Genetic Tabu Search for the Multi-objective Knapsack Problem

Vincent Barichard and Jin-Kao Hao

Vincent.Barichard@info.univ-angers.fr
Jin-Kao.Hao@univ-angers.fr

LERIA - Faculty of Sciences - University of Angers
2, Boulevard Lavoisier, 49045 Angers Cedex 01, France

January 15, 2003

Abstract

We introduce a hybrid algorithm for the 0-1 multi-dimensional multi-objective knapsack problem. This algorithm, called GTS^{MOKP}, combines a genetic procedure and a tabu search operator. The algorithm is evaluated on 9 well-known benchmark instances and shows highly competitive results compared with two state-of-the-art algorithms.

1 Introduction

Given a set of items (or objects), each being associated a vector of profits and weights, the 0-1 multi-dimensional multi-objective knapsack problem (MOKP) consists in selecting a subset of items in order to maximize a multi-objective function while satisfying a set of knapsack constraints. More formally, the MOKP can be stated as follows:

$$\text{MOKP01} \begin{cases} \max & z^j(x) = \sum_{i=1}^{n} c_i^j x_i & j = 1, ..., o \\ \text{s.t.} & \sum_{i=1}^{n} w_i^l x_i \leq b_l & l = 1, ..., m \\ & x_i \in \{0, 1\} & i = 1, ..., n \end{cases}$$

where n is the number of items, x_i is a decision variable o the number of objectives, z^j is the j^{th} component of the multi-objective function z, and m the number of constraints of the problem. Just like the NP-hard 0-1 single multi-dimensional knapsack problem MKP, the MOKP may be used to formulate

many practical problems such as capital budgeting and resource allocation. For resolution purpose, several heuristic algorithms have been developed. We may mention neighborhood algorithms based on simulated annealing [10, 12] and tabu search [4], evolutionary algorithms based on vector evaluated genetic algorithm (VEGA) [9], non-dominated sorting genetic algorithm (NSGA) [11, 2] and strength Pareto evolutionary algorithm (SPEA) [14]. Very recently, hybrid algorithms combining genetic search and local search[1] were also proposed [1, 8]. In particular, the multiple objective genetic local search (MOGLS) algorithm described in [6] proves to be very successful for solving large scale MOKP instances.

In this paper, we are interested in the hybrid approach and introduce GTS^{MOKP}, a genetic tabu search algorithm for the MOKP. Inspired by a genetic tabu search algorithm for the graph coloring problem [3] and the MOGLS algorithm [6], GTS^{MOKP} distinguishes itself from MOGLS essentially by the integration of a powerful neighborhood method (tabu search) within the genetic framework. To assess the performance of GTS^{MOKP}, the algorithm is extensively evaluated on a set of 9 well-known benchmarks and compared with NSGA and MOGLS, two of the most famous algorithms for the MOKP. Experimental results of GTS^{MOKP} are highly competitive on all the tested instances with respect to the competing algorithms.

The paper is organized as follows. In the next section, we introduce the general principles of our hybrid algorithm GTS^{MOKP}. Experimental results as well as comparisons of GTS^{MOKP} with two other algorithms are the subject of the section 3. Last section gives some conclusions and perspectives.

2 Genetic Tabu Search for the MOKP (GTS^{MOKP})

Genetic local search (GLS) is now recognized as a highly competitive approach for tackling hard, single or multi-objective combinatorial problems. For the MOKP, we may mention MOGLS which uses a pure descent algorithm as its local search operator [6].

The general schema of genetic local search is now well-known and becomes a quasi standard for this kind of hybridation. At each generation, a number of crossovers (via random or problem-specific operators) are carried out. For each newly generated configuration (individual) s, a local search operator $LS(s)$ is applied to s in order to improve its quality. Finally, a replacement strategy is applied to decide whether the improved configuration should be accepted in the population.

In this section, we present our hybrid genetic tabu search algorithm GTS^{MOKP} for the MOKP.

