INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Solving the Multi-objective Knapsack Problems using the Metaheuristic techniques

Models, Implementations and Results

Quebo Kenge Clemente

April 2010

Abstract

In this paper, we propose a hybrid methodology for solving a very well known combinatorial optimization problem, the knapsack problem. The implemented methodology is used to solve two multi-objective variants of this problem, the multi-objective knapsack problem with one constraint, and the multi-objective knapsack problem with several constraints. This problem appears in many practical situations, such as the selection of investment projects and budgetary control. Typically, to solve this problem we can use exact algorithms, which give us exact solutions, or metaheuristics that does not guarantee exact solutions, but in general, allow us to obtain "good" approximate solutions. However, the application of the exact methods is limited by the size of the instances of the problem and by the available computational resources, while on metaheuristics we can obtain "quality" solutions for large instances using reasonable computational resources. The proposed methodology consists of combining two metaheuristic techniques that have shown good performances in several problems of combinatorial nature: the ant colony optimization and the scatter search method. To evaluate the performance of this method, we compared the results with others calculated by some methods from literature, and the new model has been clearly superior.

Keywords: Metaheuristics, Ant Colony Optimization, Scatter Search, Multi-objective Knapsack Problem, Multi-objective Combinatorial Optimization.

1. Introduction

Suppose that we have a set of items which values and weights are known and a bag with a limited capacity. To fill out the bag with the items in a way that their total value is the highest possible without exceeding the bag's capacity is known as the knapsack problem. This problem is very popular in literature and it has been extensively studied by many authors (see for example, Martello & Toth, 1990, Pisinger, 1995, Visée et al., 1998, Captivo et al., 2003 and Zitzler, 1999). Many real life problems can be formulated like the knapsack problem or one of its variants, for example: loading problems, project selection problems, capital budgeting problems, cutting stock problems. Moreover, it can be found as a subproblem in several models, such as the partition and design of electronic circuits and the choice of a flight crew (Fréville, 2004; Ehrgott & Ryan, 2002; Martello & Toth, 1990).

There are several approaches that can be used to solve this kind of problems. These approaches are classified as exact methods and approximated methods or metaheuristics. The solutions obtained by the exact methods are exact solutions, while those obtained by the metaheuristics are approximate solutions. Due to the lack of computational resources and the problem sizes, the exact methods have limited application, i.e., they become ineffective, in particular on instances of great dimension (see for example, Visée et al., 1998, Captivo et al., 2003 and Luila, 2008). While heuristic methods, though they do not provide exact solutions, they have been used as a reliable alternative, since they have proven themselves as being efficient when dealing with big size instances, since they achieve "good" approximate solutions while using a reasonable computational resources, what would be practically impossible to achieve through the exact methods (Ghosh, 2004; Fréville, 2004).

In this article is presented a hybrid algorithm based on the combination of two metaheuristics, which particularly have been showing good performances in solving several problems of combinatorial nature: the ant colony optimization and the scatter search method. The idea is to introduce some improvements in the ant colony algorithm proposed in Alaya et al. (2007) so it can be combined to the scatter search algorithm proposed by Gomes da Silva et al. (2006) and analyze its application when solving big size instances of the two multi-objective variants of the knapsack problem, the multi-objective knapsack problem with a single

constraint and the multi-objective knapsack problem with several constraints.

The rest of the paper is organized as follows: in section 2 are presented some of the theoretical concepts related to the multi-objective combinatorial optimization and a formal definition of the two variants of the problem in study; sections 3 and 4 are respectively dedicated to the metaheuristics of the ant colonies and scatter search; in section 5 are presented the different strategies of combination of the two metaheuristics in study; in section 6 the results are presented as well as commentaries regarding the computational experiments carried out with the implemented models; and section 7 presents the main conclusions and future research.

2. Multi-objective Combinatorial Optimization

In several real life problems there are usually more than one objective to be met (see for example, Kwark et al., 1996; Teng and Tzeng, 1996). Multi-objective combinatorial optimization problems (MCOPs), is the common designation for combinatorial optimization problems with more than one objective to optimize. These problems are characterized by having a multiplicity of solutions, i.e., there is not a single best solution (the global optimum), but a set of particular feasible solutions called *Pareto* set or *non-dominated* solutions, that are superior to others when considering all objectives. The multiplicity of solutions is explained by the fact that the objectives are conflicting ones. For example, when choosing a new car to buy, several objectives can be considered. On one hand, it is desirable to acquire a car that presents a good performance (in terms of maximum speed and acceleration capacity). On the other hand, the car must be safe, economical and present the minimum of acquisition cost possible.

More formally, a generic MCOP (in the case of maximization problem) can be defined by the following:

$$max\ z\ (x) = \big(z_1(x), z_2(x), \dots, z_i(x), \dots, z_m(x)\big) \quad (2.1)$$

$$subject\ to:$$
$$x \in X = \{x \in \mathbb{N}^n : x \geq 0,\ Ax = b, b \in \mathbb{N}^h\} \quad (2.2)$$

where $z_i(x) = c^i x$ represents the i-th objective function, i.e., $i \in \{1, 2, \dots, m\}$; m is the number of objective functions; c^i represents the vector of values associated to i-th objective function; n is the number of decision variables; X represents the feasible region in the decision space; h is the number of restrictions/constraints; x is the vector of decision variables, i.e., $x = (x_1, x_2, \dots, x_n)^T$; A represent the technological coefficients matrix ($h \times n$); and, b is the vector of the independent terms of the restrictions.

