A GRASP algorithm for the multi-objective knapsack problem

Dalessandro Soares Vianna and **José Elias Claudio Arroyo**
Universidade Candido Mendes – UCAM-Campos,
Núcleo de Desenvolvimento e Pesquisa em Informática – NPDI,
Campos dos Goytazazes, RJ 28040-320, Brasil
{dalessandro, jclaudio}@ucam-campos.br

Abstract

In this article, we propose a Greedy Randomized Adaptive Search Procedure (GRASP) to generate a good approximation of the efficient or Pareto optimal set of a multi-objective combinatorial optimization problem. The algorithm is applied for the 0/1 knapsack problem with *r* objective functions. This problem is formulated as *r* classic 0/1 knapsack problems. *n* items, each one with *r* costs and *r* weights, have to be inserted in *r* knapsacks with different capacities in order to maximize the *r* total costs (objectives). The algorithm is based on a weighted scalar function (linear combination of the objectives), for it is necessary to define a preference or weight for each objective.

At each iteration of the algorithm, a preference vector is defined and a solution is built considering the preferences of each objective. The found solution is submitted to a local search trying to improve its weighted scalar function.

In order to find a variety of efficient solutions, we use different preference vectors, which are distributed uniformly on the Pareto frontier.

The proposed algorithm is tested on problems with $r = 2, 3, 4$ objectives and $n = 250, 500, 750$ items. The quality of the approximated solutions is evaluated comparing with the solutions given by two genetic algorithms from the literature.

Keywords: GRASP algorithm, multi-objective combinatorial optimization, knapsack problem.

1. Introduction

Many practical optimization problems, generally, involve simultaneous minimization (or maximization) of several conflicting decision criteria. For example, in the design of a complex hardware/software system, often the cost of such system is to be minimized, while maximum performance is desired. There is an opposition between these criteria. So decision maker has to select the best compromise solution, taking into account the preference of the criteria.

The goal of multi-objective 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 to the research of 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 (Ehrgott and Gandibleux, 2000). These methods are generally valid to bi-objective (r = 2) problems but can not be adapted easily to a higher number of objectives. Also, the 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 because of their efficiency, generality and relative simplicity of implementation. These techniques generate good approximated solutions in a short computational time. Several researches have proposed heuristic procedures to solve MOCO problems (Coello, 2000; Ehrgott and Gandibleux, 2000; Jones *et al*., 2002; Van Veldhuizen and Lamont, 2000).

The literature on the multi-objective knapsack problem is rather scarce. The methods proposed by Ulungu and Teghem (1995) and Visée *et al.* (1998) are based on exact algorithms, Ben *et al.* (1999), Jaskiewicz (2002) and Zitzler and Thiele (1999) use genetic algorithms, the methods of Gandibleux and Frévile (2000) and Hansen (1997) are based on tabu search, and the methods proposed by Czyzak and Jaskiewicz (1998), Teghem *et al*. (2000) and Ulungu *et al.* (1998) are based on simulated annealing.

The overview of multi-objective metaheuristic methods, published by Jones *et al.* (2002), shows that 70% of the articles utilize genetic algorithms as the primary metaheuristic, 24% simulated annealing, and 6% tabu search. Jones *et al.* (2002) comments that, there is no sign in the literature reviewed of the newer metaheuristic techniques, such as GRASP, being applied in the multi-objective case. More recently, Festa and Resende (2004) publish an annotated bibliography of the GRASP metaheuristic and there are not references of GRASP application to MOCO problems.

In this paper we propose the GRASP algorithm to solve MOCO problems generating a set of approximated efficient solutions. The algorithm is tested solving the 0/1 knapsack problem with multiple objectives. The performance of the proposed algorithm is compared to two genetic algorithms from de literature: **MOGLS** (*Multi-objective Genetic Local Search*) suggested by Jaskiewicz (2002) and **SPEA2** (Zitzler *et al.*, 2002), which is an improved version of the genetic algorithm **SPEA** (*Strength Pareto Evolutionary Algorithm*) proposed by Zitzler and Thiele (1999).

The organization of the paper is as follows. In the next section, we present the formulation of a MOCO problem, give the uses concepts and present a formal definition of the multi-objective knapsack problem. In section 3, we discuss with more details the proposed GRASP algorithm. We present computational results in Section 4. Finally Section 5 contains our concluding remarks.