[1] The term of local search is often used as synonym of pure descent methods. In this paper, this term is used to encompass all neighborhood based methods such as tabu search and simulated annealing.

2.1 General algorithm

GTS^{MOKP} follows the above general schema for genetic local search. We give the skeleton of GTS^{MOKP} in algorithm 1. More details are given in section 2.2.

Input: a MOKP instance
Output: set of non dominated solutions
  $P \leftarrow$ InitPopulation ($|P|$)
  while not Stop-Condition **do**
    $\lambda \leftarrow$ GetRandomWeightVector (used for fitness assignment, see next section)
    fitness assignment of each individual s in P according to $f(s, r, \lambda)$ (see next section)
    $TP \leftarrow$ The N "best" individuals of P
    $(p_1, p_2) \leftarrow$ ChooseParents(TP)
    $s \leftarrow$ Crossover(p_1, p_2)
    $s \leftarrow$ TabuSearch($s, NumberOfIterations$)
    $P \leftarrow$ AddPopulation(s, P)
  end while

Algorithm 1: GTS^{MOKP}: the genetic tabu search algorithm for the MOKP.

Notice that although it is not explicitly stated in the algorithm, non-dominated solutions are always recorded in an appropriate data structure. It is these non-dominated solutions that constitute the output of the algorithm. We presents in the following subsections the different components of GTS^{MOKP}.

2.2 Search space and fitness assignment

A *configuration* (or an *individual*) s is any binary vector with n components satisfying all the m constraints of the problem. The *search space* S is then defined to be the set of all such binary vectors, which is clearly a subset of $\{0,1\}^n$.

To evaluate the fitness of a configuration s, we use, like in [7], a weighted linear function defined as follows:

$$f(s, r, \lambda) = \sum_{i=1}^{o} \lambda_i * (r_i - z_i(s)) \quad (1)$$

where

- s the configuration to be evaluated;
- $z_i(s)$ (i=1...o) the valuation of s for the i^{th} component of the objective function of the problem;
- λ a weight vector composed of o components which is scalarized to [0..1], and whose i^{th} component λ_i corresponds to the weight given to the i^{th} objective;

- r a o-components reference vector.

Therefore, for each configuration s and a given reference vector r, a fitness value is assigned to s according to the valuations of s with respect to the objectives and the (variable) weight given to each objective. This fitness function (1) defines thus a total order for the configurations of the search space S. This order serves as the basis for the genetic and tabu search operators.

2.3 Genetic search

After assigning a fitness to each individual of P, the current population is sorted according to the fitness values. The first (best) N^2 individuals are copied to a temporary population. From this temporary population, two individuals are randomly selected and then mated using one-point random crossover. After restoring the feasibility, the child is improved by the tabu search algorithm (see next subsection). If the improved individual s_* is better than the worst one of the temporary population, s_* is added to the current population and replaces the oldest individual. Otherwise, s_* is rejected.

2.4 Tabu search (TS)

The TS operator *TabuSearch(s,L)* aims to improve a feasible configuration s produced by the crossover for a maximum of L iterations before inserting the improved s into the population.

This section gives a brief review of the most important Tabu Search features. For a comprehensive presentation of TS, the reader is invited to consult the book of Glover and Laguna [5]. We give below some notations necessary for a good understanding of our TS algorithm.

Neighborhood and move. The neighborhood function $\mathcal{N}$ is defined over the search space S and associates each configuration s to a subset of S. More precisely, let s a configuration, then $\mathcal{N}=\{s' \mid s'$ is feasible and Hammingdistance$(s, s')=1\}$. In other words, for each $s \in S$, one can get one neighboring configuration s' by adding (flipping a variable from 0 to 1) or dropping (flipping a variable from 1 to 0) one item from s in such a way that the knapsack constraints are always satisfied. The operation of flipping a variable to get a neighboring configuration is called a *move*. A move from s to $s' \in \mathcal{N}$ can be identified without ambiguity by the attribute j if s' is obtained by flipping the j^{th} component of s.