The feasible region in the objective space, i.e., the set of all images of the points of X is defined through the following expression:

$$Z = \{z \in \mathbb{N}^m : z = \big(z_1(x), z_2(x), \dots, z_m(x)\big), x \in X\} \quad (2.3)$$

Definition 2.1 (*Dominance*). Let us consider $z^1 = \big(z_1(x), z_2(x), \dots, z_m(x)\big) \in Z$ and $z^2 = \big(z_1(y), z_2(y), \dots, z_m(y)\big) \in Z$ two vectors of objective values of two feasible solutions $x\ and\ y \in X$. Then, z^1 dominates z^2 if $z_i(x) \geq z_i(y)$ for all $i \in \{1, 2, \dots, m\}$, and $z_i(x) > z_i(y)$ for at list one $i \in \{1, 2, \dots, m\}$, and it is written $z^1 \geqq z^2$ and $z^1 \neq z^2$.

Definition 2.2 (*Non-dominated* solution). A vector $z^1 \in Z$ is called non-dominated if and only if there is not another vector $z^2 \in Z$, such that $z^2 \geqq z^1$ and $z^2 \neq z^1$. Otherwise z^1 is a dominated vector.

Definition 2.3 (*Efficient* solution). A feasible solution $x \in X$ is said to be efficient if it is impossible to find another feasible solution $y \in X$ that is at least so good as x in all objectives and is strictly better in at least one criterion, i.e. a solution x is said to be efficient, if and only if it is impossible to find another feasible solution $y \in X$ such that $z_i(y) \geq z_i(x)$ for all $i \in \{1, 2, \dots, m\}$ and $z_i(y) > z_i(x)$ for one $i \in \{1, 2, \dots, m\}$.

2.1 The 0/1 Multi-objective Knapsack Problems

In this section we present the formal definition of two variants of multi-objective knapsack problems in study. The first variant of this problem, i.e., the 0/1 Multi-objective knapsack problem with a single constraint (MOKP-1) has the following mathematical formulation:

$$max\ z\ (x) = \big(z_1(x), z_2(x), \dots, z_i(x), \dots, z_m(x)\big) \quad (2.4)$$

$$subject\ to:$$
$$\sum_{j=1}^{n} w_j x_j \leq W \quad (2.5)$$
$$x_j \in \{0, 1\}, j \in \{1, 2, \dots, n\}, i \in \{1, 2, \dots, m\} \quad (2.6)$$

where $z_i(x) = \sum_{j=1}^{n} c_j^i x_j$ represent the i-th objective function; n is the number of items/objects; m is the number of objective function; c_j^i represents the value/profit of item j on criterion i; w_j is the weight/cost of item j; and, W represents the overall

knapsack capacity. A decision variable $x_j = 1$, if and only if the item j is included in the knapsack, otherwise $x_j = 0$. In this problem, we assume that W, c_j^i and w_j are positive integers and $w_j \leq W$ with $\sum_{j=1}^{n} w_j > W$ for all $j \in \{1, 2, \dots, n\}$ and $i \in \{1, 2, \dots, m\}$.

The mathematical formulation of the second variant of this problem, i.e., the 0/1 Multi-objective knapsack problem with several constraints (MOKP-2), is identical to the mathematical formulation of MOKP-1, differing only in the number of restrictions. More specifically, the MOKP-2 is formulated as follows:

$$max\ z\ (x) = \big(z_1(x), z_2(x), \dots, z_i(x), \dots, z_m(x)\big) \quad (2.7)$$

$sujbject\ to$:

$$\sum_{j=1}^{n} w_j^i x_j \leq W_i \quad (2.8)$$

$$x_j \in \{0, 1\}, j \in \{1, 2, \dots, n\}, i \in \{1, 2, \dots, m\} \quad (2.9)$$

3. The Ant Colony Optimization

The ant colony optimization (ACO) is a stochastic methodology developed from the observation of the behavior of real ants in the nature. Basically, in this method the problem to be solved (in our case, the MOKP-1/MOKP-2) is modulated as a problem of identification of efficient paths in a graph, taking advantage of the auto-organized and cooperative behavior of the artificial/virtual (Dorigo et al., 1991; Dorigo & Stützle, 2004; Alaya et al., 2007).

Let us consider the transformation of the MOKP-1 (or MOKP-2) in a connected and non-oriented graph $G = (V, E)$, where the set of vertices V represents the problem items ($j \in \{1, 2, \dots, n\}$) and the set of edges E represents the connection between the vertices. According to Alaya et al. (2007), in each iteration/cycle an ant chooses an objective function ($i \in \{1, 2, \dots, m\}$), and based on this choice it selects the most attractive vertices/items, among the ones that are available. Each ant has a memory (a sort of taboo list) that stores the previously selected paths and prevents that each vertex is selected more than once. After these proceedings, an update of the pheromone trails present each vertex is done. More precisely, it is performed an evaporation of pheromones present on each item, and an amount of pheromone are lay on the vertex corresponding to the solution with highest profit in each objective function, i.e., the items related to best profit are rewarded to increase their desirability in the following iterations. The solution of the problem, i.e. the set of the potentially efficient/non-dominated solutions, will be composed by the set of all vertices belonging to the efficient paths previously identified by each ant and/or by the values of the corresponding objective function.

The selection of a generic item/vertex $j \in \{1, 2, \dots, n\}$, obeys a rule that results from the consideration of the quantity of pheromones and the heuristic information associated to this item/vertex, which values depend on the objective function previously selected. This rule is defined through the expressions (Alaya et al., 2007):

$$p_j = \frac{(\tau_j^i)^{\alpha} * (\eta_j^i)^{\beta}}{\sum_{\forall\, y \in Cand} (\tau_y^i)^{\alpha} * (\eta_y^i)^{\beta}} \quad (3.1)$$

$$0 < p_j \leq 1;\ \tau_j^i, \eta_j^i > 0;\ \alpha, \beta \geq 0 \quad (3.2)$$

where: τ_j^i and η_j^i respectively represent the quantity of pheromones and the heuristic information / efficiency associated to the item j on the objective function i; α and β respectively are two parameters that determine the importance of τ_j^i and η_j^i; and, $Cand$ is the list that contains all the candidate items, while y is a generic element of this list.

