J Math Model Algor (2008) 7:143–159
DOI 10.1007/s10852-008-9078-9

Scatter Search for the 0–1 Multidimensional Knapsack Problem

Said Hanafi · Christophe Wilbaut

Received: 1 March 2007 / Accepted: 21 December 2007 /
Published online: 27 February 2008

Abstract The evolutionary metaheuristic called scatter search has been applied successfully to optimization problems for several years. In this paper, we apply the scatter search technique to the well-known 0–1 multidimensional knapsack problem. We propose a new relaxation-based diversification generator, which produces an initial population with elite solutions. The computational results obtained for a set of classic and correlated instances clearly show that (1) this generator can also be used as a heuristic for solving the multidimensional knapsack problem; (2) using the population produced by our generator as a starting point for the scatter search algorithm leads to better performance. We also enhance the scatter search algorithm by integrating memory and by using adapted intensification phases. Overall, the results are interesting and competitive compared to other population-based algorithms, such as genetic algorithms.

Keywords Scatter search · Metaheuristic · Multidimensional knapsack problem

Mathematics Subject Classification (2000) 68R05

1 Introduction

Scatter search (**SC**), which combines decision rules and problem constraints, has its origins in surrogate constraint strategies. Scatter search is a population-based metaheuristic that uses a reference set to keep the *best* solutions visited during the search. The notion of best solutions is not limited to the objective value, but also

S. Hanafi (✉) · C. Wilbaut
LAMIH UMR 8530, Université de Valenciennes et du Hainaut-Cambrésis Le Mont Houy,
59313 Valenciennes Cedex9, France
e-mail: said.hanafi@univ-valenciennes.fr

C. Wilbaut
e-mail: christophe.wilbaut@univ-valenciennes.fr

takes solution diversity into account. In general, SC has the advantage of producing a set of elite solutions from which the decision-maker can choose.

The original form of scatter search was proposed at the end of the 1970s [10], though several efficient applications have been proposed more recently. For instance, Gomes da Silva et al. [17] applied scatter search to the bi-criteria multidimensional knapsack problem. Marti et al. [24] used it to solve the problem of assigning teaching assistants to invigilate final exams at the University of Barcelona in Spain. They formulated the problem as an integer program with a weighted objective function combining the two objectives of the problem: maximizing the teaching assistant preferences and minimizing their utilization. Laguna and Marti [21] have proposed many useful implementations for SC, as well as many interesting illustrations.

Glover [14] proposed a template defining and implementing five main components of scatter search and its generalization, path relinking. Our scatter search algorithm is constructed according to these five components: (1) A diversification generation method designed to produce a solution set that provides a reasonable representation of the problem's search space; (2) an improvement method intended to create better feasible and infeasible solutions; (3) a reference set update method designed to create and update the reference set; (4) a subset generation method intended to produce subsets of the solutions in the reference set, which are then combined to create new solutions; (5) a structured combination method designed to create new solutions from the previous solution subsets.

The 0–1 multidimensional knapsack problem (**MKP**) is a well-known optimization problem, which can be formulated as follows:

$$(\text{MKP}) \left[\begin{array}{ll} \max & \sum_{j=1}^{n} p_j x_j \\ \text{subject to:} & \sum_{j=1}^{n} a_{ij} x_j \le c_i \ \forall i \in M = \{1, \ldots, m\} \\ & x_j \in \{0, 1\} \quad \forall j \in N = \{1, \ldots, n\} \end{array} \right.$$

where $N = \{1, \ldots, n\}$ is the set of items, $M = \{1, \ldots, m\}$ is the set of knapsack constraints, and all the data are non negative (i.e., $p_j \ge 0, a_{ij} \ge 0, c_i \ge 0, \forall i \in \{1, \ldots, m\}$, $\forall j \in \{1, \ldots, n\}$). In this formulation, p_j is the profit associated with item $j \in N$, whereas c_i is the capacity of knapsack $i \in M$, and a_{ij} is the consumption of resource $i \in M$ when item j is selected. Without loss of generality, we assume that

$$\max\{a_{ij} : j \in N\} \le c_i \le \sum_{j \in N} a_{ij} \ \forall i \in M.$$

The objective of the problem is to select a subset of items to maximize the profit with respect to the knapsack constraints. In this paper, we also use the following shortcut notation:

$$(\text{MKP}) \ \max\{p^T x : Ax \le c; x \in \{0, 1\}^n\}.$$

Originally used as a capital budgeting model, the MKP is a general resource allocation model [23]. Applications range from cutting stocks [9] to the more recent daily scheduling of satellite photographs [26]. Numerous heuristics and metaheuristics have been applied to the MKP. Many of the most efficient methods for solving the MKP have been presented and reviewed by Fréville [5] and Fréville and Hanafi [6]. To our knowledge, this is the first time that SC has been applied to the MKP. In our

opinion, the comparison of the scatter search algorithm performance on this well-known problem with the performance of other population-based algorithms provides food for thought.

Our scatter search algorithm is based on the five components described above to which we have added several specific ingredients for solving the MKP: (1) We propose a new relaxation-based generator for producing an initial population of elite solutions. As our computational results show, this generator improves the performance of the scatter search algorithm. In addition, our generator can be used on its own as a heuristic for solving the MKP; (2) we employ a common tool for knapsack problem resolution: local search methods based on the efficiency of the variables; (3) in our proposition, infeasible solutions are allowed to be part of the reference set; (4) we implement a dynamic management of the reference set to avoid generating the subsets more than once; (5) we propose enhancing the scatter search by introducing memory and defining complementary intensification phases.