Evaluation of the neighborhood. The quality of each neighboring configuration $s' \in \mathcal{N}$ is evaluated by the weighted linear fitness function (1), just like it is done for evaluating any other configuration of the search space S.

[2] $N \leq |P|$ is to be fixed empirically. In this paper, N=20 for $|P| = 150$ to 350.

Tabu list. The role of tabu list is to prevent the search from short-term cycling. Since a move corresponds to flipping a single variable, the index of the flipped variable, say j is classified tabu for the next k iterations. The value k, called also tabu tenure, is a parameter to be fixed empirically. The tabu tenure is reset to 0 for each tabu run.

The skeleton of our TS algorithm used in GTS^{MOKP} is given in algorithm 2.

Input: a feasible configuration s_0, the number of iterations L
Output: a new feasible configuration s
$s \leftarrow s_0$
for $i = 0$ to L **do**
choose the best authorized move
update the tabu list with the chosen move
perform the chosen move in s
update the set of non-dominate configuration with s
end for

Algorithm 2: The Tabu search algorithm.

3 Experimental Results

3.1 Test data

In this section, we will compare our GTS^{MOKP} algorithm with NSGA and MOGLS, two of the most powerful algorithms for the MOKP. The experiments are based on the nine instances that were used in [16]. These instances have 2, 3 and 4 objectives, in combination with 250, 500 and 750 items. Moreover, for each instance, the number of constraints is equal to the number of objectives. The instances were generated randomly with uncorrelated profits and weights, and the capacities of the knapsack constraints were set to be half of the total weight regarding the corresponding constraint. As a result, half of the items are expected to be in the optimal solutions.

3.2 Performance measures

In order to evaluate the results (the trade-off fronts) produced by the different algorithms, we use two measures which are scaling-independent with regard to each objective criterion: *The size of the dominated space ($\mathcal{S}$)* and *the coverage of two sets* [15].

Definition 1 *The size of the dominated space ($\mathcal{S}$). Let $A = (x_1, x_2, ..., x_l) \subseteq X$ be a set of l decision vectors. The function $\mathcal{S}$ gives the volume enclosed by the union of the polytopes $p_1, p_2, ..., p_l$, where each p_i is formed by the intersections of the following hyper-planes arising out of x_i, along with the axes:*

for each axis in the objective space, there exists a hyper-plane perpendicular to the axis and passing through the point $(f_1(x_i), f_2(x_i), ..., f_k(x_i))$. *In the two-dimensional case, each* p_i *represents a rectangle defined by the points* $(0, 0)$ *and* $(f_1(x_i), f_2(x_i))$.

This measure cannot be used to compare two sets relatively to each other. In order to determine the dominance ratio between two sets, we apply a second measure.

Definition 2 *The coverage of two sets. Let* $A, B \subseteq X$ *be two sets of decision vectors. The function* $\mathcal{C}$ *maps the ordered pair* (A, B) *to the interval* $[0, 1]$*:*

$$\mathcal{C}(A, B) := \frac{|\{b \in B \mid \exists\, a \in A : a \geq b\}|}{|B|}$$

The value $\mathcal{C}(A, B) = 1$ *means that all decision vectors in* B *are weakly dominated by* A. *The opposite,* $\mathcal{C}(A, B) = 0$, *represents the situation when none of the points in* B *are weakly dominated by* A.

3.3 Experimental settings

The GTS^{MOKP} algorithm is programmed in CAML and compiled using OCAML. In order to get fair comparisons between our algorithm and other state-of-the-art algorithms, we implemented, also in CAML, NSGA and MOGLS, two algorithms recognized as among the most effective ones for the MOKP. In our implementations, the most important data structures are shared by the three algorithms. This provides us a solid basis for a fair comparison between these algorithms.

Before comparing GTS^{MOKP}, NSGA and MOGLS, we first compared our implementation of NSGA and MOGLS with that of MOMHLib [7]. Results showed that those two implementations give totally comparable performance.