Depending on the variant of the knapsack problem, the heuristic information can be calculated through expressions: $\eta_j^i = c_j^i / w_j$ and $\eta_j^i = c_j^i / w_j^i$, respectively for the MOKP-1 and MOKP-2. The pheromones update are done through the expression $\tau_j^i = (1 - \rho) * \tau_j^i + \Delta\tau_j^i$, in which ρ represents the evaporation rate ($\rho \in [0, 1]$) and $\Delta\tau_j^i$ represents the update rate, i.e., the amount/quantity of pheromone lay on the item $j \in \{1, 2, \dots, n\}$. This value is defined according to the quality of the constructed solution, more precisely, $\Delta\tau_j^i = 1/(1 + z_i^{'} - z_i^{''})$, where $z_i^{''}$ is the highest profit on the i-th objective function for the current cycle, and $z_i^{'}$ the highest profit achieved in the i-th objective function since the beginning of the run.

To avoid a quick convergence of the results and/or the possibility of the search getting stuck in a local optimum, the quantity of pheromones present in each item is limited between two values, i.e., $\tau_{min} \leq \tau_j^i \leq \tau_{max}$.

The corresponding pseudo-code of the proposed ACO algorithm for the MOKP-1/MOKP-2 is presented in Algorithm 1. This pseudo-code is characterized by the following main procedures:

1) Initialization of the problem variables, i.e., initialization of the pheromone factors and the list that stores the set of all potentially non-dominated/efficient solutions;

2) Solution construction, i.e., each ant of the colony randomly chooses an objective function and based on it selects the most attractive items of $Cand$ list, according to probability rule defined above;

3) Update of the pheromone trails present in the vertex corresponding to the best solutions of the cycle, and update of the list that stores the potentially non-dominated/efficient solutions.

Algorithm 1: The ACO algorithm for the MOKP-1 / MOKP-2

(1) **BEGIN**
(2) $\tau_j^i \leftarrow \tau_{max}$ {Initialization of the pheromone trails associated to each item};
(3) $S \leftarrow \emptyset$ {Initialization of the list that stores all potentially non-dominated/efficient solutions};
(4) Generate a population of ants A;
(5) **REPEAT**
(6) **FOR** ($k = 1$) **TO** $|A|$ **DO**
(7) **BEGIN**
(8) $f_i \leftarrow$ Randomly choose an object function $\{z_1, z_2, \dots, z_i, \dots, z_m\}$;
(9) $s^k \leftarrow \emptyset$ {List that stores the solution constructed by the ant k};
(10) $Cand \leftarrow \{1, 2, \dots., j, \dots., n\}$ {lista de itens candidatos à solução s^k};
(11) **WHILE** ($Cand \neq \phi$) **DO**
(12) $j \leftarrow$ Choose an item with probability p_j $(Cand)$;
(13) $s^k \leftarrow s^k \cup \{j\}$ {Without violating the constraints};
(14) Update $Cand$ {Remove the previously selected j};
(15) **END WHILE**
(16) **END**
(17) **END FOR**
(18) **FOR** (i =1) **TO** m **DO**
(19) **FOR** (j =1) **TO** n **DO**
(20) $\tau_j^i \leftarrow (1 - \rho)\tau_j^i$ {Pheromones evaporation};
(21) $\tau_j^i \leftarrow \tau_j^i + \Delta\tau_j^i$ {$j \in s^k$ with the highest profit};
(22) **IF** ($\tau_j^i < \tau_{min}$) **THEN** $\tau_j^i \leftarrow \tau_{min}$ **END IF**
(23) **IF** ($\tau_j^i > \tau_{max}$) **THEN** $\tau_j^i \leftarrow \tau_{max}$ **END IF**
(24) **END FOR**
(25) **END FOR**
(26) $S \leftarrow S \cup_{k=1}^{|A|} s^k$ {Add s^k potentially non-dominated/efficient};
(27) **UNTIL** (Maximum number of cycles / time reached)
(28) **END**

4. The Scatter Search Method

The scatter search method (SS) is an evolutionary method (i.e., a method based on populations) that obtain solutions by combining others. This method was formally introduced by Glover (1977) while performing a heuristic study for linear integer programming problems. To combine all solutions, the SS method explores pre-selected regions from the feasible region, operating on sets of solutions called reference sets, which are generally composed by the "best" solutions of a population/sample. In the case of the MCOP, particularly the MOKP-1/MOKP-2, the concept of "best" solutions only takes into account the diversity of the solutions. The SS based method proposed in this paper is the result of an adaptation of the SS method proposed by Gomes da Silva et al. (2006), more specifically, the proposed method results of a generalization of the SS method applied to the bi-criterion knapsack problem, for a multi-criterion problem.

The proposed method is organized according to the usual structure/steps of SS method described in Glover (1998), which we introduce in the context of a metaheuristic developed to be combined with the ACO algorithm described in the previous section.

A. The diversification method

This step consists on generating a collection of diverse trial solutions, i.e., the initial set/population of solutions. In our case it consists on applying the ACO method described in the previous section to obtain the initial set of potentially non-dominated/efficient solution.

B. The Improvement Method

This method aims to enhance the solutions from the diversification method. In our case, once the applying

of the ACO method allows us to obtain "quality" solutions, the solution improvement procedure will be applied strictly to solutions generated through the combination method.

