New Heuristics for the Maximum Diversity Problem

Geiza C. Silva, Marcos R. Q. de Andrade, Luiz S. Ochi, Simone L. Martins, and Alexandre Plastino

Universidade Federal Fluminense, Departamento de Ciência da Computação
Rua Passo da Pátria, 156 – Bloco E – 3^o andar – Boa Viagem
24210-240, Niterói, RJ, Brazil
{gsilva, mandrade, satoru, simone, plastino}@ic.uff.br

Abstract. The Maximum Diversity Problem (MDP) consists in identifying, in a population, a subset of elements, characterized by a set of attributes, that present the most diverse characteristics between themselves. The identification of such solution is an NP-hard problem. Some heuristics are available to obtain approximate solutions for this problem. In this paper we propose different GRASP heuristics for the MDP, using distinct construction procedures and path-relinking techniques. Performance comparison between related work and the proposed heuristics is provided. Experimental results show that the new GRASP heuristics are quite robust and are able to find high-quality solutions in a feasible computational time.

1 Introduction

Consider $P = \{p_1, \ldots, p_n\}$ a set of elements, defined over the index set $N = \{1, 2, ..., n\}$, and p_{ik}, $k \in L = \{1, \ldots, l\}$, the l attributes associated with each element p_i. The maximum diversity problem (MDP) [3, 5, 6] consists in identifying a subset M from the population P, so that the m elements from M present the maximum possible diversity among them. The measure of diversity d_{ij} between a pair of elements (i, j) is calculated by a function applied on their attributes. This problem can be formulated as:

Maximize

$$z = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_i x_j, \tag{1}$$

subject to

$$\sum_{i=1}^{n} x_i = m, \tag{2}$$

where x_i is a binary variable indicating if an element i is selected to be a member of the subset M.

Many applications [7] can be solved using the resolution of this problem, such as human resource management, measure of biodiversity, and VLSI design.

Glover et al. [5] presented mixed integer zero-one formulation for this problem, that can be used to solve small instances by exact methods. They also show that this problem belongs to the class of NP-hard problems.

Some heuristics are available to obtain approximate solutions. Weitz and Lakshminarayanan [14] developed five heuristics to find groups of students with the most possible diverse characteristics, such as nationality, age and graduation level. They tested the heuristics using instances based on real data and implemented an exact algorithm for solving them.

Constructive and destructive heuristics were presented by Glover et al. [6], who created instances with different sizes of population (maximum value was 30) and showed that the proposed heuristics obtained results close (2%) to the ones obtained by the exact algorithm, but much faster.

Ghosh [3] proposed a GRASP (Greedy Randomized Adaptive Search Procedure) that obtained good results for small instances of the problem.

In this work, we initially present two new GRASP heuristics for the MDP, using distinct construction procedures. These proposals are developed using the concept of reactive GRASP introduced by Prais and Ribeiro [9]. The proposed GRASP strategies are compared to previous work based on computational experiments for instances created with maximum population of 500 individuals. Thereafter, motivated by the good results achieved, we also propose an improvement on the GRASP heuristic that obtained the best results. We introduce a path-relinking technique into this best GRASP heuristic as an intensification mechanism.

This paper is organized as follows. In Section 2 we describe the GRASP proposed by Ghosh [3] and present the new GRASP heuristics developed. Based on computational experiments, these strategies are then compared. In Section 3 we create another GRASP heuristic combining a path-relinking technique with the pure GRASP proposed in the previous section which presented the best behavior. The performance comparison of all these strategies is presented in Section 4. Concluding remarks are presented in Section 5.

2 GRASP Heuristics

GRASP [2] is an iterative process, where each iteration consists of two phases: construction and local search. In the construction phase a feasible solution is built, and its neighborhood is explored by a local search. The result is the best solution found over all iterations.

The construction phase of GRASP is an iterative process where, at each iteration, the elements $c \in C$ that do not belong to the solution are evaluated by a greedy function $g : C \rightarrow \Re_+$, that estimates the gain of including it in the partial solution. They are ordered by their estimated value in a list called restricted candidate list (RCL) and one of them is randomly chosen and included in the solution. The size of the RCL is limited by a parameter α. For a maximization problem, only the elements whose g values are in the range $[(1-\alpha)g_{max}, g_{max}]$

are placed in RCL. This iterative process stops when a feasible solution is obtained.

The solutions generated by the construction phase are not guaranteed to be locally optimal. So a local search is performed to attempt to improve each constructed solution. It works by successively replacing the current solution by a better one from its neighborhood, until no more better solutions are found.

In Subsection 2.1, we describe the GRASP proposed by Ghosh [3]. In Subsection 2.2, we present two new GRASP procedures with different construction phases for the maximum diversity problem.

2.1 Ghosh Algorithm

Ghosh [3] proposed a GRASP heuristic for the MDP and tested it for small instances. The construction phase consists of m iterations. For each iteration k, M_{k-1} is a partial solution with $k-1 (1 \leq k \leq m)$ elements. The element i^* to be inserted in each iteration is selected based on its contribution to the overall diversity $z(M)$. First, a lower bound $\Delta z_L(i)$ and an upper bound $\Delta z_U(i)$ are computed for all $i \in N \setminus M_{k-1}$. Then a random number u is sampled from a uniform distribution $U(0,1)$ that is used to compute $\Delta z'(i) = (1-u)\Delta z_L(i) + u\Delta z_U(i)$. The selected element i^* is the one that presents the larger $\Delta z'$ and is included in M_{k-1} to obtain M_k. The computations of $\Delta z_L(i)$ and $\Delta z_U(i)$ are:

$$\Delta z_L(i) = \sum_{j \in M_{k-1}} d_{ij} + \sum_{n-m+1 \leq r \leq n-k} d_i^r(Q_{ik});$$

$$\Delta z_U(i) = \sum_{j \in M_{k-1}} d_{ij} + \sum_{1 \leq r \leq m-k} d_i^r(Q_{ik});$$

where $d_i^r(Q_{ik})$ is the r^{th} largest distance in $\{d_{ij} : j \in Q_{ik}\}, Q_{ik} = N \setminus (M_{k-1} \cup \{i\})$.

The lower bound $\Delta z_L(i)$ is computed by adding the distances between i and the elements that are already in the solution to the sum of distances between i and the $m-k-1$ elements that are not in solution M_{k-1} and which have smaller distances to i. The upper bound $\Delta z_U(i)$ is computed by adding the same first term of $\Delta z_L(i)$ to the sum of distances between i and the $m-k-1$ elements that are not in solution M_{k-1} and which have larger distances to i.

After a solution is constructed, a local search phase should be executed in trying to improve the initial solution. The neighborhood of a solution defined by Ghosh [3] is the set of all solutions obtained by replacing an element in the solution by another that does not belong to the set of its elements. The incumbent solution M is initialized with the solution obtained by the construction phase. For each $i \in M$ and $j \in N \setminus M$, the improvement due to exchanging i by j, $\Delta z(i,j) = \sum_{u \in M_{\{i\}}} (d_{ju} - d_{iu})$ is computed. If for all i and j, $\Delta z(i,j) < 0$, the local search is terminated, because no exchange will improve z. Otherwise, after the elements of the pair (i,j) that provides the maximum $\Delta z(i,j)$ are interchanged, a new incumbent solution M is created and the local search is performed again.

2.2 New Algorithms

Usually, the local search phase demands great computational effort and execution time, so the construction phase plays an important role to diminish this effort by supplying good starting solutions for the local search. Next, we propose two construction procedures which are combined with the local search developed by Ghosh, obtaining two new GRASP procedures.

These new construction procedures are based on two existing techniques: the widely used filtering of constructed solutions [10] and reactive GRASP proposed by Prais and Ribeiro [9].

Filtering of constructed solutions leads to a more greedy construction. For each GRASP iteration, the construction algorithm is executed X times generating X solutions and only the best solution is selected to be used as the initial solution for the local search phase.