In our experimentation, the following settings are used:

- for NSGA, we set the mutation rate to 0.2, and the neighborhood distance to 0.4, according to [16] and [6];
- for GTS^{MOKP}, we set the number of generations to 50, the number of tabu iterations to $L = 12$ for each run of the tabu algorithm and the tabu tenure to 2.
- for NSGA and MOGLS, we allow always a larger number of generations to give a computation time superior or equal to the time given to GTS^{MOKP} (see table 1).
- the population sizes were set according to the number of items and the number of objectives of the instance, but the same size is used for the three algorithms for a given instance (see table 1).

Instance		Method			
Number of objectives	Number of items	Initial population size (all algorithms)	Number of generation		
			NSGA	MOGLS	GTS^{MOKP}
2	250	150	500	110	50
	500	200	500	140	50
	750	250	500	150	50
3	250	200	1100	160	50
	500	250	1100	170	50
	750	300	1100	170	50
4	250	250	3000	300	50
	500	300	3000	320	50
	750	350	3000	320	50

Table 1: Parameters settings for different algorithms and instances.

Number of Objectives	Number of Items	NSGA	MOGLS	GTS^{MOKP}
2	250	60	54	50
	500	205	150	135
	750	330	200	185
3	250	265	250	210
	500	900	490	465
	750	1800	740	660
4	250	1417	1264	1188
	500	3500	2300	2200
	750	7000	3100	2500

Table 2: Average running times given to each algorithm (seconds).

Table 1 summarizes the settings of the main parameters used by NSGA, MOGLS and GTS^{MOKP}.

Table 2 shows, for each of the three compared algorithms and for each problem instance, the computing time obtained from the above parameter settings[3]. We notice that NSGA is always given more computing time than MOGLS, which is given in turn more computing time than GTS^{MOKP}. Thus, we are sure that GTS^{MOKP} is in no case favored compared with the two other competing algorithms.

[3]Timing is based on binary codes generated by the compiler OCAML and a PC running Linux (Bi-Pentium III 1 Ghz).

Number of Objectives	Number of Items	NSGA	MOGLS	GTS^{MOKP}
2	250	9.52e+7	9.86e+7	9.87e+7
	500	3.91e+8	4.08e+8	4.08e+8
	750	8.36e+8	8.93e+8	8.93e+8
3	250	8.45e+11	9.33e+11	9.35e+11
	500	6.74e+12	7.72e+12	7.73e+12
	750	2.33e+13	2.71e+13	2.72e+13
4	250	6.97e+15	8.11e+15	8.12e+15
	500	1.09e+17	1.35e+17	1.36e+17
	750	5.52e+17	7.19e+17	7.20e+17

Table 3: Average of the size of the dominated space $\mathcal{S}$.

3.4 Comparative results

In this section, we present experimental results of GTS^{MOKP} together with those of NSGA, MOGLS. Results shown below for each instance represent averaged ones from several independent runs.

Table 3 shows the results for the $\mathcal{S}$ measure (size of the dominated space). From the table, one observes first that both GTS^{MOKP} and MOGLS gives significantly better results (larger $\mathcal{S}$ values) than NSGA for all the instances. When comparing GTS^{MOKP} and MOGLS, one observes that GTS^{MOKP} outperforms MOGLS for 7 out of the 9 instances and gives the same results for 2 bi-objective instances (500 and 750 items). One may also notice the remarkable results of the GTS^{MOKP} on the hardest instances.

Following [16], figure 1 summarizes the results on the $\mathcal{C}$ measure (set coverage). On this figure, each small black bar represents the results of the $\mathcal{C}$ measure between two methods, for each problem instance. Because of a low dispersion of the results, the average is significant for the $\mathcal{C}$ measure.

