International Journal of Latest Research in Science and Technology
Volume 3, Issue 4: Page No.186-194. July-August 2014
https://www.mnkpublication.com/journal/ijlrst/index.php

ISSN (Online):2278-5299

A MEMORY-BASED GRASP HEURISTIC FOR THE MULTIOBJECTIVE MULTIDIMENSIONAL KNAPSACK PROBLEM

[1]Dalessandro Soares Vianna, [2]Fermín Alfredo Tang Montané, [3]Edwin Benito Mitacc Meza, [1]Carlos Bazilio Martins
[1]Department of Computation, Fluminense Federal University, Rio das Ostras, RJ, Brazil
[2]Department of Computation, State University of Norte Fluminense, Campos do Goytacazes, RJ, Brazil
[3]Department of Engineering, Fluminense Federal University, Rio das Ostras, RJ, Brazil

Abstract*- *In real optimization problems is generally desirable to optimize more than one performance criterion (or objective) at the same time. The goal of the multi-objective combinatorial optimization (MOCO) is to optimize simultaneously r > 1 objectives. We developed a GRASP algorithm that incorporates a memory-based approach for solving the multiobjective multidimensional knapsack problem. There are, in the scientific literature, some memory-based GRASP algorithms for mono-objective problems, however it was not found any memory-based GRASP algorithm for multiobjective problems. Computational experiments on benchmark instances show that the proposed algorithm is very robust and outperforms other heuristics in terms of solution quality and running times.

Keywords - multiobjective multidimensional knapsack problem; multiobjective combinatorial optimization; GRASP.

I. INTRODUCTION

Many practical optimization problems, generally, involve simultaneous minimization (or maximization) of several conflicting decision criteria. The goal of multiobjective combinatorial optimization (MOCO) is to optimize simultaneously $r > 1$ criteria or objectives. MOCO problems have a set of optimal solutions (instead of a single optimum) in the sense that no other solutions are superior to them when all objectives are taken into account. They are known as *Pareto optimal* or *efficient* solutions.

Solving MOCO problems is quite different from single-objective case ($r = 1$), where an optimal solution is searched. The difficulty is not only due to the combinatorial complexity as in single-objective case, but also due to finding all elements of the efficient set, whose cardinality grows with the number of objectives.

In the literature, some authors have proposed exact methods for solving specific MOCO problems [10] [30] [35]. These methods are generally valid for bi-objective ($r = 2$) problems but cannot be adapted easily to a higher number of objectives. Also, exact methods are inefficient to solve large-scale NP-hard MOCO problems. As in the single-objective case, the use of heuristic/metaheuristic techniques seems to be the most promising approach to MOCO problems because of their efficiency, generality and relative simplicity of implementation. These techniques generate good approximated solutions in a short computational time. Several articles have proposed heuristic procedures to solve MOCO problems [5] [7] [9] [10] [22] [24] [32].

The literature on the multiobjective knapsack problem is rather scarce. Some of the works found are as follow: the methods proposed by Ulungu and Teghem [30] and Visée et al. [35] are based on exact algorithms; Jaskiewicz [21], Zitzler and Thiele [36] and Alves e Almeida [1] use genetic algorithms; the methods of Gandibleux and Frévile [16] and Hansen [18] are based on tabu search; and the methods proposed by Czyzak and Jaskiewicz [8] and Ulungu, Teghem and Ost [31] are based on simulated annealing. Vianna and Dianin [34] have proposed algorithms based on GRASP and ILS metaheuristics.

In [33], Vianna and Arroyo proposed a GRASP algorithm, called **GRASP-MULTI**, for solving the multi-objective knapsack Problem. It outperformed two well known genetic algorithms from literature: **MOGLS** (Multi-objective Genetic Local Search) suggested by Jaskiewicz [21] and **SPEA2** [37], which is an improved version of the genetic algorithm **SPEA** (Strength Pareto Evolutionary Algorithm) proposed by Zitzler and Thiele [36]. After that, Alves and Almeida [1] proposed a genetic algorithm, called **MOTGA** (Multiple objective Tchebycheff based Genetic Algorithm), that also outperforms the **MOGLS**, **SPEA** and **SPEA2**. It is based on the Tchebycheff scalarizing function, which performs several stages, each one intended for searching potentially nondominated solutions in a different part of the Pareto frontier.

This paper presents a new algorithm for the multi-objective knapsack problem, called **MMGRASP** (Memory-based Multiobjective GRASP), which is an adaptation of a previous algorithm, **GRASP-MULTI**, described in [33]. Each iteration of **GRASP-MULTI** algorithm solves a mono-objective problem, where the goal is to maximize a linear combination of the objective functions of the original multi-objective problem. The linear combination is obtained by assigning weights to each objective function, which represents a search direction for the multi-objective problem.

Publication History
Manuscript Received : *26 June 2014*
Manuscript Accepted : *1 July 2014*
Revision Received : *10 August 2014*
Manuscript Published : *31 August 2014*

In the **GRASP-MULTI** algorithm, as in any traditional GRASP algorithm, each iteration is independent, i.e., when one iteration finishes, the information found during this iteration is not used in the next iterations. In the **MMGRASP**, the iteration k+1 uses the best solution found during the iteration k as a starting point.

There are, in the scientific literature, some memory-based GRASP algorithms for mono-objective problems [2] [6] [11] [14] [15], however it was not found any memory-based GRASP algorithm for multiobjective problems. So, we believe that it is an innovation of this work.

The organization of the paper is as follows. In the next section, we present the formulation of a MOCO problem and a formal definition of the multiobjective knapsack problem. In section III, we discuss with more details the proposed memory-based multiobjective GRASP algorithm. We present computational results in Section IV, where we use, among others, the time-to-target experiment proposed in [13] and commonly used in mono-objective problems [3] [28] [29]; but its use in multiobjective problems is another innovation of this work. Finally, Section V contains our concluding remarks.

II. MULTIOBJECTIVE OPTIMIZATION

A. *Problem statement and basic definitions*

Given a vector function of r components $f = (f_1, \ldots, f_r)$ defined on a finite set Ω, consider the multi-objective combinatorial optimization problem: Maximize $f(x) = (f_1(x) = z_1, \ldots, f_r(x) = z_r)$, subject to $x \in \Omega$.