C. The reference set generation and update method

Basically, in this method, the "best" solutions are chosen from the main sample to form a set of solutions (the reference set) which lately will be combined to obtain new solutions. In our case, the notion of "best" only takes into account the diversity of solutions from the initial set. More precisely, each solution $s^u \in S$ $(u = 1, 2, \dots, |S|)$ is ordered according to non-decreasing values of the following dissimilarity (Gomes da Silva et al., 2006):

$$d_u^{ref} = \left|\left| s^{ref} - s^u \right|\right| = \sum_{j=1}^{n} \left| x_j^{ref} - x_j^u \right| \quad (4.1)$$

$$u = 1, 2, \dots, |S| \quad (4.2)$$

where s^{ref} represents the reference solution, i.e., the solution from which the dissimilarity values of the others solutions are defined. This solution corresponds to $s^u \in S$ with the highest value of $\sum_{i=1}^{m} z_i(x^u)$ for all $u = 1, 2, \dots, |S|$. After ordering all solutions, the reference set (R) is built from S, by dividing it into groups/clusters with approximately the same number of solutions, and one solution (the mean solution) from each clusters is selected in order to define the reference set, R. We consider a maximum of 20 elements in the reference set.

D. The subset generation method

In this phase the reference set is divided into subsets of solutions to be combined to create new ones. As the combination method is based on the exploration of the distinctions between the solutions in each subset, we should not consider very dissimilar solutions from R. In this case, the subsets will composed by consecutive pairs of solutions from R. Hence, there will be $|R|$ - 1 combination (Gomes da Silva et al., 2006). To avoid a repetitive combination of solutions, a taboo list is created, where all previously combined pairs are stored.

E. The solution combination method

In this method, the solutions from the built subsets are combined in order to obtain the new ones. Let $s^0 = \{z^0, x^0\}$ and $s^1 = \{z^1, x^1\}$ be two potentially non-dominated/efficient solution of one subset, where z^0 and z^1 represent the respective non-dominated vectors, and x^0 and x^1 the respective efficient solutions; in the nomenclature of Glover (1999) s^0 is called the *initiating* solution and s^1 is called the *guiding* solution. The guiding solution is used to identify decision variables whose values should be change in the initiating solution, in order to make it equal to guiding solution, i.e., from the s^0 new solutions are obtained by inserting the characteristics of s^1. More precisely, the items to be removed from s^0 are identified (i.e., items with $x_j^0 = 1$) and also the items that must be inserted into s^0 (i.e., items with $x_j^0 = 0$). Those items with $x_j^0 = 1$ are ordered according to non-decreasing values of their efficiency, and those with items $x_j^0 = 0$ are ordered according to non-increasing values of their efficiency, in a way to remove item with less efficiency and insert items with highest efficiency. The efficiency associated to each item is calculated by using the equations, $\eta_j^i = max_{i=1}^{m}\{c_j^i / w_j\}$ and $\eta_j^i = max_{i=1}^{m}\{c_j^i / w_j^i\}$, respectively for the MOKP-1 and MOKP-2. Since the path from s^0 to s^1 is different from s^1 to s^0, the solutions are swapped and once again the above combination procedure is applied. The corresponding pseudo-code of the proposed SS algorithm for the MOKP-1/MOKP-2 is presented in Algorithm 2. This pseudo-code is characterized by the following main procedures:

1) Initialization of the problem variables;
2) Obtaining the initial population/set by applying the ACO algorithm;
3) Construction and update of the reference set, definition of the subsets and application of the combination method to obtain new solutions;
4) Update of the list containing all potentially non-dominated/efficient solution.

Algorithm 2: The SS algorithm for the MOKP-1 / MOKP-2

(1) **BEGIN**
(2) $S \leftarrow \emptyset$ {Initialization of the list that stores all potentially non-dominated/efficient solutions};
(3) $\hat{S} \leftarrow \emptyset$ {Initialization of list that stores all resulting solutions from the combination method};
(4) $Tabu \leftarrow \emptyset$ {Initialization of the taboo list, i.e., the list that store all the previous combinations};
(5) $S \leftarrow$ Get initial the set of potentially non-dominated/efficient {apply the **ACO Algorithm**};
(6) **REPEAT**
(7) $R \leftarrow \emptyset$ {Initialization of the reference set};

(8) $s^{ref} \leftarrow$ Find the s^u with highest value of $\sum_{i=1}^{m} z_i(x^u)$ ($\boldsymbol{S}$) {The reference solution};
(9) **FOR** ($u = 1$) **TO** $|S|$ **DO**
(10) $d_u^{ref} \leftarrow \sum_{j=1}^{n} \left| x_j^{ref} - x_j^u \right|$ {Compute the dissimilarity values};
(11) **END FOR**
(12) Order the list S according to non-decreasing values of dissimilarity;
(13) $R \leftarrow R \cup_{u=1}^{|S|} s^u$ {Add s^u candidates for the reference set};
(14) **FOR** ($r = 1$) **TO** ($|R| - 1$) **DO**
(15) **IF** ($\{R^r, R^{r+1}\} \notin Tabu$) **THEN**
(16) $\hat{S} \leftarrow \hat{S} \cup$ {Apply the **Combination method** (R^r, R^{r+1})};
(17) $\hat{S} \leftarrow \hat{S} \cup$ {Apply the **Combination method** (R^{r+1}, R^r)};
(18) $Tabu \leftarrow Tabu \cup \{R^r, R^{r+1}\}$ {update the taboo list};
(19) $S \leftarrow S \cup \hat{S}$ {Add $\hat{S}$ potentially non-dominated/efficient};
(20) **END IF**
(21) **END FOR**
(22) **UNTIL** (maximum number of cycles/time reached)
(23) **END**