It is observed that once again the results of NSGA are inferior to those of MOGLS and GTS^{MOKP} for all the instances. When comparing MOGLS and GTS^{MOKP}, we observe that the results are in favor of GTS^{MOKP} for all benchmarks instances. Indeed, the values obtained by $\mathcal{C}(MOGLS, GTS^{MOKP})$ are smaller than the values obtained by $\mathcal{C}(GTS^{MOKP}, MOGLS)$. We notice that on several instances (500 items 3 objectives, and 500 items 4 objectives) the factor rate between the results is greater than 10.

In summary, even if GTS^{MOKP} is run in unfavorable conditions (fewer generations and less running times), it gives results which are at least as good as or better than those of NSGA and MOGLS for the 9 benchmark instances according to the two used measures. In order to convince us of this superiority, we also counted for each algorithm the number of solutions which are part of the Pareto optimal front. Because of the size of the problem instances, we only

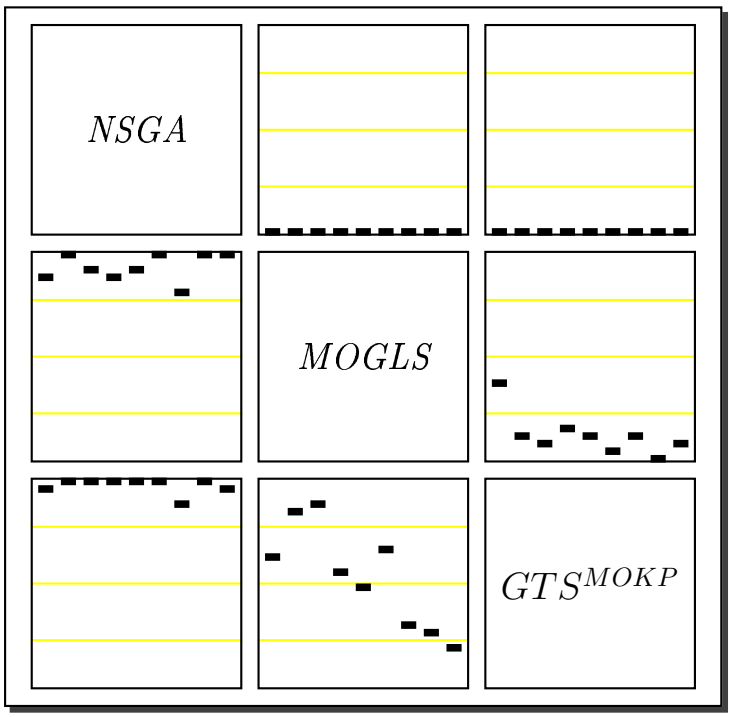

Figure 1: Results of the comparison with $\mathcal{C}$ measure. Each chart contains nine box plots representing the distribution of $\mathcal{C}$ values for a certain ordered pair of algorithms; the three box plots to the left relate to 2 objectives and (from left to right) 250, 500 and 750 items; correspondingly the three middle box plots relate to 3 objectives and the three to the right to 4 objectives. The scale is 0 at the bottom and 1 at the top of each chart. Furthermore, each rectangle refers to algorithm A associated with the corresponding row and algorithm B associated with the corresponding column and gives the fraction of B weakly dominated by A : $\mathcal{C}(A, B)$.

have the Pareto optimal front for the instances with 2 objectives, 250 and 500 items.

Again, the results of GTS^{MOKP} compare very favorably with those of NSGA and MOGLS. Indeed, for the first instance (250 items), 9.33% of the solutions found by GTS^{MOKP} are elements of the Pareto front while this rate is respectively 0.18% for NSGA and 6.51% for MOGLS. For the second instance (500 items), this rate is respectively 0.35% for GTS^{MOKP}, 0.07% for MOGLS and 0% for NSGA.

Finally, let us mention that we carried out another experiment where GTS^{MOKP} is given the same number of generations as for MOGLS. We observed that GTS^{MOKP} obtains much better results than those reported in this paper, thus even better results than MOGLS.