A. solution x dominates x' if $f(x)$ dominates $f(x')$, that is, if $f_j(x) \geq f_j(x')$, for all objective j, and $f_j(x) > f_j(x')$ for at least one objective j. A solution $x^* \in \Omega$ is *Pareto optimal* (or *efficient*) if there is no $x \in \Omega$ such that x dominates x^*. A solution $x^* \in S \subseteq \Omega$ is *nondominated* in S if there is no $x \in S$ such that x dominates x^*.

B. *Multiobjective knapsack problem (MOKP)*

In the literature are studied different versions of the 0/1 multi-objective knapsack problem [16] [36]. In this paper we use the same formulation considered by Zitzler and Thiele [36], Jaskiewicz [21] and Alves and Almeida [1] in their experiments, which consider the multi-objective problem by allowing r knapsacks with different capacities (multidimensional knapsack). This problem can be formulated as follows:

$$\text{Maximize } f_j(x) = \sum_{i=1}^{n} c_{ij} x_i, \quad j = 1, \ldots, r$$

$$\text{Subject to } \sum_{i=1}^{n} w_{ij} x_i \leq W_j, \quad j = 1, \ldots, r$$

$$x_i \in \{0,1\}, \quad i = 1, \ldots, n$$

where c_{ij} and w_{ij} are, respectively, the profit and weight of item i according to knapsack j, W_j is the capacity of knapsack j and $x = (x_1, \ldots, x_n)$ is a vector of binary variables such that $x_i = 1$ if the item i belongs to the knapsacks and $x_i = 0$, otherwise.

The objectives are conflicting because the benefit of putting an item i into a knapsack j (c_{ij}) can be high, while placing the same item i in another knapsack l (c_{il}) may not be attractive (low benefit).

III. MEMORY-BASED MULTIOBJECTIVE GRASP ALGORITHM – MMGRASP

GRASP – Greedy Randomized Adaptive Search Procedure [12] [27] – is a multi-start metaheuristic, in which each iteration consists of two phases: construction and local search. The construction phase builds a feasible solution using a greedy randomized algorithm, while the local search phase calculates a local optimum in the neighbourhood of the feasible solution. Both phases are repeated a pre-specified number of iterations and the best overall solution is kept as the result.

In Subsections III.A and III.B are presented, respectively, the constructive and local search phases of the **MMGRASP** algorithm, which are also used in the **MULTI-GRASP** algorithm [33]. Details about the memory-based approach is given in Subsection III.C.

A. *Greedy randomized construction*

To generate an initial set of dominating solutions, a greedy heuristic is used to maximize a linear combination of the objective functions:

$$f(x) = \sum_{j=1}^{r} \lambda_j f_j(x)$$

$$\text{where } \sum_{j=1}^{r} \lambda_j = 1$$

$$0 \leq \lambda_j \leq 1$$

The preference vector $\lambda = (\lambda_1, \ldots, \lambda_r)$, generally, determinates a search direction on the Pareto optimal frontier. Figure 1 presents the implemented constructive algorithm, **BuildSolution**, which is a greedy randomized algorithm that builds a solution by inserting items with the higher value for the following ratio:

$$\frac{\sum_{j=1}^{r} \lambda_j c_{ej}}{\sum_{j=1}^{r} w_{ej}} \tag{1}$$

This ratio measures the benefit of including an item e in the knapsacks. As bigger the ratio, better is the benefit of the item. The **BuildSolution** algorithm receives as input parameters the solution x to be built, the percentage α used in the selection of the next element to be inserted in x, the search direction λ and the *lPareto* list, where the dominants solutions are stored. As output, the algorithm returns the built solution x. In line 1, the candidates list CL is defined. In this

list are inserted all the items out of the knapsacks. The *CL* list is sorted in decreasing order according to the ratio 1. As showed in line 3, the restricted candidates list *RCL* is composed of the $\alpha \times |CL|$ first items of *CL* list. The loop in lines 4-8 is responsible by the randomization of the algorithm. An item *e* is randomly selected from *RCL* and inserted in *x*. This process is repeated while the insertion of *e* does not violate the capacity of the knapsacks. The loop in lines 9-14 looks for additional insertions from *CL*. This stage is greedy, respecting the sorting of *CL* list, and try to improve, if possible, the solution found in the previous stage (loop in lines 4-8). Experiments have shown that only very few items are inserted during this stage. Thus, an improvement in the current solution can be achieved without compromising the greedy-randomized feature of the algorithm. In line 15 is verified if the solution *x* is a dominant solution and, finally, the solution *x* is returned in line 16.

Procedure BuildSolution (*x*, α, λ, *lPareto*)
Input
x – solution to be built;
α – percentage used on the definition of the restricted candidates list (*RCL*);
λ – vector of preferences (search direction);
lPareto – list of dominant solutions that will be updated with *x*.
Output
x – built solution.
Begin
01. Insert each item *e* ($x_e = 0$) in the candidates list *CL* sorted (decreasing) by Ratio 1.
02. Let *RCL* be a list with the α% first items of *CL*;
03. Select randomly an item *e* from *RCL*;
04. **While** $x \cup x_e$ does not violate W_j, for j=1, ..., r **do**
05. $x \leftarrow x \cup x_e$; //insert *e* in the knapsacks
06. Remove the item *e* of *CL*;
07. Select randomly an item *e* from *RCL*;
08. **End_while**
09. **For** $i \leftarrow 1$ **to** $|CL|$ **do**
10. $e \leftarrow$ the i^{th} item of *CL*;
11. **If** $x \cup x_e$ does not violate W_j, for j=1, ...,r **then**
12. $x \leftarrow x \cup x_e$; //insert *e* in the knapsacks
13. **End_if**
14. **End_for**
15. Verify the insertion of *x* in *lPareto*;
16. **Return** *x*;
End-BuildSolution

Fig. 1. Construction algorithm.

B. *Local search*

Figure 2 presents the **LocalSearch** algorithm that removes the worst items from the knapsacks according to the Ratio 1 and uses the **BuildSolution** algorithm to produce a new solution. This algorithm receives as input parameters the solution *x* to be refined, the percentage β that is used at the solution reconstruction stage, the search direction λ and the *lPareto* list, where the nondominated solutions are stored.