5. Strategies for Combining the ACO and SS Methods

The implementations made in the last two sections (3 and 4) correspond to a combination of the methods ACO and SS in which the SS method is applied after executing the ACO method. Besides this particular strategy, others strategies for combining both methods may be considered, for example, the combination in which the SS algorithm is executed before applying the ACO algorithm. However, this alternative would require an extra implementation, i.e., a new method to generate the initial set of solutions. Nevertheless, we propose the following combination strategies/variants:

a) **ACO-SS1 Combination** - The two methods are implemented in separate cycles, i.e., the resulting algorithm as two cycles: in the first the ACO method is implemented and in the second the SS. This strategy corresponds to the implementation made in the previous sections.

b) **ACO-SS2 Combination** - Both methods are implemented in a common cycle, more precisely, the SS method is incorporated in the cycle corresponding to the ACO method. The idea is to use both methods for a reciprocal improvement of the results produced individually. In other words, after the construction of solutions by every ants (in each cycle), the SS method is used to improve and/or to find new solutions, which will provide new "clues" to the ants in the following cycles.

c) **ACO-SS3 Combination** - Corresponds to an extension of the ACO-SS2 combination, more precisely, after the ACO-SS2 combination, the SS method is executed again.

6. Computational Experiments and Results

The instances of MOKP-1 were randomly generated using the instance generator proposed by Klingman et al. (1974), and those corresponding to the MOKP-2 were used the instances defined in Zitzler (1999). These instances and the results of state-of-the-art evolutionary algorithm are accessible at http://www.tik.ee.ethz.ch/sop/download/supplementary/testProblemSuite/ (website of the Institute of Technology of Zurich). To compare performances, we use the C-measure introduced in Zizler and Thiele (1999):

$$C(S', S'') = \frac{|\{z'' \in S'' : \exists\, z' \in S'; z' \succcurlyeq z''\}|}{|S''|} \qquad (6.1)$$

where, $z' \succcurlyeq z''$ means $z_i'' \leq z_i'$ for all $i \in \{1, 2 \ldots, m\}$, i.e., the vector z'' is weakly dominated by z'. When $C(S', S'') = 1$, it means that all points in S'' are dominated by or equal to the points in S', whereas $C(S', S'') = 0$, means that none of the points in of S'' are covered by the set S'. Note that both $C(S', S'')$ and $C(S'', S')$ have to be considered, since $C(S', S'') \neq 1 - C(S'', S')$.

The performances are done in terms of the values obtained with C-measure corresponding to 10 runs. The algorithmic parameters have been set as displayed in the Table 1.

6.1 Comparison of the different variants

In Table 2 a comparison is made between the three variants of ACO-SS algorithm in C-measure applied to the MOKP-1 instances. From this table it is possible to see that the variant ACO-SS3 produces better results in comparison to ACO-SS2 and ACO-SS1 in the instance with 2 objectives and 100 items, $n100l2$.

More precisely around 77.4% of the results from ACO-SS2 are covered by those of ACO-SS3, whereas only 74.2% of the results of ACO-SS3 are covered by ACO-SS2. 78.6% of the results from ACO-SS1 are covered by those of ACO-SS3, and only 78.3% of the results from ACO-SS3 are weakly dominated by or equals to those of ACO-SS1. Similarly, for the instance $n100l3$ it is observed that the ACO-SS3 algorithm continues to be the best alternative; around 71.8% of the results of ACO-SS1 and 66% of ACO-SS2 are weakly dominated by or equals to the results obtained by ACO-SS3, while only 39.8% and 62.3% of the ACO-SS3 results are covered by those of ACO-SS1 and ACO-SS2, respectively. However, for the instance $n250l2$ ACO-SS1 algorithm appears to have better results than both ACO-SS1 and ACO-SS3. In the rest of instances the ACO-SS2 is strictly better than the other two algorithms. There is therefore a certain balance between the ACO-SS2 and ACO-SS3 algorithms, taken that both are strictly better than the others in two instances.

Comparing the performance of the three algorithms applied to MOKP-2, it is observed again that none of the algorithms is strictly better than the others in all tested instances, i.e., there is a balance between ACO-SS1 and ACO-SS2 algorithms, both show good performances in the same number of instances (see Table 3) the ACO-SS1 algorithm in instances $ztn250l2$, $ztn500l2$ and $ztn750l2$; while ACO-SS2 shows better performances in instances $ztn100l2$, $ztn100l3$ and $ztn250l3$. It is also seen that ACO-SS3 is not in any way better than ACO-SS1 and ACO-SS2 in every tested instance.

Figure 1 and Figure 2 display the compromise surfaces found by ACO-SS1, ACO-SS2 and ACO-SS3 for the bi-objective instances with 500 items, respectively for the MOKP-1 and MOKP-2, corresponding to the overall potentially non-dominated sets found over 10 runs. These figures show that for the MOKP-1 there is a small balance between the three variants, while for the MOKP-2 we can see that ACO-SS1 is clearly the best algorithm, and we can also see that the returned potentially non-dominated/efficient sets are not well dispersed along the Pareto frontier.

6.2 Comparison of ACO-SS with evolutionary algorithms

Hereunder, we compared for each instance of the MOKP-2 the variants which had better performance in the previous section with following state-of-the-art evolutionary algorithm: *FFGA* (Fonseca & Fleming, 1993); *HLGA* (Hajela & Lin, 1992); *NPGA* (Horn et al., 1994); *NSGA* (Srinivas & Deb, 1994); *SPEA* (Zitzler & Thiele, 1999); *VEGA* (Schaffer, 1985).