Prais and Ribeiro [9] proposed the procedure called reactive GRASP, for which the parameter α used in the construction phase is self adjusted for each iteration. For the first construction iteration, an α value is randomly selected from a discrete set $A = \{\alpha_1, \ldots, \alpha_m\}$. Each element α_i has a probability p_i associated and, initially, a uniform distribution is applied, thus we have $p_i = 1/m, i = 1, \ldots, m$. Periodically the probability distribution $p_i, i = 1, \ldots, m$ is updated using information collected during the former iterations. The aim is to associate higher probabilities to values of α that lead to better solutions and lower ones to values of α that guide to worse solutions.

The two new GRASP algorithms described below use both techniques described before.

KLD (K Largest Distances) This algorithm constructs an initial solution by randomly selecting an element from a RCL of size K at each construction iteration. The RCL is created by selecting, for each element $i \in N$, the K elements $j \in N\backslash\{i\}$ that exhibit larger values of d_{ij} and computing s_i by the sum of these K values. Then, a list is created containing all elements i sorted in descending order by their s_i values and the K first elements are selected to compose the RCL list.

The procedure developed to implement the reactive GRASP starts considering m_it to be the total number of GRASP iterations. In the first block of iterations $B_1 = 0.4m\_it$, four different values for $K \in \{K_1, K_2, K_3, K_4\}$ are evaluated by dividing the block into four equal intervals $c_i, i = 1, \ldots, 4$. The value K_i is used for all iterations belonging to interval c_i. The values of K_i are shown in Table 1, where $\mu = (n - m)/2$. After the execution of the last iteration of block B_1, the quality of the solutions obtained for each K_i is evaluated. The average diversity value $zm_i = \sum_{1 \leq q \leq 0.1m\_it} z(sol_{iq})$ for the solutions $sol_{iq}, i = 1, \ldots 4; q = 1, \ldots, 0.1m\_it$; is obtained for each K_i. The values K_i are stored in a list LK ordered by their zm_i values.

Then the next block of iterations $B_2 = 0.6m\_it$ is divided into four intervals y_i, each one with different number of iterations. For each interval a value for K

Table 1. K values for block B_1

i	c_i	K
1	$[1, \ldots, 0.1m\_it]$	$m + \mu - 0.2\mu$
2	$(0.1m\_it, \ldots, 0.2m\_it]$	$m + \mu - 0.1\mu$
3	$(0.2m\_it, \ldots, 0.3m\_it]$	$m + \mu + 0.1\mu$
4	$(0.3m\_it, \ldots, 0.4m\_it]$	$m + \mu + 0.2\mu$

is adopted as shown in Table 2. In this way, the values K_i that provide better solutions are used in a larger number of iterations.

Table 2. K values for block B_2

i	y_i	K
1	$(0.4m\_it, \ldots, 0.64m\_it]$	lk_1
2	$(0.64m\_it, \ldots, 0.82m\_it]$	lk_2
3	$(0.82m\_it, \ldots, 0.94m\_it]$	lk_3
4	$(0.94m\_it, \ldots, m\_it]$	lk_4

At each GRASP iteration, the filter technique is applied by constructing 400 solutions and only the best solution is sent to the local search procedure.

The pseudo-code, including the description of the procedure for the construction phase using K largest distances heuristic, is given in Fig. 1.

In line 1, the cost of the best solution found in the execution of max_sol_filter iterations is initialized. The value K to be used to build the Restricted Candidate List (RCL) is calculated by the procedure det_K in line 2. This procedure defines the value for K, implementing the reactive GRASP described before. In line 3, the RCL is built. From line 4 to line 15, the construction procedure is executed max_sol_filter times and only the best solution is returned to be used as an initial solution by the local search procedure. From line 6 to line 10, a solution is constructed by the random selection of an element from RCL. In lines 11 to 14, the best solution found by the construction procedure is updated. If the GRASP iteration belongs to B_1 block, the cost of the solution found using the selected K is stored in line 17. When the first block B_1 of iterations ends, the values K_i are evaluated and put in the list LK sorted in descending order, in line 20.

The local search strategy employed is the same as the one developed by Ghosh.

KLD2 This algorithm is similar to the previously described algorithm. The difference between them is the way that the Restricted Candidate List is built. In the former algorithm, the RCL is computed before the execution of the construction iterations and, for each iteration, the only modification made in the RCL is the removal of the element that is inserted in the solution.

```
procedure constr_KLD($it\_GRASP, m\_it, numsol, n, m$)
1.   $best\_cost\_sol \leftarrow 0$;
2.   $K \leftarrow det\_K(it\_GRASP, m\_it, LK)$;
3.   $RCL \leftarrow Build\_RCL(K)$;
4.   for $j = 1, \ldots, max\_sol\_filter$ do
5.     $sol \leftarrow \{\}$;
6.     for $k = 1, \ldots, m$ do
7.       Randomly select an individual $e^*$ from RCL;
8.       $sol \leftarrow sol \cup \{e^*\}$;
9.       $RCL \leftarrow RCL - \{e^*\}$;
10.    end for;
11.    if $(z(sol) > best\_cost\_sol)$ then do
12.      $sol\_constr \leftarrow sol$;
13.      $best\_cost\_sol \leftarrow z(sol)$;
14.    end if
15.  end for;
16.  if $(it\_GRASP < 0.4m\_it)$ then do
17.    $Update\_Sol\_K(K, sol\_eval, z(sol\_constr))$;
18.  end if;
19.  if $(it\_GRASP == 0.4m\_it)$ then do
20.    $LK \leftarrow Build\_LK(sol\_eval)$;
21.  end if;
22.  return $sol\_constr$.
```

Fig. 1. Construction procedure used to implement the KLD heuristic

The first element of the constructed solution is selected in the same way performed by KLD heuristic, which means that an element is randomly selected from the RCL built as described in line 3 of Fig. 1.

In KLD2, the following elements are selected from an RCL built using an adaptive procedure. Let M_c be a partial solution with $c, 1 \leq c < m$, elements and $i \in N \backslash M_c$ a candidate to be inserted in the next partial solution M_{c+1}. For each i, we select the $(K - c - 1)$ elements $j \in N \backslash (M_c \bigcup \{i\})$,that present larger values of d_{ij} and calculate the sum of the $(K - c - 1)$ values of d_{ij} obtaining s_i. To select the next element to be inserted, an initial candidate list is created based on the greedy function $gf(i)$ shown in (3), where the first term corresponds to the sum of distances from the candidate i to the elements $j \in M_c$, and the second term stands for the sum of distances from element i to the $(K - c - 1)$ elements that are not in the solution M_c and present larger distances to i. The initial candidate list is formed by the elements i, sorted in descending order with respect to $gf(i)$, and the first K elements are selected from this list to build the RCL.

$$gf(i) = \sum_{j \in M_c} d_{ij} + s_i \tag{3}$$

The Reactive GRASP and the construction filter are implemented in the same way as in KLD. Once this construction algorithm demands much more execution time than KLD algorithm, only 2 solutions, instead of 400, are generated to be filtered.

2.3 Comparing the three strategies

The computational experiments were performed on four sets (A, B, C, and D) of test problems with different characteristics [13]. An instance of a set consists of a population P with size n and a diversity matrix $DivMat$, which contains the diversity d_{ij} between elements i and j of P. The instances of a specific set are obtained from the same base diversity matrix, differing among themselves by the population size. Different base matrices are used to generate instances for each set. All sets A, B, and C contain instances with population sizes $n = 50, 100, 150, 200, 250$. The instances for set D have populations sizes $n = 100, 200, 300, 400, 500$. For all instances, tests were performed to find subsets M of sizes 10%, 20%, 30%, and 40% of the population size.

The algorithms were implemented in C and compiled with `gcc 3.3.2`. The tests were performed on a 1.4 GHz AMD Athlon with 250 Mbytes of RAM.