The loop in lines 1-2 initializes all the positions of the vector *Marked* with **false**. An item *e* can be removed from the knapsack only if *Marked*[*e*] = **false**. The loop in lines 3-15 is executed while exist elements that can be removed, that is, elements still unmarked. In line 4, the solution *x* is attributed to the auxiliary solution *y*. In line 5 are removed from *y* the elements that present the shortest values of the Ratio 1. This process is repeated while there exists an element that is out of the knapsack that can not be inserted without violates any restriction of the problem. The items are removed from the knapsacks until the free space obtained in this way allows the insertion of any item that remains out of the knapsacks. This step is completely greedy. In line 6 the procedure **BuildSolution** is executed completing the construction of the solution *y*.

Procedure LocalSearch (*x*, β, λ, *lPareto*)
Input
x – solution to be refined;
β – percentage used at the reconstruction of solution *x*;
λ – vector of preferences (search direction);
lPareto – list of dominant solutions that will be updated with the found solutions.
Output
x – refined solution.
Begin
01. **For** $i \leftarrow 1$ **to** *n* **do**
02. *Marked*[*i*] $\leftarrow$ **false**;
03. **While** there exists an item *e* such that *Marked*[*e*] = **false** **do**
04. $y \leftarrow x$;
05. Remove the unmarked item *j* ($y_j = 1$) that presents the shortest value of the Ratio 1. Repeat this process until any item *g* ($y_g = 0$) may be chosen for insertion;
06. $y \leftarrow$ **BuildSolution** (*y*, β, λ, *lPareto*);
07. **If** $f(y) > f(x)$ **then**
08. $x \leftarrow y$;
09. **For** $i \leftarrow 1$ **to** *n* **do**
10. *Marked*[*i*] $\leftarrow$ **false**;
11. **Else**
12. Let *e* be the unmarked item of *x* that presents the smallest value of the Ratio 1;
13. *Marked*[*e*] $\leftarrow$ **true**;
14. **End_if**
15. **End_while**
16. **Return** *x*;
End-LocalSearch

Fig. 2. Local search algorithm.

If the new solution, *y*, is better than *x*, then the solution *x* is updated at line 8 and the vector *Marked* is reinitialized in lines 9-10. Otherwise, in line 13, is marked the first element that is removed from *y* during the stage described in line 5. In line 16, the refined solution, *x*, is returned.

The number of iterations of the local search algorithm depends on the quality of the solution *x* received as a parameter.

C. *Memory-based approach*

One strategy used on a Multiobjective GRASP is to explore, at each iteration, a different search direction. The search direction *i* is characterized by the weights associated to each objective, that is, by the preference vector $\Lambda_i = (\lambda_1, ..., \lambda_r)$.

In the proposed strategy, we use the vector $\Lambda = (\Lambda_1, \Lambda_2, ..., \Lambda_m)$ to store the m search directions to be evaluated. In the **GRASP-MULTI** algorithm [33], all the m search directions are analysed by both GRASP phases, construction and local search. In the **MMGRASP** algorithm, only a percentage of these search directions are analysed by both GRASP phases. These directions, called *base search directions*, are chosen uniformly on the vector Λ. The search direction Λ_1 is always a base search direction. To determine the next base search direction, we add the value of the expression $\lfloor m/b \rfloor$, where b represents the number of base search directions desired. This process is repeated until the b base search directions be defined. In Figure 3, we have $m = 9$ search directions and $b = 3$ base search directions ($\lfloor m/b \rfloor = 3$) – Λ_1, Λ_4 and Λ_7 – represented by the solid lines, in which a solution x is built by **BuildSolution** algorithm and refined by **LocalSearch** algorithm. The others directions are evaluated as follow:

- Let Λ_j be a base search direction (in Figures 3 and 4, search directions: Λ_1, Λ_4 and Λ_7) and x^j the solution obtained by the algorithm after evaluating this direction. The memory-based approach uses, during the evaluation of direction Λ_{j+1}, the solution x^j instead of the GRASP construction phase. For example, for evaluating the search direction Λ_5 in Figure 4 it is used the solution x^4, obtained during the evaluation of Λ_4, as initial solution; so, this solution is refined using the **LocalSearch** algorithm. Being the vector Λ well organized, that is, each search direction differing just a little from the previous one (at the end of this subsection we describe with details how to organize Λ), the solution x^j, when evaluated with the weights of direction Λ_{j+1}, looses just a little of its quality, constituting in this way a good initial solution – in the experiments done, a solution generally better than the one found by the GRASP constructive algorithm (traditional construction) – for the local search phase. In Figure 5, the 150 first iterations of a GRASP algorithm with 1000 and 5000 search directions are presented. In both executions, the instance "kn500_3", which will be presented in Section V, is used. Note that when the memory-based approach is used, better initial solutions are produced.
- In the evaluation of the direction Λ_{j+2}, the solution x^{j+1} is used as the initial solution for the local search phase. This process is repeated for all the search directions in Λ, which was not analysed yet (see Figure 4).

In the following subsections, we describe two strategies for organizing the vector Λ. The first one was proposed by Murata, Ishibuchi and Gen [25] and was used in the **GRASP-MULTI** algorithm [33]. The last one represents a modification, proposed in this paper, of the Murata, Ishibuchi and Gen strategy.

1. *Vector Λ organization strategy proposed by Murata, Ishibuchi and Gen* [25]

The strategy proposed by Murata, Ishibuchi and Gen [25], which is used by Vianna and Arroyo [33] in the **GRASP-MULTI** algorithm, generates each component of the vector Λ obtaining r non-negatives integers with sum equal to s,

$$v_1 + v_2 + \ ... + v_r = s, \text{ where } v_i \in \{0, ..., s\}$$

which s is a value large enough to produce m search directions. The number of generated search directions for r objectives and a value s, $N_r(s)$, is calculated as follows:

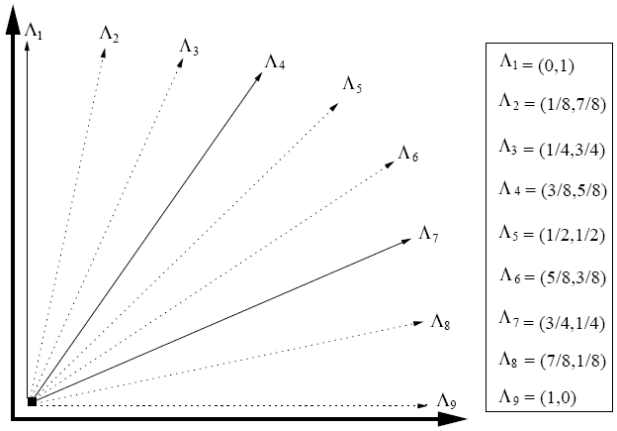

Fig. 3. Base search directions (solid lines).