Each column of Table 4 shows the values of the C-measure corresponding to the comparison between the algorithm ACO-SS (column 1) and a set of algorithms: FFGA (column 2), HLGA (column 3), NPGA (column 4), NSGA (column 5), SPEA (column 6) and VEGA (column 7). From this table it is possible to see that the results obtained from the ACO-SS method are clearly better than those obtained from the other algorithms. For example, it is observed that there is no scenario where the results from FFGA and HLGA algorithm cover those obtained from ACO-SS algorithm, whereas the results from ACO-SS cover 100% of those algorithms. In the other algorithms, only a small part of the ACO-SS results appear to be weakly dominated, which obviously makes them not better than the ACO-SS. The only exception is found in instance $ztn250l2$ where the results from Zitzler and Thiele (1999) are better than those of ACO-SS; around 18.7% of the results from SPEA method are weakly dominated by or equals to those of ACO-SS algorithm, while about 59% of the ACO-SS results are covered by those of the SPEA method.

7. Conclusions and Future Research

We proposed in this work a hybrid algorithm, based on ACO and SS methods, developed to solve two variants of the multi-objective knapsack problem (MOKP-1/MOKP-2). In the developed methodology, the results are initially generated by the ACO algorithm and later refined and intensified by the SS method, which results on three alternatives/strategies, named: ACO-SS1, ACO-SS2 and ACO-SS3 algorithm. The first alternative consists into executing both methods in two different cycles, where the first is related to the ACO algorithm and the second to SS method. In the second alternative, the two methods are used for a reciprocal improvement of the results produced individually; the method SS is called in each cycle of the ACO method. Finally the third alternative consists into extending the alternative ACO-SS2, i.e., SS is executed again at the end.

The experimental results showed that none of the three variants is strictly better than the others when considering all tested instances. However, some variants revealed to be clearly superior in two types of instances. When compared to others methods in the reviewed literature, the ACO-SS method is by far the

best algorithm in almost every tested instances. The results obtained from the computational experiments show that the combination of the ACO and SS methods was able to provide interesting results. Although these results are better than those of the compared algorithms, the applied models must be subjected to further studies in view to improve the performance, in particular in terms of dispersion along the Pareto frontier. Furthermore, other improvements can be made in order to reduce memory usage and time spent.

Variant	α	β	#ACO	#SS	#Ants	ρ	τ_{min}	τ_{max}	|R|
ACO-SS1	1	4	4000	60	100	0.01	0.01	6	20
ACO-SS2	1	4	2500	8	100	0.01	0.01	6	20
ACO-SS3	1	4	2000	10	100	0.01	0.01	6	20

Table 1 - Parameter settings

		Compared Variants					
	Instance	C (1, 2)	C (2, 1)	C (1, 3)	C (3,1)	C (2, 3)	C (3, 2)
Average	n100l2	0,776	0,757	0,783	**0,786**	0,742	**0,774**
Max		0,846	0,831	0,883	0,929	0,914	0,903
Min		0,724	0,613	0,667	0,636	0,571	0,651
Average	n100l3	0,425	0,713	0,398	**0,718**	0,623	**0,660**
Max		0,498	0,779	0,494	0,758	0,680	0,722
Min		0,378	0,628	0,356	0,647	0,535	0,555
Average	n250l2	**0,661**	0,437	**0,649**	0,465	0,501	0,610
Max		0,888	0,750	0,876	0,629	0,856	0,884
Min		0,357	0,169	0,497	0,222	0,218	0,281
Average	n500l2	0,396	**0,413**	0,433	0,364	**0,508**	0,461
Max		0,618	0,646	0,628	0,452	0,863	0,801
Min		0,201	0,185	0,302	0,202	0,200	0,145
Average	n750l2	0,286	**0,346**	0,322	0,296	**0,550**	0,374
Max		0,457	0,504	0,427	0,411	0,913	0,646
Min		0,151	0,213	0,163	0,057	0,294	0,054

Table 2 - Comparison of the 3 variants using the C-measure (MOKP-1)

		Compared Variants					
	Instance	C (1, 2)	C (2, 1)	C (1, 3)	C (3,1)	C (2, 3)	C (3, 2)
Average	ztn100l2	0,497	**0,552**	0,550	0,519	**0,594**	0,494
Max		0,831	0,877	0,924	0,939	0,841	0,758
Min		0,231	0,284	0,191	0,194	0,414	0,186
Average	ztn100l3	0,337	**0,431**	0,374	0,388	**0,462**	0,383
Max		0,764	0,635	0,588	0,723	0,870	0,735
Min		0,104	0,096	0,068	0,167	0,141	0,025
Average	ztn250l2	**0,648**	0,287	**0,682**	0,234	0,433	0,503
Max		1,000	0,932	1,000	0,846	0,990	1,000
Min		0,007	0,000	0,016	0,000	0,000	0,000
Average	ztn250l3	0,030	**0,553**	0,038	0,448	**0,437**	0,382
Max		0,092	0,732	0,100	0,595	0,831	0,895
Min		0,005	0,469	0,008	0,219	0,037	0,020
Average	ztn500l2	**0,583**	0,254	**0,475**	0,325	0,374	0,551
Max		1,000	0,628	1,000	0,772	1,000	1,000
Min		0,139	0,000	0,004	0,000	0,000	0,000

Average	ztn750l2	**0,433**	0,139	**0,461**	0,168	0,367	0,560
Max		0,718	0,353	0,788	0,693	1,000	1,000
Min		0,161	0,000	0,013	0,000	0,000	0,000

Table 3 - Comparison of the 3 variants using the C-measure (MOKP-2)