In Tables 3, 4, 5, and 6, the results of computing 500 iterations for each GRASP heuristic using instances A, B, C, and D are shown. The first and second columns identify two parameters of each instance: the size n of the population and the number m of elements to be selected. Each procedure was executed three times and for each one the average value of the solutions is presented. When different solutions are found, bold values indicate the best result found and underlined values represent the second best result.

The three strategies found the same result in 39 from 80 tests. In those 41 tests in which different solutions were reached, both KLD and KLD2 heuristics found better solutions than Ghosh algorithm in 30 cases. The KLD2 algorithm found 31 best solutions, while KLD obtained 25 and Ghosh found 7.

Comparing KLD and KLD2, individually to Ghosh heuristic, KLD outperformed Ghosh in 30 tests and KLD2 in 34 tests. Ghosh found better solutions than KLD in 9 tests and than KLD2 in 5 tests.

When evaluating KLD2 against KLD, we can observe that, considering the 80 conducted tests, both strategies found the same result in 58 cases, KLD2 outperformed KLD in 16 tests, and KLD found better solutions in 6 cases.

Tables 7, 8, 9, and 10 report the CPU times observed for the execution of the same sets of instances. The first and second columns identify the two parameters of each instance. For each GRASP heuristic, the average time for three executions is reported. Among the heuristics, algorithm KLD is the most efficient related to execution time. This strategy executed faster than the other two in 71 from 80 tests. Heuristic KLD2, which achieved better quality solutions, demanded more time than the other two heuristics for all instances.

Next, a deeper analysis of the results obtained for the GRASP heuristics is performed. Some tests were executed for two instances from group D: the first one has parameters $n = 200$ and $m = 40$, and the second one, $n = 300$ and $m = 90$. Each GRASP heuristic was executed until a solution was found with a greater or equal cost compared to a target value. Two target values were used for each

Table 3. Comparison of Ghosh, KLD, and KLD2 for instances from set A

n	m	Ghosh	KLD	KLD2
50	5	86733.0	86733.0	86733.0
50	10	334976.0	334976.0	334976.0
50	15	692704.0	692704.0	692704.0
50	20	1171416.0	1171416.0	1171416.0
100	10	352286.0	**353730.0**	**353730.0**
100	20	1267277.0	1267277.0	1267277.0
100	30	2674152.0	2674152.0	2674152.0
100	40	4544642.0	4544642.0	4544642.0
150	15	753041.0	**772982.0**	**772982.0**
150	30	2758381.0	2758381.0	2758381.0
150	45	**5825682.0**	5825149.5	**5825682.0**
150	60	9959068.0	**9960461.0**	**9960461.0**
200	20	1321476.6	**1330354.6**	1330146.6
200	40	**4788086.0**	**4788086.0**	4788078.0
200	60	**10211323.0**	**10211323.0**	10207669.0
200	80	17544448.0	17544448.0	17544448.0
250	25	2041049.0	**2047828.0**	2046995.4
250	50	**7388210.5**	7382340.5	7385769.5
250	75	15788177.0	15795352.0	**15801103.0**
250	100	27148842.0	**27161422.0**	27159518.0

Table 4. Comparison of Ghosh, KLD, and KLD2 for instances from set B

n	m	Ghosh	KLD	KLD2
50	5	80120.0	**84517.0**	**84517.0**
50	10	316409.0	316409.0	316409.0
50	15	659250.0	659250.0	659250.0
50	20	1094343.0	1094343.0	1094343.0
100	10	337420.0	337420.0	337420.0
100	20	1207522.0	1207522.0	1207522.0
100	30	2509045.0	2509045.0	2509045.0
100	40	4219476.0	4219476.0	4219476.0
150	15	737342.0	737342.0	737342.0
150	30	2613286.0	2613286.0	2613286.0
150	45	5519104.0	5519104.0	5519104.0
150	60	9374611.0	9374611.0	9374611.0
200	20	1294817.0	1294817.0	1294817.0
200	40	4630545.0	4630545.0	4630545.0
200	60	9811296.0	9811296.0	9811296.0
200	80	16759895.0	16759895.0	16759895.0
250	25	1983723.0	1983723.0	1983723.0
250	50	7178042.5	7178042.5	7178042.5
250	75	15303499.0	15303499.0	15303499.0
250	100	26047022.0	26047022.0	26047022.0

Table 5. Comparison of Ghosh, KLD, and KLD2 for instances from set C

n	m	Ghosh	KLD	KLD2
50	5	93007.0	93007.0	93007.0
50	10	380750.3	**381379.0**	**381379.0**
50	15	851353.0	851353.0	851353.0
50	20	1502908.0	1502908.0	1502908.0
100	10	399449.0	399449.0	399449.0
100	20	1570800.0	1570800.0	1570800.0
100	30	3475608.0	3475608.0	3475608.0
100	40	6067776.0	6067776.0	6067776.0
150	15	896529.3	**898585.0**	**898585.0**
150	30	3498361.0	**3502567.0**	**3502567.0**
150	45	7748205.0	7748205.0	7748205.0
150	60	13609447.6	**13611261.0**	**13611261.0**
200	20	1590285.3	**1595768.0**	**1595768.0**
200	40	6199891.3	**6207580.0**	**6207580.0**
200	60	13744341.6	**13749403.0**	**13749403.0**
200	80	24124858.0	**24133320.0**	**24133320.0**
250	25	2483263.0	**2488888.0**	**2488888.0**
250	50	9668626.6	9685273.0	**9685430.0**
250	75	21440150.3	**21464360.0**	**21464360.0**
250	100	37718428.0	**37753120.0**	**37753120.0**

Table 6. Comparison of Ghosh, KLD, and KLD2 for instances from set D

n	m	Ghosh	KLD	KLD2
100	10	318.0	**333.0**	**333.0**
100	20	1178.0	**1195.0**	**1195.0**
100	30	2457.0	2457.0	2457.0
100	40	4142.0	4142.0	4142.0
200	20	1233.0	**1247.0**	**1247.0**
200	40	4443.0	4446.3	**4447.0**
200	60	9437.0	9437.0	9437.0
200	80	**16225.0**	16182.3	**16225.0**
300	30	2666.0	2686.0	**2688.5**
300	60	9652.0	9667.7	**9682.5**
300	90	20725.0	20640.0	**20726.0**
300	120	**35880.0**	35871.0	35878.0
400	40	4615.0	4635.3	**4647.0**
400	80	16900.5	16916.3	**16930.0**
400	120	**36298.7**	36175.0	36272.5
400	160	62442.3	62313.0	**62478.0**
500	50	7079.3	**7124.0**	7112.5
500	100	26219.0	26197.0	**26229.0**
500	150	56571.0	**56572.0**	**56572.0**
500	200	97255.0	97213.0	**97316.5**

Table 7. Computational time for Ghosh, KLD, and KLD2 for instances from set A

n	m	Ghosh	KLD	KLD2
50	5	2.4	2.6	3.9
50	10	4.6	4.5	7.2
50	15	6.7	6.9	10.7
50	20	8.7	9.8	14.0
100	10	15.5	7.9	28.7
100	20	38.7	21.4	62.1
100	30	64.4	40.7	100.9
100	40	93.5	62.6	139.4
150	15	56.2	21.9	102.7
150	30	146.4	76.5	240.8
150	45	288.1	164.1	415.4
150	60	449.3	267.7	607.0
200	20	146.6	51.6	263.1
200	40	425.4	214.8	644.6
200	60	820.5	490.0	1149.8
200	80	1108.8	796.4	1700.1
250	25	322.5	108.7	536.5
250	50	982.7	505.3	1384.3
250	75	1854.2	1167.2	2568.6
250	100	2696.0	1885.5	3876.5