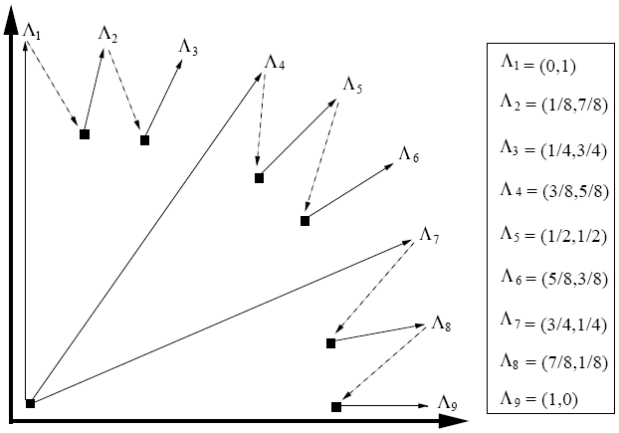

Fig. 4. Memory-based approach.

$$N_2 = s + 1.$$

$$N_3 = \sum_{i=0}^{s} N_2(i) = \sum_{i=0}^{s} (i+1) = (s+1)(s+2)/2$$

$$N_4 = \sum_{i=0}^{s} N_3(i) = \sum_{i=0}^{s} (i+1)(i+2)/2.$$

For instance, for $r = 2$ objectives and $s = 5$ we have 6 vectors (v_1, v_2): (0,5), (1,4), (2,3), (3,2), (4,1) and (5,0). For $r = 3$ and $s = 3$ we have 10 vectors (v_1, v_2, v_3): (0,0,3), (0,1,2), (0,2,1), (0,3,0), (1,0,2), (1,1,1), (1,2,0), (2,0,1), (2,1,0) and (3,0,0).

With the goal of obtaining normalized directions ($\sum_{j=1}^{r} \lambda_j = 1$), we calculate $\lambda_j = v_j/s$, $v_j \in \{0, 1, 2, ..., s\}$.

The Murata, Ishibuchi and Gen algorithm used to generate the vector Λ for a problem with r=3 objectives is described in Figure 6. This algorithm receives as input parameter the integer value s. As output, the algorithm

returns the vector Λ of search directions. This algorithm calculates all the vectors $v = (v_1, v_2, v_3)$ such that the sum of its components is equal to s. After the normalization of this vector, we have the search directions.

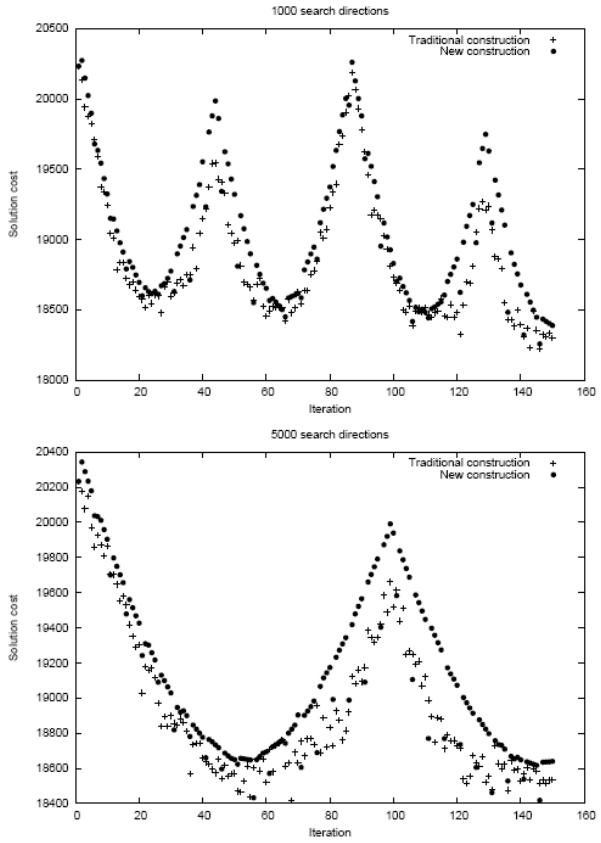

Fig. 5. Initial solutions found.

Procedure Murata_Organization (s)
Input
s – integer value used to generate the search directions.
Output
Λ – vector of search directions.
Begin
01. $i \leftarrow 1$;
02. **For** each value of v_1 ($0 \le v_1 \le s$) **do**
03. **For** each value of v_2 ($0 \le v_2 \le s$) **do**
04. **For** each value of v_3 ($0 \le v_3 \le s$) **do**
05. **If** $v_1 + v_2 + v_3 = s$ **then**
06. $\Lambda_i[1] \leftarrow v_1/s$;
07. $\Lambda_i[2] \leftarrow v_2/s$;
08. $\Lambda_i[3] \leftarrow v_3/s$;
09. $i \leftarrow i + 1$;
10. **End-if**
11. **End-for**
12. **End-for**
13. **End-for**
14. **Return** Λ;
End-Murata_Organization

Fig. 6. **Murata_Organization** algorithm.

2. Proposed Vector Λ organization – UNIFORM-DIST.

In the **Murata_Organization** algorithm described in Figure 6, the value of v_1, v_2 and v_3 is commonly analyzed in an increasing order. This strategy causes a problem when used with the new memory-based approach proposed. When the value of v_1 is incremented, v_2 and v_3 receive the value 0, which causes two consecutive search directions not near one to another. For example, if r=3 objectives and s=5, we have 21 vectors $v = (v_1, v_2, v_3)$ in the following sequence: (0,0,5), (0,1,4), (0,2,3), (0,3,2) (0,4,1), (0,5,0), (1,0,4), (1,1,3), (1,2,2), (1,3,1), (1,4,0), (2,0,3), (2,1,2), (2,2,1), (2,3,0), (3,0,2), (3,1,1), (3,2,0), (4,0,1), (4,1,0) and (5,0,0). The first six vectors have v_1=0 and each one are near to the previous vector (each component varies at most one unity). The 7th vector is the first with v_1=1 and are not near to the previous one (the first component varies one unity; the second one varies five unities; and the third one varies four unities). The same happens with the 12th, 16th and 19th vectors.