		Compared Algorithms											
	Instance	(1)-(2)	(2)-(1)	(1)-(3)	(3)-(1)	(1)-(4)	(4)-(1)	(1)-(5)	(5)-(1)	(1)-(6)	(6)-(1)	(1)-(7)	(7)-(1)
Average	ztn100l2	**1,000**	0,000	**1,000**	0,000	**0,991**	0,005	**0,882**	0,058	**0,630**	0,248	**0,904**	0,039
Max		1,000	0,000	1,000	0,000	1,000	0,051	1,000	0,182	0,947	0,446	0,966	0,082
Min		1,000	0,000	1,000	0,000	0,909	0,000	0,714	0,000	0,456	0,000	0,765	0,000
Average	ztn100l3	**1,000**	0,000	**0,999**	0,000	**0,996**	0,001	**0,991**	0,001	**0,928**	0,019	**0,966**	0,006
Max		1,000	0,000	1,000	0,000	1,000	0,005	1,000	0,007	0,994	0,060	1,000	0,022
Min		1,000	0,000	0,989	0,000	0,984	0,000	0,973	0,000	0,767	0,000	0,911	0,000
Average	ztn250l2	**1,000**	0,000	**1,000**	0,000	**1,000**	0,000	**1,000**	0,000	0,187	**0,590**	**1,000**	0,000
Max		1,000	0,000	1,000	0,000	1,000	0,000	1,000	0,000	0,569	0,833	1,000	0,000
Min		1,000	0,000	1,000	0,000	1,000	0,000	1,000	0,000	0,000	0,285	1,000	0,000
Average	ztn250l3	**1,000**	0,000	**1,000**	0,000	**1,000**	0,000	**1,000**	0,000	**1,000**	0,000	**1,000**	0,000
Max		1,000	0,000	1,000	0,000	1,000	0,000	1,000	0,000	1,000	0,000	1,000	0,000
Min		1,000	0,000	1,000	0,000	1,000	0,000	1,000	0,000	0,997	0,000	1,000	0,000
Average	ztn500l2	-	-	**1,000**	0,000	**1,000**	0,000	**1,000**	0,000	**0,927**	0,014	**1,000**	0,000
Max		-	-	1,000	0,000	1,000	0,000	1,000	0,000	1,000	0,081	1,000	0,000
Min		-	-	1,000	0,000	1,000	0,000	1,000	0,000	0,667	0,000	1,000	0,000
Average	ztn750l2	-	-	**1,000**	0,000	**1,000**	0,000	**1,000**	0,000	**0,776**	0,000	**1,000**	0,000
Max		-	-	1,000	0,000	1,000	0,000	1,000	0,000	1,000	0,000	1,000	0,000
Min		-	-	1,000	0,000	1,000	0,000	1,000	0,000	0,556	0,000	1,000	0,000

Table 4 - Comparison of the best variants with evolutionary algorithms using the C-measure (MOKP-2)

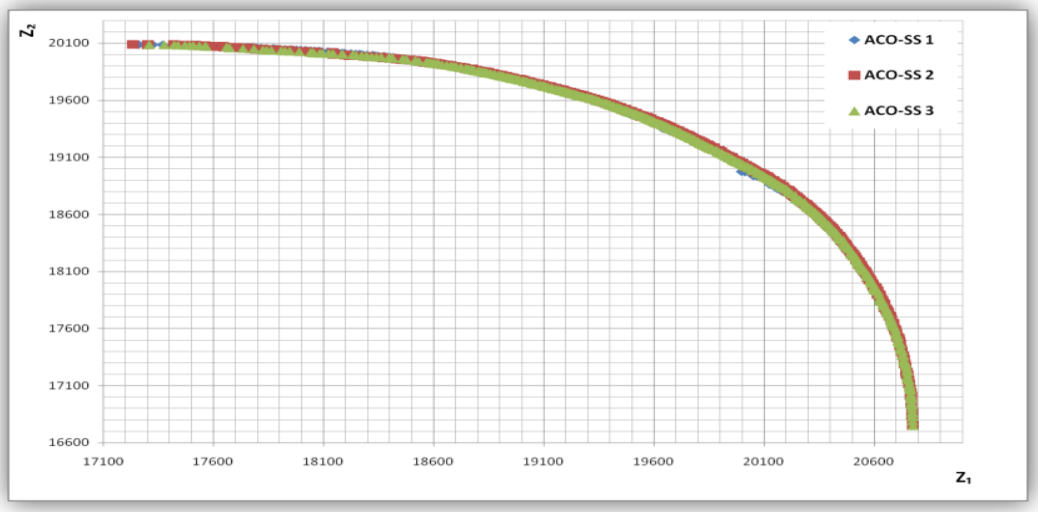

Figure 1 - The compromise surfaces in instance with 500 objects (MOKP-1)