4 Conclusion

In this paper, we have presented GTS^{MOKP}, a highly effective genetic tabu search algorithm for solving the 0-1 multidimensional multi-objective knapsack problem (MOKP). The GTS^{MOKP} algorithm combines the global nature of genetic search and the local nature of tabu search, leading to a better compromise of exploitation and exploration. The performance of GTS^{MOKP} is assessed using a set of 9 well-known benchmark instances and compared extensively with two state-of-the-art algorithms which are NSGA and MOGLS. Experimental results show that GTS^{MOKP}, even with less computing effort, compares very favorably with the competing algorithms on all the tested instances.

Both the genetic part and the tabu search part used in GTS^{MOKP} are straight forward implementation of the basic principles of these two paradigms. For this reason, we are strongly convinced many improvements are possible. For example, for the genetic part, other selection strategies should be experimented, and problem specific crossover operators may be developed. For the TS operator, we believe techniques developed for the classical 0-1 multidimensional multi-objective knapsack problem such as those presented in [13] could be very useful in the context of the MOKP.

References

[1] Corne D. W., Knowles J. D. M-PAES: a memetic algorithm for multi-objective optimization. Proceedings of the 2000 congress on evolutionary computation (CEC 2000).

[2] Deb K., Goel T. Controlled elitist non-dominated sorting genetic algorithms for better convergence. In Proceedings of Evolutionary Multi-Criterion Optimization. 67-81, 2001.

[3] Galinier P., Hao J.K. Hybrid evolutionary algorithms for graph coloring. Journal of Combinatorial Optimization. 3(4): 379-397, 1999.

[4] Gandibleux X., Mezdaoui N., Freville A. A multiobjective Tabu Search procedure to solve combinatorial optimization problems. Lecture Notes in Economics and Mathematical Systems. 455: 291-300, Springer, 1997.

[5] Glover F., Laguna M. Tabu Search, Kluwer Academic Publishers. 1997.

[6] Jaszkiewicz A. On the performance of multiple objective genetic local search on the 0/1 knapsack problem: a comparative experiment. Research report, Institute of Computing Science, Poznan University of Technology. RA-002, 2000.

[7] Jaszkiewicz A. Experiments done with the MOMHLib. http://www-idss.cs.put.poznan.pl/ jaszkiewicz/MOMHLib/

[8] Khor E.F., Tan K.C., Lee T.H. Tabu-Based Exploratory Evolutionary Algorithm for Effective Multi-objective Optimization. In Proceedings of Evolutionary Multi-Criterion Optimization. 344-358, 2001.

[9] Schaffer J.D. Multiple objective optimization with vector evaluated genetic algorithms. Ph. D. thesis. Vanderbilt University. Unpublished, 1984.

[10] Serafini P. Simulated annealing for multiobjective optimization problems. Proc. of 10th Int. Conf. on MCDM. 1: 87-96, 1992.

[11] Srinivas N., Deb K. Multiobjective optimization using non dominated sorting in genetic algorithms. Evolutionary Computation. 2(3): 221-248, 1994.

[12] Ulungu E.L., Teghem J., Fortemps Ph., Tuyttens D. MOSA method: a tool for solving multiobjective combinatorial optimization problems. Journal of Multi-Criteria Decision Analysis. 8: 221-336, 1999.

[13] Vasquez M., Hao J-K. A hybrid approach for the 0-1 multidimensional knapsack problem. Proc. of the 13^{th} Intl. Joint Conference on Artificial Intelligence (IJCAI-01). 1: 328-333, 2001.

[14] Zitzler E., Thiele L. An evolutionary algorithm for multiobjective optimization: the strength Pareto approach. Technical Report 43. 1998.

[15] Zitzler E., Thiele L. Multiobjective optimization using evolutionary algorithms: a comparative case study. Lecture Notes in Computer Science. 1498: 292-301, Springer, 1998.

[16] Zitzler E., Thiele L. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Transactions on Evolutionary Computation. 3: 257-271, 1999.