The new organization strategy, called **UNIFORM-DIST**, analyzes the **for** statement of line 3 (Figure 6) in an increasing order when the value of v_1 is even, and in a decreasing order when the value of v_1 is odd. The same happens with the **for** statement of line 4 (Figure 6) according to the value of v_2. With this strategy we maintain all the vectors near to the previous one (each component varies at most one unity). For instance, for the example of the previous paragraph we have the same 21 vectors $v = (v_1, v_2, v_3)$ organized as following: (0,0,5), (0,1,4), (0,2,3), (0,3,2) (0,4,1), (0,5,0), (1,4,0), (1,3,1), (1,2,2), (1,1,3), (1,0,4), (2,0,3), (2,1,2), (2,2,1), (2,3,0), (3,2,0), (3,1,1), (3,0,2), (4,0,1), (4,1,0) and (5,0,0).

D. ***MMGRASP*** *algorithm*

Figure 7 presents the **MMGRASP** algorithm, which is the **GRASP-MULTI** algorithm incorporated with the new memory-based approach proposed in this paper. This algorithm receives as input parameters the number of iterations (*N_iter*), the percentage α used at the construction phase, the percentage β used at the local search phase and the number b of base search directions. Parameters α and β were empirically set at 10% and 50%, respectively. Parameter b was empirically set as 30% of *N_iter*. As output, the algorithm returns the *lPareto* list, where the nondominated solutions are stored. In line 1, the *lPareto* list is initialized. In line 2, the vector Λ with the m = *N_iter* search directions is organized by **UNIFORM-DIST** strategy (Subsection III.C.2). The b base search directions are defined in line 3. The loop in lines 4-14 executes *N_iter* GRASP iterations. In line 5, the solution x is initialized. The search direction Λ_i is defined in line 6. If Λ_i is a base search direction (Λ_1 is always a base search direction), the solution x is built by the **BuildSolution** procedure in line 8. Otherwise, the solution y obtained at the previous GRASP iteration is used as initial solution. The solution x is refined in line 13. Finally, the *lPareto* list is returned.

IV. COMPUTATIONAL EXPERIMENTS

We compare the results of **MMGRASP** with the following algorithms: **GRASP_MULTI** [33] and **MOTGA** [1]. Both algorithms have outperformed three well known algorithms: **SPEA** [36], **SPEA2** [37] and **MOGLS** [21].

Procedure MMGRASP (*N_iter*, α, β, *b*)
Input
N_iter – number of GRASP iterations;
α – percentage used at the construction stage;
β – percentage used at the local search stage;
b – number of base search directions.
Output
lPareto – list of nondominated solutions.
Begin
01. *lPareto* $\leftarrow \varnothing$;
02. Organize the vector Λ of search directions by **UNIFORM-DIST** strategy;
03. Define the *b* base search directions;
04. **For** *i* $\leftarrow$ 1 **to** *N_iter* **do**
05. $x \leftarrow \varnothing$;
06. Let Λ_i be the search direction in the position *i* of Λ;
07. **If** Λ_i is a base search direction **then**
08. $x \leftarrow$ **BuildSolution** (*x*, α, Λ_i, *lPareto*);
09. **Else**
10. $x \leftarrow y$; *//y is the solution obtained in the iteration i-1;*
11. Evaluate *x* with the weights associated to the new search direction Λ_i;
12. **End_if**
13. $y \leftarrow$ **LocalSearch** (*x*, β, Λ_i, *lPareto*);
14. **End_for**
15. **Return** *lPareto*;
End-MMGRASP

Fig. 7. **MMGRASP** algorithm.

All computational experiments with the **MMGRASP** and **GRASP-MULTI** algorithms were performed on a 3.2GHz Pentium IV processor with 1 Gbyte of RAM memory. The **MMGRASP** algorithm was implemented in C using version 6.0 of the Microsoft Visual C++ compiler.

A. *Test instances*

In this work, we use the set of instances proposed by Zitzler and Thiele [36]. They generated instances with 250, 500 and 750 items, and 2, 3, and 4 objectives. Uncorrelated profits and weights were randomly generated in the interval [10, 100]. The knapsack capacities were set to half of the total weight regarding the corresponding knapsack: $W_j = 0.5\sum_{i=1}^{n} w_{ij}$. The problem instances are presented in Table 1 and are available at:
http://www.tik.ee.ethz.ch/$\sim$zitzler/testdata.html.

B. *Evaluation of computational results in multiobjective optimization*

The quality of a solution of a single-objective minimization problem is evaluated in a straightforward manner as the relative difference between the objective value of such solution and the value of an optimal solution. In multiobjective optimization, however, there is no natural single measure that is able to capture the quality of a nondominated set *H* to the Pareto optimal set or reference set *R*.

Table 1. Test instances.

Instance	*Objectives*	*Items*
kn250_2	2	250
kn250_3	3	250
kn250_4	4	250
kn500_2	2	500
kn500_3	3	500
kn500_4	4	500
kn750_2	2	750
kn750_3	3	750
kn750_4	4	750

We measure the quality of the nondominated set *H* generated by the heuristic method relative to the reference set *R* by using two measures:

- *Cardinal measure*: number of reference solutions, *NRS*, found by the heuristic method, where $NRS = |H \cap R|$.
- *Distance measure* (proposed by [8] and [31]): distance between the nondominated set *H* generated by the heuristic method and the reference set *R*. We measure the average distance, D_{avg}, and maximum distance, D_{max}, with
  - $D_{avg} = \frac{1}{|R|}\sum_{z\in R} \min_{z'\in H} d(z', z)$ and
  - $D_{max} = \max_{z\in R}\{\min_{z'\in H} d(z', z)\}$

where $d(z', z) = \max_{j=1,\ldots,r}\left\{\frac{1}{\Delta_j}(z'_j - z_j)\right\}$, $z' \in H$, $z \in R$ and Δ_j is the range of the objective f_j among all reference and heuristic solutions.
Note that D_{avg} is the average distance from a point $z \in R$ to its closest point in *H*, while D_{max} yields the maximum distance from a point $z \in R$ to any point in *H*.