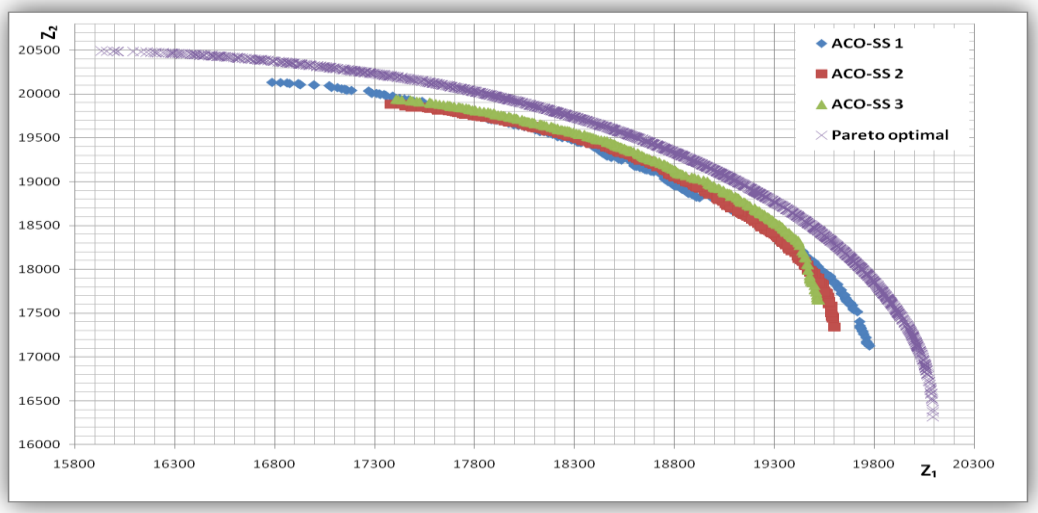

Figure 2 - The compromise surfaces in instance with 500 objects (MOKP-2)

References

Alaya, I., Solnon, C., & Ghédira, K. (2007). Ant Colony Optimization for Multi-objective Optimization Problems. *19th IEEE International Conference on Tools with Artificial Intelligence, pp. 450-457.*

Captivo, M. E., Clímaco, J., Figueira, J., Martins, E., & Santos, J. L. (2003). Solving Bicriteria 0-1 Knapsack Problems using a Labeling Algorithm. *Computers & Operations Research, 30, pp. 1865-1886.*

Dorigo, M., & Stützle, T. (2004). *Ant Colony Optimization*. MIT Press.

Dorigo, M., Maniezzo, V., & Colorni, A. (1991). *The Ant System: An Autocatalytic Optimization Process*. Technical Report 91-016 Revised. Dipartimento di Elettronica, Politecnico di Milano, Itália.

Ehrgott, M., & Ryan, D. M. (2002). Constructing Robust Crew Schedules with Bicriteria Optimization. Journal of Multi-Criteria Decision Analysis, 11 (3), pp. 139-150.

Fonseca, C.M., & Fleming, P.J. (1993). Genetic Algorithms for Multi-objective Optimization: Formulation, Discussion, Generalization. *The Fifth International Conference on Genetic Algorithms, pp. 416-423.*

Fréville, A. (2004). The Multidimensional 0–1 Knapsack Problem: An Overview. *European Journal of Operational Research, 155, pp. 1–21*.

Ghosh, A. (2004). Evolutionary Algorithms for Multi-Criterion Optimization: A Survey. *International Journal of Computing & Information Sciences, 2 (1), pp. 38-57*.

Glover, F. (1977). Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences*, 8 (1), pp. 156-166.

Glover, F (1998). A Template for Scatter Search and Path Relinking. In J. K. Hao, E. Lutton, E. Ronald, M. Schoenauer, & D. Snyers (Eds.), *Artificial Evolution, Lecture Notes in Computer Science 1363* (pp. 13-54). Springer-Verlag.

Glover, F. (1999). Scatter Search And Path Relinking. In D. Corne, M. Dorigo, F. Glover (Eds.), *New Ideas in Optimization* (pp. 297-316). McGraw-Hill.

Gomes da Silva, C., Clímaco, J., & Figueira, J. (2006). A Scatter Search Method for Bi-criteria {0,1}-Knapsack Problems. *European Journal of Operational Research, 169, pp. 373-391.*

Hajela, P., & Lin, C. Y. (1992). Genetic Search Strategies in Multi-criterion Optimal Design. *Structural Optimization, 4, pp. 99-107.*

Horn, J., Nafpliotis, N., & Goldberg, D.E. (1994). A Niched Pareto Genetic Algorithm for Multi-objective Optimization. *First IEEE Conference.*

Klingman, D., Napier, A., & Stutz, A. (1974). NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems. *Management Science, 20(5), pp. 814-821.*

Kwark, W., Lee, H., & Lee, C. (1996). Capital Budgeting With Multiple Criteria and Multiple Decision Makers. *Review of Quantitative Finance and Accounting, 7, pp. 97-112.*

Luila, E. P. (2008). *Problema da Mochila Multi-critério, Aspectos Algorítmicos e Implementação Informática.* Tese de Mestrado, Instituto Superior Técnico, Universidade Técnica de Lisboa.

Martello, S., & Toth. P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. New York: John Wile & Sons, Inc.

Pisinger, D. (1995). *Algorithms for Knapsack Problems.* PhD thesis, Dept. of Computer Science, University of Copenhagen.

Schaffer, J. (1985). Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. *International Conference on Genetic Algorithms, Lecture Notes in Computer Science, pp. 93-100.*

Srinivas, N., & Deb, K. (1994). Multi-objective Optimization using Non Dominated Sorting in Genetic Algorithms. *Evolutionary Computation, 2, pp. 221-248.*

Teng, J. Y., & Tzeng, G. H. (1996). A Multi-objective Programming Approach for selecting Non-independent Transportation Investment Alternatives. *Transportation Research-B, 30, pp. 291-3.*

Visée, M., Teghem, J., & Ulungu, E.L. (1998). Two-Phases Method and Branch and Bound Procedures to solve the Bi-objective Knapsack Problem. *Journal of Global Optimization, 12, pp. 139-155.*

Zitzler, E. (1999). Evolutionary Algorithms for Multi-objective Optimization: Methods and Applications. PhD thesis, Swiss Federal Institute of Technology, Zurich.

Zitzler, E., & Thiele, L. (1999). Multi-objective Evolutionary Algorithms: A Comparative Case Study, the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation, 3, pp. 257-271.*