Table 8. Computational time for Ghosh, KLD, and KLD2 for instances from set B

n	m	Ghosh	KLD	KLD2
50	5	2.5	2.7	3.8
50	10	4.5	4.8	7.4
50	15	6.0	7.3	11.2
50	20	6.9	9.8	15.0
100	10	16.5	8.4	28.5
100	20	36.8	21.8	62.8
100	30	52.5	40.2	103.1
100	40	67.1	64.1	159.3
150	15	57.7	21.9	103.0
150	30	133.9	75.8	246.2
150	45	210.4	159.3	432.5
150	60	226.3	262.6	652.3
200	20	141.8	52.6	263.5
200	40	399.3	218.5	676.6
200	60	637.9	468.9	1220.2
200	80	874.7	711.4	1788.4
250	25	312.7	111.0	559.3
250	50	919.6	511.7	1497.2
250	75	1701.9	1158.9	2860.7
250	100	2472.7	1735.4	4111.5

Table 9. Computational time for Ghosh, KLD, and KLD2 for instances from set C

n	m	Ghosh	KLD	KLD2
50	5	2.3	2.3	3.6
50	10	4.1	4.0	6.7
50	15	5.7	6.1	9.8
50	20	7.5	8.9	12.8
100	10	14.3	7.6	26.3
100	20	34.4	20.6	58.7
100	30	56.9	39.1	96.9
100	40	84.0	58.7	133.4
150	15	51.0	19.6	94.7
150	30	124.8	69.1	223.2
150	45	235.7	149.3	388.5
150	60	329.1	241.9	564.1
200	20	132.8	47.8	244.9
200	40	372.4	201.8	623.0
200	60	715.4	468.5	1137.5
200	80	1159.4	716.5	1626.5
250	25	301.6	143.4	570.9
250	50	979.0	605.2	1578.5
250	75	2000.2	1291.3	2925.0
250	100	3000.2	1864.1	4273.1

Table 10. Computational time for Ghosh, KLD, and KLD2 for instances from set D

n	m	Ghosh	KLD	KLD2
100	10	18.0	10.4	28.0
100	20	47.4	28.2	64.8
100	30	88.3	57.4	111.7
100	40	130.9	89.2	165.6
200	20	147.5	71.7	263.6
200	40	561.9	318.3	784.7
200	60	1075.3	626.6	1958.8
200	80	1647.0	804.3	3051.8
300	30	712.7	412.7	1159.4
300	60	2477.2	1690.5	3450.4
300	90	5603.2	3260.6	6829.5
300	120	9740.2	4453.0	11458.5
400	40	2469.3	1208.7	3202.7
400	80	9651.9	5266.5	10464.3
400	120	20210.3	10097.5	22038.4
400	160	32217.8	15750.3	32843.6
500	50	5122.4	2934.9	6988.6
500	100	26128.9	11921.7	26513.3
500	150	54392.6	29051.4	55898.1
500	200	80252.1	39043.3	84678.4

instance: the worst value obtained by these heuristics and an average of the values generated by them. Empirical probability distributions for the time to achieve a target value are plotted in Fig(s). 2 and 3. To plot the empirical distribution for each variant, each GRASP heuristic was executed 100 times using 100 different random seeds. In each execution, the time to achieve a solution whose cost was greater or equal to the target cost was measured. The execution times were sorted in ascending order and a probability $p_i = (i - 0.5)/100$ was associated for each time t_i and the points $z_i = (t_i, p_i)$ were plotted for $i = 1, \ldots, 100$. Each point z_i in the graph represents the probability p_i of finding the target value in time t_i. This kind of analysis was first used by Aiex et al. [1].

We can observe that KLD behaves better than Ghosh and KLD2. KLD algorithm presented a higher probability of finding a target solution in a given amount of time than the others. Fig. 2 (target=4442) shows that the probability of finding the target value in 10 seconds is approximately 4% for both Ghosh and KLD2, while for KLD is 60%. Considering 100 seconds, we have the probabilities 26%, 50%, and 98% respectively.

KLD2 outperformed Ghosh algorithm in these experiments, except in one test (Fig. 3, target=20640).

3 Path-relinking

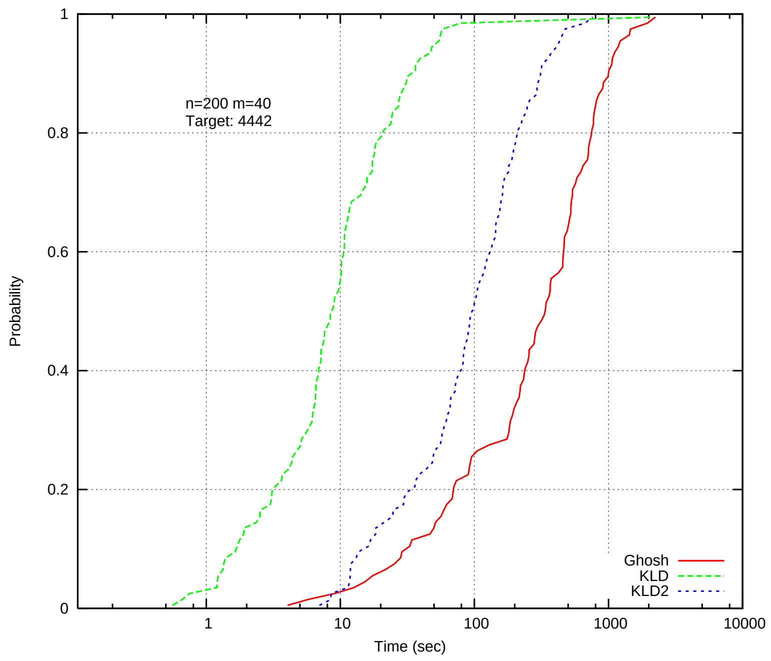
Path-relinking is a technique proposed by Glover [4] to explore possible trajectories connecting high quality solutions, obtained by heuristics like tabu search and scatter search.

The pure GRASP metaheuristic is a memoryless method, because all iterations are independent and no information about the solutions is passed from one iteration to another. The objective of introducing path-relinking to a pure GRASP algorithm is to retain previous good solutions and use them as guides in the search of new good solutions.

Laguna and Marti [8] were the first to use path-relinking within a GRASP strategy. Several extensions, improvements and successful applications of this technique can be found in the literature [11].

From the results presented in Subsection 2.3, we can see that the algorithm KLD2 presents some better results than KLD, but demands much more computational time. Besides, KLD presents a higher probability of reaching good solutions in a given amount of time. So we decided to introduce a path-relinking strategy to KLD in order to improve the trade-off between solution quality and computation time.

Path-relinking is applied to a pair of solutions $\{s_i, s_g\}$. First, the set of moves which should be executed to reach s_g (the guiding solution) starting from s_i (initial solution) is computed. At each step, all moves needed are evaluated, the one which generates the best cost solution is executed and the set of available moves is updated. This procedure terminates when s_g is reached.