2. Multi-objective Optimization

2.1. 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* $x \in \Omega$.

The image of a solution $x \in \Omega$ is the point $z = f(x)$ in the objective space. A point z *dominates* z' if $\forall j \; z_j = f_j(x) \geq z'_j = f_j(x')$, and $z_j > z'_j$ for at least one j. A solution x dominates x' if the image of x dominates the image of x'. A solution $x^* \in \Omega$ is *Pareto optimal* (or *efficient*) if there is no $x \in \Omega$ such that $z = f(x)$ dominates $z^* = f(x^*)$. A solution $x^* \in S \subset \Omega$ is *nondominated* if there is no $x \in S$ such that $z = f(x)$ dominates $z^* = f(x^*)$.

2.2. Multi-objective knapsack problem (MOKP)

In the literature are studied different versions of the 0/1 multi-objective knapsack problem (Zitzeler and Thiele, 1999; Gandibleaux and Fréville, 2000). In this paper we use the same formulation considered by Zitzeler and Thiele (1999) and Jaskiewicz (2002) in their experiments, which considers the multi-objective problem by allowing r knapsacks with different capacities. This problem can be formulated as follows:

$$\textit{Maximize} \quad f_j(\boldsymbol{x}) = \sum_{i=1}^{n} c_{ij} x_i, \qquad \textit{subject to} \quad \sum_{i=1}^{n} w_{ij} x_i \leq W_j, \qquad j = 1,...,r \qquad \text{and } x_i \in \{0,1\}, \qquad i = 1,...,n$$

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

3. Multi-objective GRASP heuristic

GRASP - Greedy Randomized Adaptive Search Procedure (Feo e Resende, 1995) 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, whose neighborhood is investigated until a local minimum (or maximum) is found during the local search phase. The best overall solution is kept as the result.

3.1. 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: $f(x) = \sum_{j=1}^{r} \lambda_j f_j(x)$, where $\sum_{j=1}^{r} \lambda_j = 1$ and $0 \leq \lambda_j \leq 1$.

The preference vector $\lambda = (\lambda_1, ..., \lambda_r)$, generally, determinates a search direction on the Pareto optimal frontier. With the goal of generating a solution, first, a preference vector λ is defined. For this vector is generated a solution x, whose pondered function $f(x)$ is maximized.

Murata et al. (2001) introduces a way of generating the preference vector distributed uniformly on the Pareto frontier. The vectors are generated combining r non-negatives integers with the sum of s:

$$w_1 + w_2 + ... + w_r = s, \text{ where } w_i \in \{0, 1, 2, ..., s\}$$

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

With the goal of obtaining normalized preferences ($\sum_{j=1}^{r} \lambda_j = 1$) we have $\lambda_j = w_j/s$, $w_j \in \{0, ..., s\}$.

The number of generated vectors for r objectives and for a value of s, $N_r(s)$, is calculated as follows:

$N_2(s) = s + 1$. $\quad N_3(s) = \sum_{i=0}^{s} N_2(i) = \sum_{i=0}^{s} (i+1) = (s+1)(s+2)/2$. $\quad N_4(s) = \sum_{i=0}^{s} N_3(i) = \sum_{i=0}^{s} (i+1)(i+2)/2$.

Procedure BuildSolution (x, α, λ, *lPareto*)
Input
x – solution to be built;
α – percentage used on the definition of the candidates restricted list (*CRL*);
λ – vector of preferences;
lPareto – list of dominant solutions that will be updated with x.
Output
x – built solution.
Begin
01. Insert each element $x_e \notin x$ ($x_e = 0$) in the candidates list *CL* sorted (decreasing) by $\sum_{j=1}^{r} \lambda_j c_{ej} / \sum_{j=1}^{r} w_{ej}$;
02. Let *CRL* be a list with the α% firsth elements of *CL*;
03. Select randomly an element x_e of *CRL*;
04. **While** $x \cup x_e$ does not violate W_j, for j=1, ..., r **do**
05. **Begin**
06. $x \leftarrow x \cup x_e$ ($x_e \leftarrow 1$) ;
07. Remove the element x_e of *CL*;
08. Select randomly an element x_e of *CRL*;
09. **End_while**
10. **For** i=1 **until** |*CL*| **do**
11. **Begin**
12. $x_e \leftarrow$ the i^{th} element of *CL*;
13. **If** $x \cup x_e$ does not violate W_j, for j=1, ..., r **then**
14. **Begin**
15. $x \leftarrow x \cup x_e$ ($x_e \leftarrow 1$) ;
16. **End_if**
17. **End_for**
18. Verify the insertion of x in *lPareto*;
19. **Return** x;
End_BuildSolution

Figure 1: Construction algorithm.