When the Pareto optimal set is not known and *H*' is the set of nondominated points generated by another heuristic method, we define the reference set *R* as the nondominated points of ($H \cup H'$) and use the same measures mentioned above to assess the approximation of *H* and *H*' relative to *R*.

C. *Test instances*

The experiments done were conducted using the test instances described in Table 1, which were proposed by Zitzler et al. [36] and has been also used by **GRASP-MULTI** [33] and **MOTGA** [1] algorithms.

In the first experiment, the **MMGRASP** algorithm, proposed at this paper, was run five times to each test instances. Each run finished when the average running time spent by **MOTGA** algorithm was achieved. The goal of this experiment is to evaluate **MMGRASP** and **MOTGA** algorithms running the same time in a similar machine. Table 2 shows the average running times of **MOTGA**.

Table 2. Average running times of MOTGA algorithm on a Pentium IV 3.2 GHz.

Instance	*Time*(*s*)
kn250_2	1.5
kn250_3	7.2
kn250_4	19.5
kn500_2	2.7
kn500_3	12.8
kn500_4	33.4
kn750_2	4.2
kn750_3	18.2
kn750_4	51.9

Table 3 presents comparative results for the first experiment. In the second column is presented the number $|R|$ of reference solutions. In the following columns are presented, for each algorithm (**MOTGA** and **MMGRASP**) and for each instance, the total number of obtained solutions (*TNS*), the number of reference solutions (*NRS*), the average distance (D_{avg}) and the maximum distance (D_{max}).\

The results show that when the number of reference solutions (*NRS*) is compared, the proposed algorithm, **MMGRASP**, generates a larger number of reference solutions on 8 instances from a total of 9 instances. So, by the cardinal measure, **MMGRASP** performs better than **MOTGA**. When the average distance, D_{avg}, and the maximum distance, D_{max}, are compared, **MMGRASP** also performs better than **MOTGA**.

In the second experiment, the **MMGRASP** algorithm was compared with the **GRASP-MULTI** algorithm. The goal of this experiment is to show the efficiency of the memory-based approach proposed in this paper. 100 GRASP iterations are executed by both algorithms. The algorithms are compared using the cardinal and distance measures presented at Subsection IV.B.

Table 4 presents comparative results for the second experiment. In the second column is presented the number $|R|$ of reference solutions. In the following columns are presented, for each algorithm – **GRASP-MULTI** (in the table called just as **GRASP**) and **MMGRASP** –, and for each instance, the total number of obtained solutions (*TNS*), the number of reference solutions (*NRS*), the average distance (D_{avg}), the maximum distance (D_{max}) and the consumed execution time in seconds

The results show that when the number of reference solutions (*NRS*) is compared, the proposed algorithm, **MMGRASP**, generates a larger number of reference solutions for all the test instances. So, by the cardinal measure, **MMGRASP** performs better than **GRASP-MULTI**. When the average distance, D_{avg}, and the maximum distance, D_{max}, are compared, **MMGRASP** also performs better than **GRASP-MULTI**. We also can see that **MMGRASP** is faster than **GRASP-MULTI**

In another experiment comparing **MMGRASP** and **GRASP-MULTI**, we use the time-to-target method [3] [13]. Time-to-target plots (tttplots) display on the ordinate axis the probability that an algorithm will find a solution at least as good as a given target value within a given running time, shown on the abscissa axis. They were used by Feo et al. [13] and have been advocated by Hoos and Stützle [19] [20] as a way to characterize the running times of stochastic algorithms for combinatorial optimization. Aiex et al. [3] preconized and largely explored the use of tttplots to evaluate and compare different randomized algorithms running on the same instance. Their use has been growing ever since and they have been extensively applied in computational studies of sequential and parallel randomized algorithms [27] [28] [29]. The foundations of the construction of time-to-target plots, together with their interpretation and applications, were surveyed by Aiex et al. [4].

Table 3: Comparison of MOTGA and MMGRASP algorithms running the same time in a similar machine.

Instance	$\|R\|$	TNS		NRS		D_{avg}		D_{max}	
		MOTGA	MMGRASP	MOTGA	MMGRASP	MOTGA	MMGRASP	MOTGA	MMGRASP
kn250_2	174.0	100.2	**195.4**	62.0	**112.6**	0.0030	**0.0011**	0.0230	**0.0085**
kn500_2	305.6	200.4	**368.6**	138.2	**175.2**	0.0018	**0.0011**	0.0149	**0.0070**
kn750_2	417.6	246.2	**611.8**	**215.8**	202.0	**0.0012**	0.0014	0.0134	**0.0064**
kn250_3	1999.2	535.0	**1753.0**	504.2	**1495.0**	0.0140	**0.0023**	0.0687	**0.0588**
kn500_3	3960.6	1024.0	**3594.4**	987.6	**2973.0**	0.0131	**0.0027**	0.0729	**0.0588**
kn750_3	5658.8	1448.8	**4924.8**	1387.0	**4271.8**	0.0129	**0.0032**	0.6586	**0.0701**
kn250_4	5294.4	1079.4	**4516.6**	1060.0	**4234.4**	0.0267	**0.0047**	0.1112	**0.0891**
kn500_4	10094.2	2096.2	**8289.2**	2073.8	**8020.4**	0.0278	**0.0051**	0.1109	**0.0946**
kn750_4	14304.4	3056.8	**11642.4**	3010.8	**11293.6**	0.0275	**0.0064**	0.1145	**0.1104**

Table 4: Comparison of GRASP-MULTI and MMGRASP algorithms.