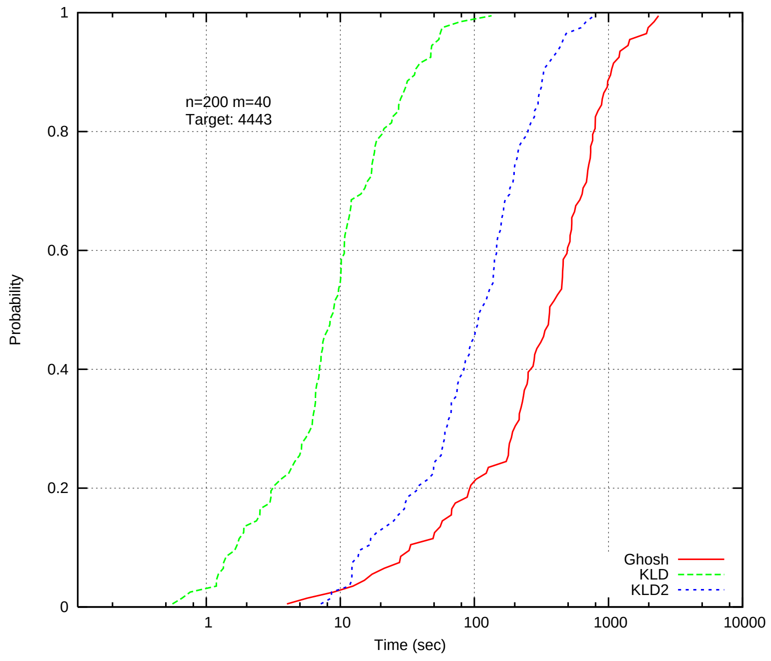
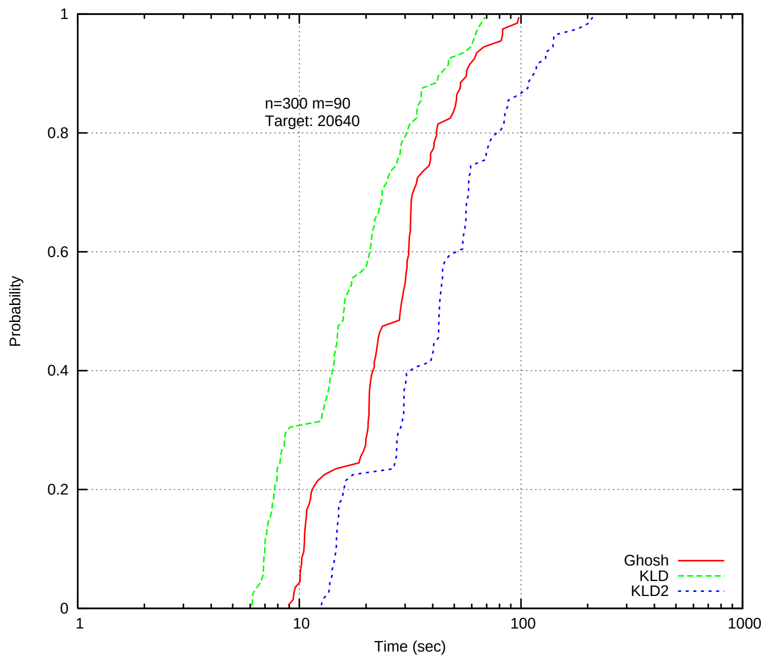


Fig. 2. Comparison of GRASP heuristics for the instance $n = 200, m = 40$ with targets values 4442 and 4443



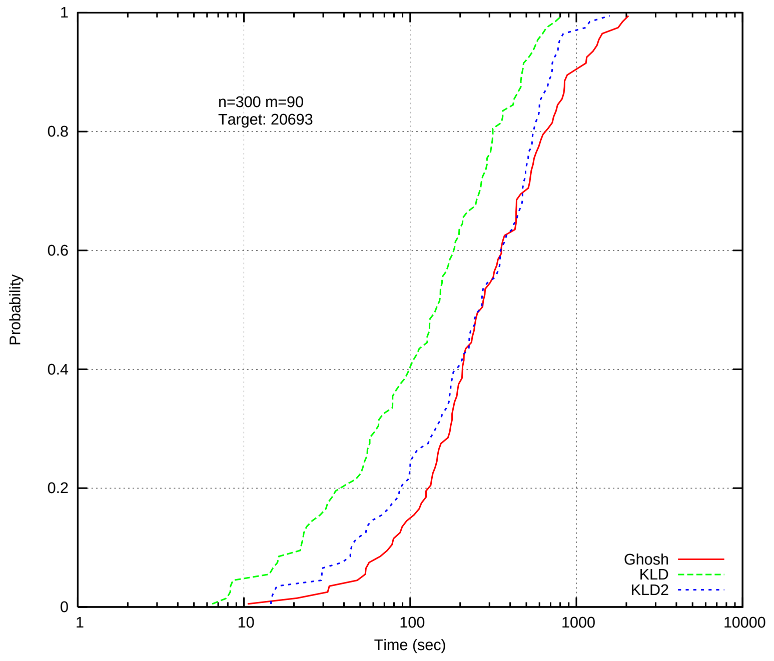


Fig. 3. Comparison of GRASP heuristics for the instance $n = 300, m = 90$ with targets values 20640 and 20693

To use path-relinking within a GRASP procedure an elite set E is maintained, in which good solutions found in GRASP iterations are stored. Two basic strategies for introducing path-relinking into GRASP may be used [11]:

– it is performed after each GRASP iteration using a solution from the elite set and a local optimum obtained after the GRASP local search.
– it is applied to all pairs of elite solutions, either periodically or after all GRASP iterations terminate.

The first strategy seems to be more effective than the second one [12], so this strategy was chosen. The elite set E is maintained, in which good solutions found in GRASP iterations are stored to be combined with solutions created by other GRASP iterations. The path-relinking is activated only after the elite set achieves its size.

For each solution Sol generated in a GRASP iteration, one of the solutions $e \in E$ is randomly selected and a path-relinking is applied between them. A path-relinking is performed by starting from an initial solution s_i and gradually incorporating attributes from a guide solution s_g to it, until s_i becomes equal to s_g. Several ways to select s_i and s_g are used [11]:

– Backward relinking: the best cost solution between Sol and e is set as the initial solution s_i and the other one as s_g.
– Forward relinking: the worst cost solution between Sol and e is set to s_i and the best one to s_g.
– Backward and Forward relinking: two different paths are explored, from s_i to s_g and from s_g to s_i.
– Mixed relinking: both trajectories are explored in alternate way. At each iteration of path relinking, the roles of initial and guide solutions are inverted.

Ribeiro et al. [12] observed that using the backward and forward strategy demands almost twice the time used to explore only one of them, and presents very small improvement in solution quality. They have also observed that best solutions are found when the relinking strategy uses the best cost solution as the initial solution s_i. The reason for this behavior is that the neighborhood of the initial solution is much more carefully explored than that of the guiding one, so there is a better chance to investigate in more detail the neighborhood of the most promising solution. Therefore, we implemented a backward path relinking strategy.

Figure 4 shows the GRASP with path-relinking developed in this work. In line 1, the instance parameters are obtained and the elite set is initialized in line 3. For each GRASP iteration a solution is constructed in line 5 and, in line 6, a local search is executed using the KLD heuristic. If the elite set is full, the path relinking should be performed. In line 8, the element of the elite set is randomly selected according to uniform probability function and, in line 9, the path-relinking is performed. In line 11, the best solution between that generated by the GRASP iteration and the one by path-relinking is evaluated to verify if it should be inserted in the elite set. If the elite set is not full and the solution is not already in the elite set, then it is inserted into it. If the elite set is full and the solution cost is better than the worst-cost elite set solution, then this new solution replaces the worst one. In lines 13 and 14 the best solution and best cost are updated.

```
procedure GRASP_PR
01.   $Read\_Input\_Data(n, m, DivMat, MaxIter)$;
02.   $z\_best \leftarrow -\infty$;
03.   $E \leftarrow \{\}$;
04.   for $i = 1, \ldots, MaxIter$ do
05.     $Sol \leftarrow Build\_Random\_Greedy\_Solution\_KLD(n, m, DivMat)$;
06.     $Sol \leftarrow Local\_Search\_KLD(Sol)$;
07.     if $E$ is full then do
08.       $SolElite \leftarrow Select\_Sol\_Elite(E)$;
09.       $Sol \leftarrow Path\_Relinking(Sol, SolElite)$;
10.     end if;
11.     $Update\_Elite(E, Sol)$;
12.     if $z(Sol) > z\_best$ do
13.       $Best\_Sol \leftarrow Sol$;
14.       $z\_best \leftarrow z(Sol)$;
15.     end if;
16.   end for;
17.   return $Best\_Sol$;
```