Figure 1 presents the implemented constructive algorithm, which 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 vector of preferences 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 elements with $x_i = 0$. The *CL* list is sorted (decreasing) according to the following ratio:

$$\frac{\sum_{j=1}^{r} \lambda_j c_{ej}}{\sum_{j=1}^{r} w_{ej}} \qquad (1)$$

As showed in line 3, the candidates restricted list (CRL) is composed of the $\alpha \times |LC|$ firsth elements of *CL* list. The loop in lines 4-9 is responsible by the randomization of the algorithm. An item e is randomly selected of *CRL* and inserted in x. This process is repeated while the insertion of e does not violate any restriction of the problem. The loop in lines 10-17 completes the solution x. This stage is greedy respecting the sorting of

CL list. In line 18 is verified if the solution *x* is a dominant solution and, finally, the solution *x* is returned in line 19.

3.2 Local search

```
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;
    lPareto – list of dominant solutions that will be updated with the found solutions.
Output
    x – refined solution.
Begin
01. For i=1 to n do
02.     Marked[i] ← false;
03. while exist an item e such that Marked[e] = false do
04. Begin
05.     y ← x;
06.     Remove the unmarked element y_e∈y (y_e = 1) that presents the smallest value of the ratio (1). Repeat
        this process until any element y_g∉y (y_g = 0) may be chosen for insertion;
07.     y ← BuildSolution (y, β, λ, lPareto);
08.     If f(y) < f(x) then
09.     Begin
10.         x ← y;
11.         For i=1 to n do
12.             Marked[i] ← false;
13.     Else
14.         Let x_e be the unmarked element of x that presents the smallest value of the ratio (1);
15.         Marked[e] ← true;
16.     End_if
17. End_while
18. Return x;
End_LocalSearch
```

Figure 2: Local search algorithm

The local search algorithm is presented in Figure 2. It receives as input parameters the solution *x* to be refined, the percentage β that is used at the solution reconstruction stage, the vector of preferences λ and the *lPareto* list, where the dominant solutions are stored. As output, the algorithm returns the refined solution *x*. The loop in lines 1-2 initializes all the positions of the array *Marked* with false. An item *e* can be removed from the knapsack only if *Marked*[*e*] = false. This array is very important for the algorithm termination. The loop in lines 3-17 is executed while exist elements that can be removed, that is, elements still unmarked. In line 5, the solution *x* is attributed to the auxiliary solution *y*. In line 6 are removed from *y* the elements that present the shortest values of the ratio (1). This process is repeated while exist an element that is out of the knapsack and may be inserted without violates any restriction of the problem. This step is completely greedy. In line 7 the procedure **BuildSolution** is executed completing the construction of the solution *y*. If the new solution, *y*, is better than *x*, then the solution *x* is updated at line 10 and the array *Marked* is reinitialized in lines 11-12. Thus, the process is repeated with the new solution *x*. If *y* is not better than *x*, then, in line 15, is marked the first element that is removed from *y* during the stage described in line 6. In line 18, 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.

3.3. GRASP-MULTI heuristic

Procedure GRASP-MULTI (*N_iter*, α, β)
Input
N_iter – number of GRASP iterations;
α – percentage used at the construction stage;
β – percentage used at the local search stage.
Output
lPareto – list of dominant solutions.
Begin
01. *lPareto* $\leftarrow \varnothing$;
02. **For** *i*=1 **to** *N_iter* **do**
03. **Begin**
04. $x \leftarrow \varnothing$ ($x_i \leftarrow 0$, $\forall i$);
05. Define the vector λ for the iteration *i* according to the preference specification method described at 3.1;
06. $x \leftarrow$ **BuildSolution** (x, α, λ, *lPareto*);
07. $x \leftarrow$ **LocalSearch** (x, β, λ, *lPareto*);
08. **End_for**
09. **Return** *lPareto*;
End_GRASP-MULTI

Figure 3. Implemented GRASP algorithm.

