M. Gallego, and A. Duarte. A branch and bound algorithm for the maximum diversity problem. *European Journal of Operational Research*. [in press].

[15] 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.

[16] 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.

[17] 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.

Received April, 2009