Fig. 4. Algorithm for GRASP with path-relinking

We show in Fig. 5 the details on our implementation of path-relinking for the MDP. In line 1, the solution generated by the GRASP iteration Sol and the solution from the elite size $SolElite$ are compared to determine which has the better cost. BS will contain the best solution and WS the worst. In lines 3 and 4, the initial and guide solutions are set. The initial solution is set as the best-cost solution between Sol and $SolElite$. From line 5 to 12, the steps of path-relinking are performed until the initial solution reaches the guide solution. In line 6, the elements that are in the guide solution and are not in the initial solution are found. In line 7, a step of the path-relinking is performed. Intermediate solutions are obtained by replacing an element that belongs to the initial solution and do not belong to the guide solution with all other elements that belong to the guide solution and do not belong to the initial solution. The solution which presents the best cost among all these intermediate solutions is selected ($IntSol$) as the result of an iteration of path-relinking. This solution then is more similar to the guide solution because one element from the initial solution was replaced by another one from the guide solution. In line 9, if this new intermediate solution has a better cost than the current best intermediate solution ($BestSolPR$), then the latter is updated by this new one. In line 11, the intermediate solution is set as the initial solution for the next step of the path-relinking.

```
procedure Path_Relinking($Sol$, $SolElite$)
01.   $Find\_Best\_Worst(Sol, SolElite, BS, WS)$;
02.   $BestSolPR \leftarrow BS$;
03.   $Initial \leftarrow BS$;
04.   $Guide \leftarrow WS$;
05.   while $Initial$ not equal to $Guide$ do
06.     $DifEl \leftarrow Obtain\_Different\_Elements(Initial, Guide)$;
07.     $IntSol \leftarrow Obtain\_Int\_Sol(Initial, DifEl)$;
08.     if $z(IntSol) > z(BestSolPR)$ then do
09.       $BestSolPR \leftarrow IntSol$;
10.     end if;
11.     $Initial \leftarrow IntSol$;
12.   end while;
13.   return $BestSolPR$;
```

Fig. 5. Algorithm for path-relinking

The computational experiments implemented to evaluate the performance of this heuristic are presented in next section.

4 Computational Results

The performance of the new developed GRASP with path-relinking was evaluated using all instances from set D. These instances are larger and, according to the results presented in Subsection 2.3, they are more difficult to solve.

The new algorithm was also implemented in C and compiled with `gcc 3.3.2`. The tests were performed on a 1.4 GHz AMD Athlon with 250 Mbytes of RAM.

In Table 11, we show the results of computing 500 iterations for the pure GRASP (KLD) and for KLD with path-relinking (KLD+PR) using instances from set D. The elite set size was 5 and the path-relinking strategy starts to be performed only after this set set is full, i.e., during the first five iterations, path-relinking is not applied. The first and second columns identify the parameters of each instance: the size n of the population and the number m of elements to be selected. For all instances, each procedure was executed three times. The third and fourth columns show the average values obtained by KLD and KLD+PR strategies. For these average values, when two different results are reached, a bold value indicates the best average found. In order to compare the results obtained by KLD+PR to those reached by other proposed strategies (Ghosh, KLD, KLD2, or any other presented in the related literature [13]), the last column shows the best known average value for each test.

The two algorithms obtained the same results for population size 100. Considering the other 16 tests, with population size larger or equal to 200, different average diversity values were reached in 12 tests. KLD+PR heuristic found better results than KLD algorithm in all these 12 cases.

We can also observe that the average diversity solutions reached by KLD+PR are equal to the best known average values in almost all cases: in 16 from 20 tests.

Table 11. Comparing KLD and KLD+PR average results for instances from set D

n	m	KLD	KLD+PR	Best Average
100	10	333.0	333.0	333.00 ([13],KLD,KLD2,KLD+PR)
100	20	1195.0	1195.0	1195.0 ([13],KLD,KLD2,KLD+PR)
100	30	2457.0	2457.0	2457.0 ([13],Ghosh,KLD,KLD2,KLD+PR)
100	40	4142.0	4142.0	4142.0 ([13],Ghosh,KLD,KLD2,KLD+PR)
200	20	1247.0	1247.0	1247.0 ([13],KLD,KLD2,KLD+PR)
200	40	4446.3	**4450.0**	4450.0 (KLD+PR)
200	60	9437.0	9437.0	9437.0 ([13],Ghosh,KLD,KLD2,KLD+PR)
200	80	16182.3	**16225.0**	16225.0 ([13],Ghosh,KLD2,KLD+PR)
300	30	2686.0	**2691.0**	2692.0 ([13])
300	60	9667.7	**9684.0**	9684.0 (KLD+PR)
300	90	20640.0	**20740.3**	20740.3 (KLD+PR)
300	120	35871.0	**35879.0**	35881.0 ([13])
400	40	4635.3	**4655.7**	4655.7 (KLD+PR)
400	80	16916.3	**16935.0**	16946.7 ([13])
400	120	36175.0	**36302.0**	36302.0 (KLD+PR)
400	160	62313.0	**62467.3**	62478.0 (KLD2)
500	50	7124.0	7124.0	7124.0 (KLD,KLD+PR)
500	100	26197.0	**26246.3**	26246.3 (KLD+PR)
500	150	56572.0	56572.0	56572.0 (KLD,KLD2,KLD+PR)
500	200	97213.0	**97333.3**	97333.3 (KLD+PR)

Ghosh heuristic obtained only four best known average values, KLD reached 8 values, and KLD2 9 values. Distinct algorithms developed in [13] found 10 best known average values. For seven instances, KLD+PR achieved a new best average which were not previously reached by any other strategy.

In Table 12, we show the best diversity values achieved by both KLD and KLD+PR strategies in the conducted results. The third and fourth columns show the best values obtained by them. For these best values, when two different results are reached, a bold value indicates the best one. In order to compare the results obtained by KLD+PR to those reached by the other proposed strategies, the last column shows the best known value for each test.

Again, the two algorithms obtained the same results for population size 100. Considering the other 16 tests, different best diversity values were reached in 10 tests. In all of them, KLD+PR heuristic found better results than KLD algorithm.

The KLD+PR heuristic was able to find the best known solution in 15 from 20 tests.

Ghosh heuristic obtained only six best known values, KLD reached 8 values, and KLD2 9 values. The algorithms developed in [13] together found 17 best known average values. However, we observe that, from these strategies, the one with best behavior achieved 12 best results.

Table 12. Comparing KLD and KLD+PR best results for instances from set D

n	m	KLD	KLD+PR	Best Result
100	10	333.0	333.0	333.00 ([13],KLD,KLD2,KLD+PR)
100	20	1195.0	1195.0	1195.0 ([13],KLD,KLD2,KLD+PR)
100	30	2457.0	2457.0	2457.0 ([13],Ghosh,KLD,KLD2,KLD+PR)
100	40	4142.0	4142.0	4142.0 ([13],Ghosh,KLD,KLD2,KLD+PR)
200	20	1247.0	1247.0	1247.0 ([13],KLD,KLD2,KLD+PR)
200	40	4448.0	**4450.0**	4450.0 ([13],KLD+PR)
200	60	9437.0	9437.0	9437.0 ([13],Ghosh,KLD,KLD2,KLD+PR)
200	80	16207.0	**16225.0**	16225.0 ([13],Ghosh,KLD2,KLD+PR)
300	30	2691.0	2691.0	2694.0 ([13])
300	60	9689.0	9689.0	9689.0 (KLD,KLD+PR)
300	90	20640.0	**20743.0**	20743.0 (KLD+PR)
300	120	35871.0	**35879.0**	35881.0 ([13],Ghosh,KLD2)
400	40	4653.0	**4658.0**	4658.0 ([13],KLD+PR)
400	80	16925.0	**16945.0**	16956.0 ([13])
400	120	36175.0	**36306.0**	36315.0 ([13])
400	160	62313.0	**62487.0**	62487.0 (KLD+PR)
500	50	7130.0	7130.0	7131.0 ([13])
500	100	26201.0	**26254.0**	26254.0 ([13],KLD+PR)
500	150	56572.0	56572.0	56572.0 ([13],Ghosh,KLD,KLD2,KLD+PR)
500	200	97213.0	**97344.0**	97344.0 ([13],KLD+PR)

In two cases ($n = 300, m = 90$ and $n = 400, m = 160$), KLD+PR achieved a new best solution which were not previously found by any other strategy.