Figure 3 presents the implemented GRASP algorithm that receives as input parameters the number of iterations (*N_iter*), the percentage α used at the reconstruction stage and the percentage β used at the local search stage. As output, the algorithm returns the *lPareto* list, where the dominant solutions are stored. In line 1, the *lPareto* list is initialized. The loop in lines 2-8 executes *N_iter* GRASP iterations. In line 4, the solution *x* is initialized. In line 5, the vector of preference λ is defined following the method described at Subsection 3.1. The solution *x* is built in line 6 and refined in line 7. Finally, the *lPareto* list is returned.

4. Computational experiments

All computational experiments have been performed on a 2GHz Pentium IV processor with 1 Gbyte of RAM memory. The **GRASP-MULTI** algorithm was implemented in C using version 6.0 of the Microsoft Visual C++ compiler.

4.1. Test instances

In this work we use the same set of instances that was used by Zitzeler and Thiele (1999). 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 the total weight regarding the corresponding knapsack: $W_j = 0.5\sum_{i=1}^{n} w_{ij}$.

The problem instances and the approximations to the nondominated set generated by **SPEA** (Zitzeler and Thiele, 1999) and **SPEA2** (Zitzler *et al*., 2002) are available at: http://www.tik.ee.ethz.ch/~zitzler/testdata.html. The results of the SPEA2 are available only for instances with 750 items.

Jaskiewicz (2002) uses also the same set of instances to test the MOGLS algorithm. Results of his experiment are available at: http://www-idss.cs.put.poznan.pl/~jaszkiewicz/mokp/.

4.2. Evaluation of computational results in multi-objective 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 multi-objective optimization, however, there is no natural single measure that is able to capture the quality of an approximation set H to the Pareto optimal set or reference set R.

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

Cardinal measure: NRS = $|H \cap R|$ (number of reference solutions found by the heuristic method).

Distance measure (proposed by Czyzak and Jaszkiewicz (1998) and Ulungu et al. (1998)): $D_{avg} = \frac{1}{|R|}\sum_{z \in R} \min_{z' \in H} d(z', z)$ and $D_{max} = \max_{z \in R}\{\min_{z' \in H} d(z', z)\}$, where d is defined by $d(z', z) = \max_{j=1,\ldots,r}\left\{\frac{1}{\Delta_j}\left(z'_j - z_j\right)\right\}$, $z' = (z'_1,\ldots,z'_r) \in H$, $z = (z_1,\ldots,z_r) \in R$ 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 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.

4.3. Results comparison

It was executed *N_iter* = 1000 GRASP iterations for each instance described in Table 1. In this table are described, for each instance, the number r of objectives and the number n of items. The computational time, in seconds (s), consumed by the **GRASP-MULTI** algorithm is described at the last column.

The values of the parameters α (used at the construction stage) and β (used at the local search stage) that obtained the best results were 10% e 50%, respectively. It have been noted that is more efficient to use $\beta > \alpha$ because it allows that, at local search stage, a superior number of elements may have possibilities of being inserted at the solution under analysis.

First, the proposed algorithm is compared with the genetic algorithm **MOGLS** (Jaskiewicz, 2002).

Tables 2 presents comparative results between the proposed **GRASP-MULTI** algorithm and the **MOGLS** algorithm. In the second column is presented the number $|R|$ of reference solutions. In the following

columns are presented, for each algorithm and for each instance, the total number of obtained solutions (*TNS*), the number of reference solutions (*NRS*) and the average distance (D_{avg}).

Instance			Consumed time(s)
Name	**Objectives**	**Items**	
kn250_2	2	250	10
kn250_3	3	250	94
kn250_4	4	250	467
kn500_2	2	500	67
kn500_3	3	500	542
kn500_4	4	500	1960
kn750_2	2	750	204
kn750_3	3	750	1259
kn750_4	4	750	4120

Table 1: Execution time for each test instance.