Instance	$\|R\|$	TNS		NRS		D_{avg}		D_{max}		Time (s)	
		GRASP	MMGRASP	GRASP	MMGRASP	GRASP	MMGRASP	GRASP	MMGRASP	GRASP	MMGRASP
kn250_2	165.2	143.2	**149.4**	68.0	**97.2**	0.0024	**0.0014**	0.0126	**0.0095**	0.5	0.5
kn500_2	280.8	236.8	**255.2**	132.2	**149.0**	0.0016	**0.0012**	0.0120	**0.0093**	3.1	**2.9**
kn750_2	453.0	373.4	**393.4**	195.0	**258.2**	0.0017	**0.0009**	0.0125	**0.0075**	9.0	**8.3**
kn250_3	2092.4	1177.4	**1290.2**	853.8	**1239.0**	0.0065	**0.0038**	0.0619	**0.0410**	0.7	0.7
kn500_3	4128.6	2270.8	**2328.8**	1886.4	**2243.2**	0.0062	**0.0059**	0.0698	**0.0521**	4.2	**3.7**
kn750_3	5178.6	**2936.2**	2747.4	2526.6	**2652.0**	0.0062	**0.0059**	0.0661	**0.0509**	10.5	**9.8**
kn250_4	4601.2	2462.2	**2509.6**	2139.6	**2461.8**	0.0108	**0.0100**	0.1064	**0.0834**	1.5	**1.2**
kn500_4	7982.8	3382.2	**4512.4**	3815.2	**4171.2**	0.0144	**0.0111**	**0.1155**	0.1182	5.1	**4.5**
kn750_4	10816.4	4912.2	**6287.0**	4856.4	**5964.0**	0.0167	**0.0096**	0.1212	**0.1023**	12.6	**11.3**

Since "kn750_4" is the greatest and most difficult instance, the results observed for the others instances are summarized by the results obtained for this instance. The time-to-target experiment was conducted using two different stopping rules. In both, we performed 200 independent runs of each algorithm (**MMGRASP** and **GRASP-MULTI**).In the first rule, each run finishes when a number of reference solutions greater than or equal to the number of reference solutions obtained by **MOTGA** algorithm is found. In the second rule, each run finishes when an average distance from the reference set is less than or equal to the average distance obtained by **MOTGA** algorithm is found.

The empirical probability distributions of the time-to-target random variables of the first and second experiment are plotted, respectively, in Figures 8 and 9 for each algorithm. Such plots show that **MMGRASP** algorithm systematically finds better solutions than **GRASP-MULTI** in smaller computation times.

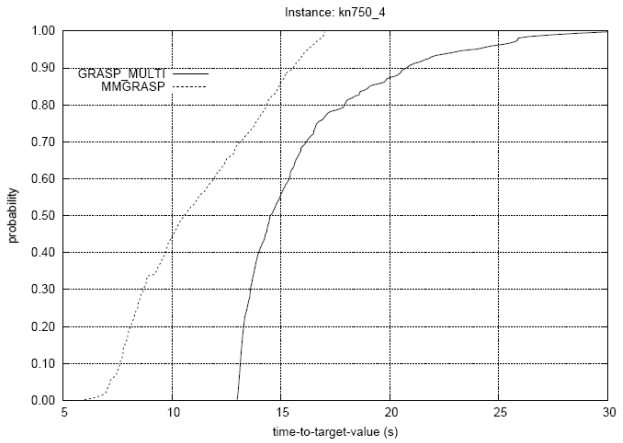

Fig. 8. tttplot experiment with stop criterion: number of reference solutions.

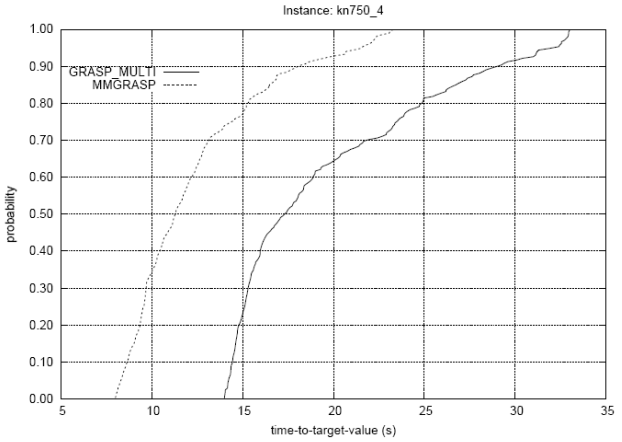

Fig. 9. tttplot experiment with stop criterion: average distance from the reference set.

V. CONCLUSION

In this paper, we propose a memory-based GRASP algorithm to generate a good approximation of the set of efficient or Pareto optimal solutions of a multiobjective combinatorial optimization problem. It is applied to solve the knapsack problem with *r* objectives and it is compared with **GRASP-MULTI** algorithm, proposed by Vianna and Arroyo [33], and with **MOTGA**, proposed by Alves and Almeida [1]. Both algorithms have outperformed three well known algorithms: **SPEA** [36], **SPEA2** [37] and **MOGLS** [21]..

In the experiments done, when the number of reference solution (*NRS*) is compared, the proposed algorithm, **MMGRASP**, generates a larger number of reference solutions on 8 instances from a total of 9 instances, comparing with **MOTGA**, and for all the nine instances, comparing with **GRASP-MULTI**. When the average distance (D_{avg}) is compared, **MMGRASP** obtained a smaller average distance than **MOTGA** on 8 instances from a total of 9 instances and a smaller average distance than **GRASP-MULTI** for all the nine instances.

In experiments done comparing **GRASP-MULTI** and **MMGRASP** algorithms, it was verified that the **MMGRASP** is faster than **GRASP-MULTI**.

Based on the obtained results, it is concluded that the proposed algorithm, **MMGRASP**, is robust, outperforming two efficient algorithms: **MOTGA** and **GRASP-MULTI**.

REFERENCES

[1] Alves M.J., Almeida M. 2007. MOTGA: A multiobjective Tchebycheff based genetic algorithm for the multidimensional knapsack problem. Computers & Operations Research, 34: 3458–3470.

[2] Ahmadi S., Osman I.H. 2005. Greedy random adaptive memory programming search for the capacitated clustering problem. European Journal of Operational Research, 162: 30-44.

[3] Aiex R.M., Resende M.G.C., Ribeiro C.C. 2002. Probability distribution of solution time in GRASP: An experimental investigation. Journal of Heuristics, 8: 343-373.

[4] Aiex R.M., Resende M.G.C., Ribeiro C.C. 2007. TTTPLOTS: A perl program to create time-to-target plots. Optimization Letters, 1: 355-366.

[5] Albuquerque, L.L., Almeida, A.T., Cavalcante, C.A.V. 2009. Aplicabilidade da programação matemática multiobjetivo no planejamento da expansão de longo prazo da geração no Brasil (in Portuguese). Pesquisa Operacional, 29(1): 153-177.

[6] Armentano V.A., França Filho M.F. 2007. Minimizing total tardiness in parallel machine scheduling with setup times: An adaptive memory-based GRASP approach. European Journal of Operational Research, 183: 100-114.