Table 13 report the CPU times observed for the execution of the same instances. For each GRASP heuristic, the average time for the three executions is reported. Heuristic GRASP with path-relinking, which achieved better quality solutions, demanded more time than the pure GRASP heuristic for all instances.

Next, the same analysis performed in Subsection 2.3 is conducted here to evaluate the empirical probability distributions for the time to achieve a target value.

We choose three instances from set D with parameters: ($n = 200$,$m = 20$), ($n = 200$,$m = 80$), and ($n = 300$,$m = 120$). Each GRASP heuristic was executed until a solution was found with a greater or equal cost compared to a target value. Two target values were used for each instance. Empirical probability distributions for the time to achieve a target value are plotted in Fig(s). 6, 7, and 8.

We can observe that KLD+PR behaves better than KLD. KLD+PR algorithm presented a higher probability of finding a target solution in a given amount of time than KLD in all tests. Fig. 6 (target=1244) shows that the probability of finding the target value in 1 second is approximately 6% for KLD, while for KLD+PR is 75%. Considering 10 seconds, we have the probabilities 50% and 100% respectively. Also in Fig. 8 (target=35873), we observe that the

Table 13. Computational time for KLD and KLD+PR for instances from set D

n	m	KLD	KLD+PR
100	10	10.4	11.0
100	20	28.2	28.4
100	30	57.4	57.7
100	40	89.2	89.7
200	20	71.7	77.4
200	40	318.3	321.3
200	60	626.6	743.6
200	80	804.3	1122.8
300	30	412.7	413.7
300	60	1690.5	1842.7
300	90	3260.6	4183.1
300	120	4453.0	7013.3
400	40	1208.7	1300.6
400	80	5266.5	5957.1
400	120	10097.5	13850.7
400	160	15750.3	20456.7
500	50	2934.9	3072.5
500	100	11921.7	15857.2
500	150	29051.4	34661.4
500	200	39043.3	52497.4

probability of finding the target value in 9 seconds is approximately 60% for KLD and for KLD+PR is 80%.

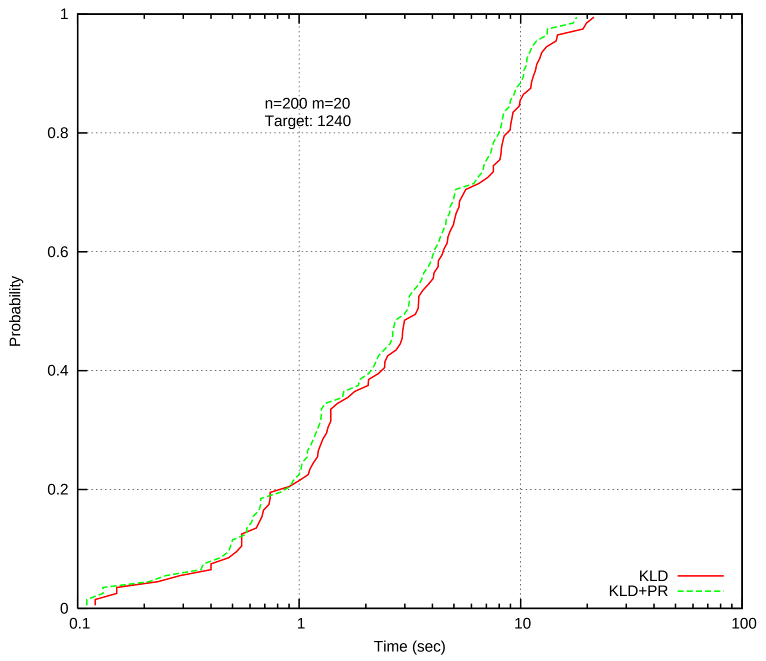
5 Concluding Remarks

This paper presented some versions of GRASP heuristics to solve the maximum diversity problem (MDP). The main goal of this work was to analyse the influence of inserting the path-relinking technique to a pure GRASP.

Experimental results show that the version which uses path-relinking technique (KLD+PR) significantly improves the average and the best quality of solutions generated by an important pure GRASP approach proposed in the literature (KLD). Our experiments also show that path-relinking speeds up convergence to target values.

References

1. Aiex, R. M., Resende, M. G. C., Ribeiro, C. C.: Probability distribution of solution time in GRASP: An experimental investigation, Journal of Heuristics **8** (2002), 343–373.
2. Feo, T. A., Resende, M. G. C.: Greedy randomized adaptive search procedures, Journal of Global Optimization **6** (1995), 109–133.



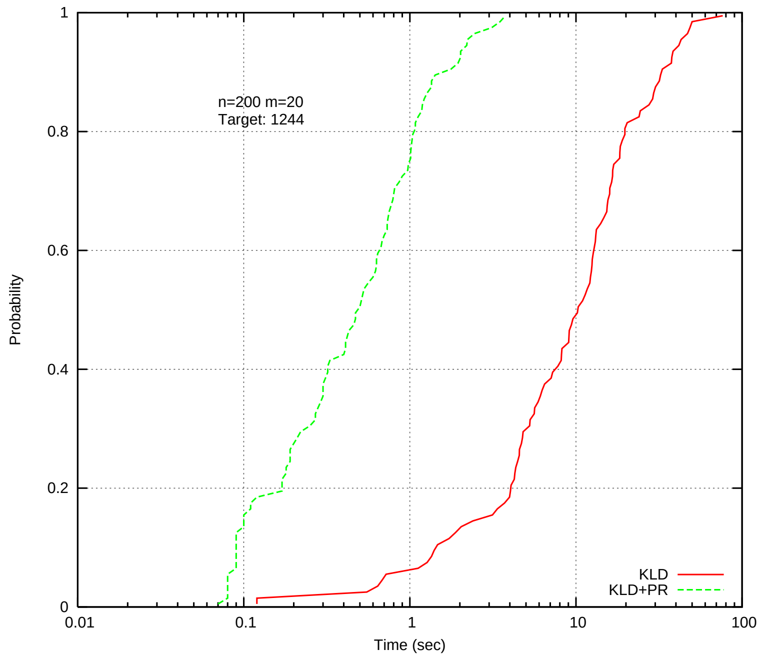


Fig. 6. Comparison of GRASP heuristics for the instance $n = 200, m = 20$ with targets values 1240 and 1244