Instance	**|R|**	**MOGLS**			**GRASP-MULTI**		
		TNS	***NRS***	D_{avg}	***TNS***	***NRS***	D_{avg}
kn250_2	209	178	0	1.22e-02	209	209	0
kn250_3	5485	1464	1078	1.75e-02	4279	4242	4.40e-05
kn250_4	22674	3952	3299	2.87e-02	19190	19099	1.23e-04
kn500_2	357	249	0	1.68e-02	357	357	0
kn500_3	13314	2360	1460	2.70e-02	11806	11803	3.54e-07
kn500_4	46489	5699	4204	4.00e-02	42157	42157	6.47e-06
kn750_2	555	356	3	1.75e-02	552	552	3.14e-06
kn750_3	20286	2910	1073	2.97e-02	18561	18561	1.64e-07
kn750_4	64796	6971	5178	4.53e-02	59515	59515	8.20e-06

Table 2: Comparison of **MOGLS** and **GRASP-MULTI** (e-k = $\times 10^{-k}$).

The results show that the **GRASP-MULTI** algorithm obtain a larger number of reference solutions and this algorithm performs better than **MOGLS** algorithm regarding the distance measure.

The proposed algorithm is also compared with the genetic algorithm **SPEA2** (Zitzler *et al.*, 2002) for instances with n = 750 items.

Instance	**|R|**	**SPEA2**			**GRASP-MULTI**		
		TNS	***NRS***	D_{avg}	***TNS***	***NRS***	D_{avg}
kn750_2	552	250	0	1.51e-01	552	552	0
kn750_3	18670	300	109	2.46e-01	18561	18561	0
kn750_4	59636	350	121	3.58e-01	59515	59515	0

Table 3: Comparison of **SPEA2** and **GRASP-MULTI** (e-k = $\times 10^{-k}$).

Tables 3 shows the number of reference solutions |R| generated by the **SPEA2** and **GRASP-MULTI** algorithms. In this table is also showed, for each algorithm and for each instance, the total number of obtained solutions (*TNS*), the number of reference solutions (*NRS*) and the average distance (D_{avg}).

The results show that the **GRASP-MULTI** algorithm outperforms the **SPEA2** algorithm for all the instances with n = 750 items.

The superiority of the proposed algorithm (comparing with **MOGLS** and **SPEA2** algorithms) can also be viewed in Figure 4, where the solutions obtained by each algorithm for the instance kn750_2 are presented.

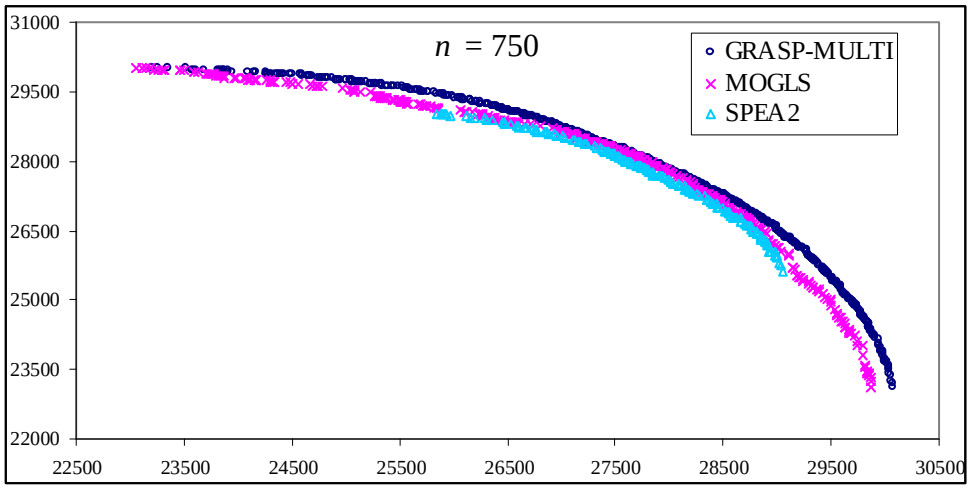

Figure 4: Dominant solutions obtained by each algorithm.

It is important to emphasize that the stop condition of the genetic algorithm **SPEA2** was fixed in 500 iterations. This algorithm, at each iteration, evaluates populations that vary between 150 and 400 solutions. The genetic algorithm **MOGLS** has as stop condition the evaluation of 50×*Tpop* solutions, where *Tpop* is the population size that can vary between 150 and 250 solutions.