[7] Coello C.A.C. 2000. An updated survey of GA-based multiobjective optimization techniques. ACM Computing Surveys, 32(2): 109–143.

[8] Czyzak P., Jaszkiewicz A. 1998. Pareto simulated annealing – a metaheuristic technique for multiple objective combinatorial optimization. Journal of Multi-Criteria Decision Analysis, 7: 34–47.

[9] Deb K. 2004. Multi-objective optimization using evolutionary algorithms. England: John Wiley & Sons Ltd.

[10] Ehrgott M., Gandibleux X. 2000. A survey and annotated bibliography of multiobjective combinatorial optimization. OR Spektrum, 22: 425–460.

[11] Fernandes E.R., Ribeiro C.C. 2005. A multistart constructive heuristic for sequencing by hybridization using adaptive memory. Electronic Notes in Discrete Mathematics, 19: 41-47.

[12] Feo T.A., Resende M.G.C. 1995. Greedy randomized adaptive search procedures. Journal of Global Optimization, 6: 109–133.

[13] Feo T.A., Resende M.G.C, Smith S.H. 1994. A greedy randomized adaptive search procedure for maximum independent set. Operations Research, 42: 860-878.

[14] Fiehler, K., Bannert, M.M., Bischoff, M., Blecker, C., Stark, R., Vaitl, D., Franz, V.H. & Rösler, F. 2010. Working memory maintenance of grasp-target information in the human posterior parietal cortex. NeuroImage, 1: 1–11.

[15] Fleurent C., Glover F. 1999. Improved constructive multistart strategies for the quadratic assignment problem. INFORMS Journal on Computing, 11: 198-204.

[16] Gandibleux X., Fréville A. 2000. Tabu search based procedure for solving the 0-1 multiobjective knapsack problem: The two objectives case. Journal of Heuristics, 6: 361–383.

[17] Gandibleux X., Ehrgott M. 2005. 1984–2004 – 20 Years of Multiobjective Metaheuristics. But What About the Solution of Combinatorial Problems with Multiple Objectives?. In: Coello AA, Aguirre AH, Zitzler E (Eds.). Evolutionary Multi-Criterion Optimization. Berlin, Springer: 33–46.

[18] Hansen P. 1997. Tabu search for multiobjective optimization: MOTS. Technical Report. Technical University of Denmark. Paper presented at The 13th International Conference on Multiple Criteria Decision Making. Cape Town, South Africa, 1997.

[19] Hoos H.H., Stützle T. On the empirical evaluation of Las Vegas algorithms â-Position paper. Technical report, Computer Science Department, University of British Columbia, 1998.

[20] Hoss H.H., Stützle T. 1998. Evaluation Las Vegas algorithms - Pitfalls and remedies. In: Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, p.238-245.

[21] Jaskiewicz A. 2002. On the performance of multiple objective genetic local search on the 0/1 knapsack problem: A comparative experiment. IEEE Transaction on Evolutionary Computation, 6(4): 402–412.

[22] Jones D.F., Mirrazavi S.K., Tamiz M. 2002. Multi-objective metaheuristics: An overview of the current state-of-art. European Journal of Operational Research, 137: 1–19.

[23] Lourenço H.R., Pinto J., Portugal R. 1998. Metaheuristics for The Bus-Driver Scheduling Problem. Economics Working Papers 304, Department of Economics and Business, Universitat Pompeu Fabra.

[24] Lins I.D., Droguett E.L. 2009. Multiobjective optimization of availability and cost in repairable systems design via genetic algorithms and discrete event simulation. Pesquisa Operacional, 29(1): 43-66.

[25] Murata T., Ishibuchi H., Gen M. 2001. Specification of genetic Search directions in cellular multi-objective genetic algorithms. In: Zitzler E, Deb K, Thiele L, Coello CAC, Corne D (eds.). Evolutionary Multi-Criterion optimization, First International Conference, EMO. Lecture Notes in Computer Science. Zurich: Springer, 1: 82–95.

[26] Murphey R., Pardalos P.M., Resende M. 2000. Frequency Assignment Problems. Handbook of Combinatorial Optimization. Kluwer: 295–377.

[27] Resende M.G.C., Ribeiro C.C. 2003. Greedy randomized adaptive search procedures. In: Glover F, Kochenberger G (eds.). Handbook of Metaheuristics. Boston, Kluwer: 219–249.

[28] Resende M.G.C., Ribeiro C.C. 2005. GRASP with path-relinking: Recent advances and applications. In: Ibaraki T, Nonobe K, Yagiura M (eds.). Metaheuristics: Progress as Real Problem Solvers. Boston, Kluwer: 29-63.

[29] [29] Ribeiro C.C., Rosseti I. 2007. Efficient parallel cooperative implementations of GRASP heuristics. Parallel Computing, 33: 21-35.

[30] Ulungu E.L., Teghem J. 1995. The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. Foundations of Computing and Decision Sciences, 20(2): 149–165.

[31] Ulungu E.L., Teghem J., Ost C. 1998. Efficiency of interactive multi-objective simulated annealing through a case study. Journal of the Operational Research Society, bf 49: 1044–1050.

[32] Van Veldhuizen D.A., Lamont G.B. 2000. Multiobjective evolutionary algorithms: Analyzing the state-of art. Evolutionary Computation, 8(2): 125–147.

[33] Vianna D.S., Arroyo J.E.C. 2004. A GRASP algorithm for the multi-objective knapsack problem. In: XXIV International Conference of the Chilean Computer Science Society (XXIV SCCC). Washington: IEEE Computer Society: 69–75.

[34] Vianna D.S., Dianin, M.F.V. 2013. Local search-based heuristics for the multiobjective multidimensional knapsack problem. Production Journal, 23(3): 478-487.

[35] Visée M., Teghem J., Pirlot M., Ulungu E.L. 1998. Two-Phases Method and Branch and Bound Procedures to solve the Bi-objectives knapsack Problem. Journal of Global Optimization, 12: 139–155.

[36] Zitzler E., Thiele L. 1999. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. IEEE Transactions on Evolutionary Computation, 3(4): 257–271.

[37] Zitzler E., Laumanns M., Thiele L. 2002. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. In: Giannakoglou K, Tsahalis D, Periaux J, Papailou P, Fogarty T (eds.). EUROGEN 2001, Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems. Athens, Greece: 95–100.