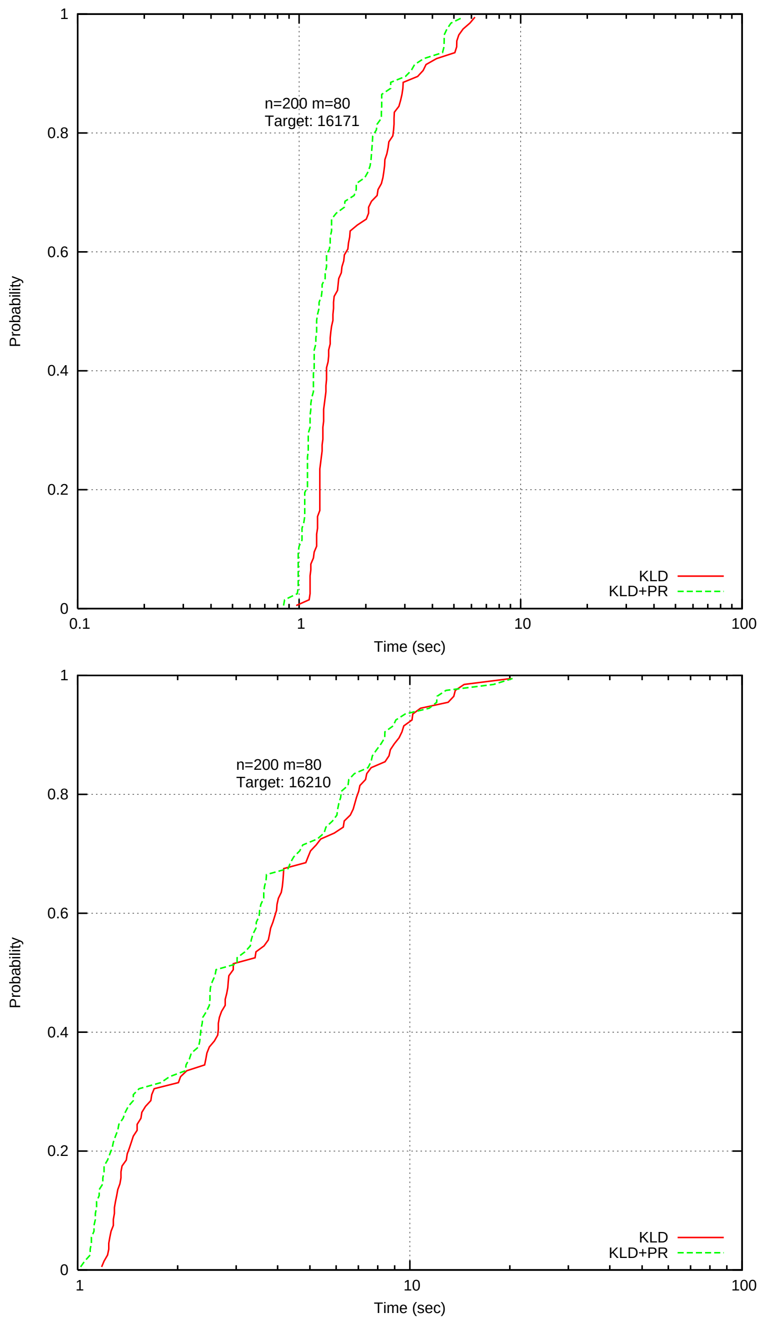


Fig. 7. Comparison of GRASP heuristics for the instance $n = 200, m = 80$ with targets values 16171 and 16210

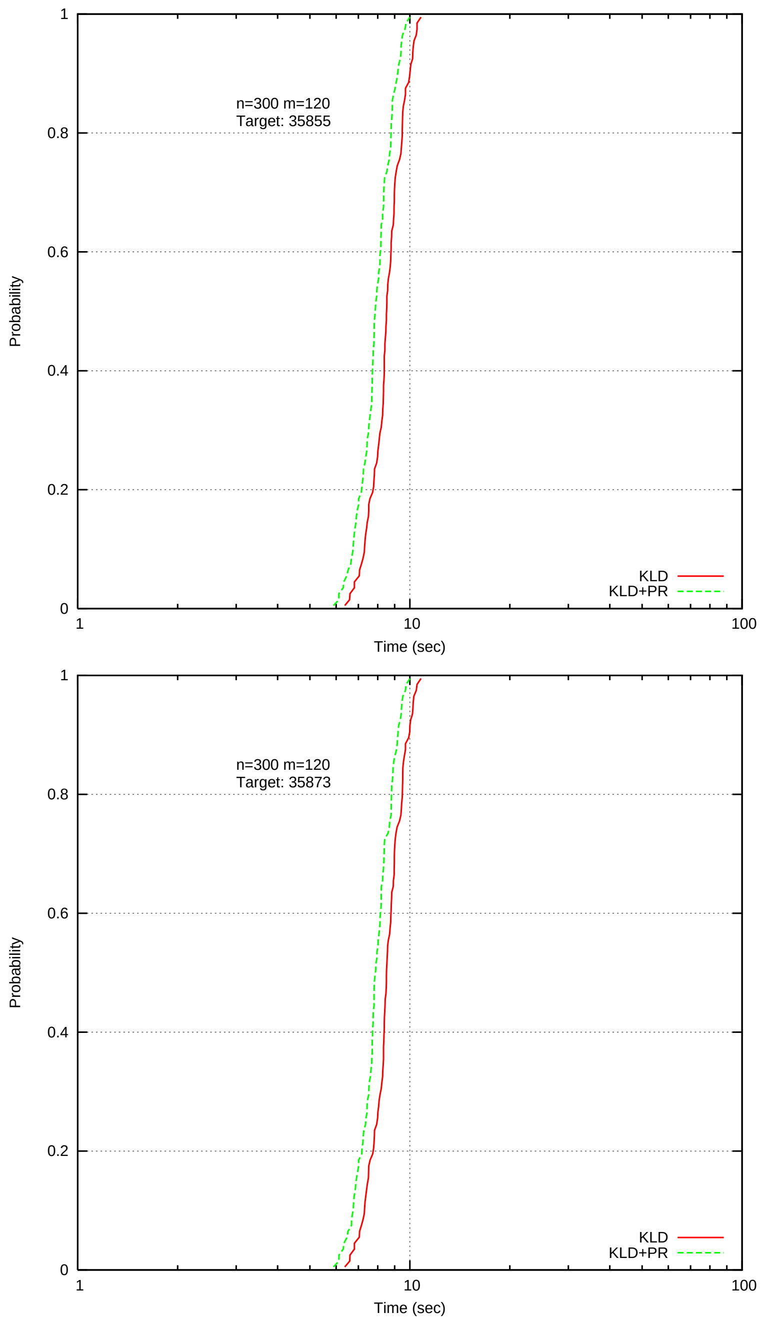


Fig. 8. Comparison of GRASP heuristics for the instance $n = 300, m = 90$ with targets values 35855 and 35873

3. Ghosh, J. B.: Computational aspects of the maximum diversity problem, Operations Research Letters **19** (1996), 175–181.
4. Glover, F., Laguna, M., Marti, R.: Fundamentals of scatter search and path-relinking, Control and Cybernetics **19** (1977), 653–684.
5. Glover, F., Hersh, G., McMillan, C.: Selecting subsets of maximum diversity, MS/IS Report No. 77-9, University of Colorado at Boulder, (1977).
6. Glover, F., Kuo, C-C., Dhir, K. S.: Integer programming and heuristic approaches to the minimum diversity problem, Journal of Business and Management **4** (1996), 93–111.
7. Kochenberger, G., Glover, F.: Diversity data mining, Working Paper, The University of Mississipi, (1999).
8. Laguna, M., Mart, R.: GRASP and path relinking for 2-layer straight line crossing minimization, INFORMS Journal on Computing **11** (1999), 44–52.
9. Prais, M., Ribeiro, C. C.: Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment, INFORMS Journal on Computing **12** (2000), 164–176.
10. Resende, M. G. C., Ribeiro, C. C.: Greedy randomized adaptive search procedures, Handbook of Metaheuristics (F. Glover and G. Kochenberger editors), (2003), 219–249.
11. Resende, M. G. C., Ribeiro, C. C.: GRASP with path-relinking: Recent advances and applications, Metaheuristics: Progress as Real Problem Solvers (T. Ibaraki et al. editors), (2005), 29–63.
12. Ribeiro, C. C., Uchoa, E., Werneck, R. F.: A hybrid GRASP with perturbations for the Steiner problem in graphs, INFORMS Journal on Computing 14 (2002), 228-246.
13. Silva, G. C., Ochi, L. S., Martins, S. L.: Experimental comparison of greedy randomized adaptive search procedures for the maximum diversity problem, Lecture Notes on Computer Science **3059** (2004), 498–512.
14. Weitz, R., Lakshminarayanan, S.: An empirical comparison of heuristic methods for creating maximally diverse groups, Journal of the Operational Research Society **49** (1998), 635–646.