5. Conclusion

In this paper, we propose a GRASP algorithm to generate a good approximation of the set of efficient or Pareto optimal solutions of a multi-objective combinatorial optimization problem. It is applied to solve the knapsack problem with *r* objectives and it is compared with the algorithms **MOGLS** (Jaskiewicz (2002)) and **SPEA2** (Zitzler et al. (2002)). The computational results show that the heuristic proposed, **GRASP-MULTI**, obtained, for all instances tested, a number of reference solutions very superior comparing with the other heuristics. It is also the most efficient heuristic when the quality of the obtained solutions (measured by the distance between solutions – D_{avg}) is compared.

Based on the obtained results, it is concluded that the metaheuristic GRASP can be applied efficiently to solve multi-objective combinatorial optimization problem.

Acknowledgment: This work was partially funded by the prefecture of Campos dos Goytacazes city. The computer where the computational experiments were executed was acquired with resource of CNPq (National Agency of Development and Research)

References

1. Ben, A.F., Chaouachi, J., and Krichen, S. (1999). “A hybrid heuristic for multiobjective knapsack problems”. Em S. Voss, S. Martello, I. Osman, and C. Roucairol (eds.), *Metaheuristics: Advances and trends in local search paradigms for optimization.* Dordrecht: Kluwer, 205-212.

2. Coello, C.A.C. (2000). “An updated survey of GA-based multiobjective optimization techniques”. *ACM Computing Surveys* 32(2), 109-143.
3. Czyzak, P., and Jaszkiewicz, A. (1998). “Pareto simulated annealing - a metaheuristic technique for multiple objective combinatorial optimization”. *Journal of Multi-Criteria Decision Analysis* 7, 34-47.
4. Ehrgott, M., and Gandibleux, X. (2000). “A survey and annotated bibliography of multiobjective combinatorial optimization”. *OR Spektrum* 22, 425-460.
5. Feo, T.A., and Resende, M.G.C. (1995). “Greedy randomized adaptive search procedures”. *Journal of Global Optimization* 6, 109-133.
6. Festa P., and Resende M.G.C. (2004). “An annotated bibliography of GRASP”. *Relatório técnico*. Submitted for the *European Journal of Operational Research*.
7. Gandibleux, X., and 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.
8. Hansen, P. (1997). “Tabu search for multiobjective optimization: MOTS”. *Relatório Técnico*. Technical University of Denmark. Paper presented at *The 13th International Conference on Multiple Criteria Decision Making*. Cape Town, South Africa.
9. 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.
10. Jones, D.F., Mirrazavi, S.K., and Tamiz, M. (2002). “Multi-objective metaheuristics: An overview of the current state-of-art”. *European Journal of Operational Research* 137, 1-19.
11. Murata T., Ishibuchi H., and Gen M. (2001). “Specification of genetic Search directions in cellular multi-objective genetic algorithms”. In Zitzler E., Deb K., Thiele L., Coello C.A.C. and Corne D. (eds.), *Evolutionary Multi-Criterion optimization, First International Conference, EMO. Lecture Notes in Computer Science*. Zurich: Springer, Vol 1, 82-95.
12. Teghem, J., Tuyttens, D., and Ulungu, E.L. (2000). “An iteractive heuristic method for multi-objective combinatorial optimization”. *Computer and Operations Research* 27, 621-634.
13. Ulungu, E.L., and 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.
14. Ulungu, E.L., Teghem, J., and Ost, C. (1998). “Efficiency of interactive multi-objective simulated annealing through a case study”. *Journal of the Operational Research Society* 49, 1044-1050.
15. Van Veldhuizen, D.A., and Lamont, G.B. (2000). “Multiobjective evolutionary algorithms: Analyzing the state-of art”. *Evolutionary Computation* 8(2), 125-147.
16. Visée M., Teghem J., Pirlot M., and 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.
17. Zitzler, E., and Thiele, L. (1999). “Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach”. *IEEE Transactions on Evolutionary Computation* 3(4), 257-271.
18. Zitzler, E., Laumanns M., and Thiele L. (2002). “SPEA2: Improving the Strength Pareto Evolutionary Algorithm”. In K. Giannakoglou, D. Tsahalis, J. Periaux, P. Papailou and T. Fogarty (eds.), *EUROGEN 2001, Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*. Vol. 1, Athens, Greece, 95-100.