The remainder of the paper is organised as follows. Section 2 first describes several existing diversification generators and then our new relaxation-based generator, and present a comparison of the results of these generators. Section 3 presents the other algorithm components, including the method for managing the reference set, while Section 4 explains our proposed integration of additional components to enhance the search. Section 5 reports the computational results obtained with our method on a set of classic and correlated MKP instances, and Section 6 offers our conclusions and provides some useful suggestions for future research.

2 Diversification Generators

The notion of diversity is important in heuristic search in general, and in scatter search in particular, making the design of a suitable diversification generator particularly important for striking the correct balance between diversification and intensification during the search. In other words, it is necessary to visit and retain both diverse and elite solutions during the search. The first component in scatter search is the method for generating an initial population of diverse solutions. This component plays a key role since it is also applied every time the reference set converges to restart the search. In this section, we present four diversification generators.

2.1 Random Generator

The first generator is a simple scheme that randomly chooses a subset of variables to set at value 1 for every solution generated. Though the solutions obtained can be feasible or infeasible, the advantages of this kind of generator are its simplicity and the few parameters required. The generator's main drawback is that it does not ensure a high level of quality or diversity in the population generated.

2.2 Sequential Generator

The second generator, called the "sequential generator" in this paper, was proposed by Glover [14]. This generator produces a set of solutions based on a seed solution

x, the number of problem variables n, and a parameter denoted h^*. Two solutions x' and x'' are generated at each step according to the following rules:

$$x'_1 = 1 - x_1,\ x'_{1+jh} = 1 - x_{1+jh},\ \text{ for } j = 1, 2, \ldots,\ j^* = \lfloor n/h \rfloor,$$

where $\lfloor \alpha \rfloor$ denotes the greatest integer less than α and h denotes the current iteration. The other components in x' are the same as those in x. Solution x'' is then the complement of x' [i.e., $x'' = e - x'$, where $e = (11 \ldots 1)$].

Glover [14] provided an approximation of the total number of solutions generated by the sequential generator: $h^* \times (h^* + 1)$. He also proposed adding a condition to avoid duplicate solutions. However, this condition is not sufficient to avoid the duplication, as we show in the following expressions.

Let $q = \lfloor n/2 \rfloor$, then $n = 2q$ if n is even, or $n = 2q + 1$ if n is odd. The number of distinct solutions generated by the sequential generator (denoted D_{n,h^*} when h^* and n are fixed) is then determined by the following expressions:

$$\begin{aligned}
&D_{n,1} = 1;\ D_{n,2} = 3\\
&D_{n,h^*} = h^*(h^* + 1) - 3 \ \text{ for } \ h^* = 3, \ldots, q + 1\\
&D_{n,h^*} = (q + 1)(q + 2) - 3 + (n - q)(n - q + 1) - (n - h^* + 1)(n - h^* + 2)\\
&\qquad\qquad \text{for } \ h^* = q + 2, \ldots, n
\end{aligned}$$

The number of duplicate solutions (denoted R_{n,h^*}) is determined by the following expressions:

$$\begin{aligned}
&R_{n,h^*} = 0 \ \text{ for } h^* = 1, 2, \ldots, q + 1\\
&R_{n,h^*} = 2 \times (h^* - q)(h^* - q - 1) \qquad \text{if } n \text{ is even and } h^* = q + 2, \ldots, n\\
&R_{n,h^*} = 2 \times ((h^* - q)(h^* - q - 2) + 1) \ \text{ if } n \text{ is odd and } h^* = q + 2, \ldots, n
\end{aligned}$$

It is also possible to modify the sequential generator to generate only distinct solutions. This is a consequence of the experimental observation that duplicate solutions are generated in groups when running the algorithm.

As the previous expressions show, the number of solutions generated increases very quickly with n and h^*. When $n \leq 25$, a large value of h^* is necessary to generate more solutions. On the contrary, a small value of h^* (less than $n/2$) seems to be enough to generate more solutions when n increases. This combination also ensures that duplicate solutions will not be generated. Although the sequential generator is a fast algorithm, it does not ensure the quality of the population generated.

2.3 Dichotomous Generator

Glover [14] also proposed another generator, called the "dichotomous generator". This generator partitions the variable set N into two subsets. Two solutions are then produced by taking the complement of a seed solution over both subsets. This process is repeated—partitioning of each of the subsets into another two subsets, generating two solutions, and so on—until all the subsets contain only one variable. This generator constructs approximately $2 \times (1 + \log n)$ solutions. Like the sequential generator, the dichotomous generator produces an initial population rapidly, but although it

better controls the diversity of the generated population, it only produces a small number of solutions and has no control over the solution quality.

2.4 Relaxation-based Generator

Our proposed generator, called the "relaxation-based generator" in this paper, generates an initial population that produces interesting solutions in terms of objective value. As we show in the computational experiments, it enables the scatter search algorithm to visit elite solutions more quickly. To describe this generator, we first need to introduce the notion of reduced problems. Such problems are defined from a feasible problem solution y and a variables subset J, and can be expressed as:

$$(\text{MKP}(y,J))\ \max\left\{p^T x : Ax \leq c; x_j = y_j\ \forall j \in J, x \in \{0,1\}^n\right\} \tag{1}$$

Our relaxation-based generator (see Algorithm 1) seeks to determine a feasible solution for every problem variable (loop *for* in line 2). To do this, it associates the reduced problem MKP($e - y, \{j\}$) with the variable x_j. A feasible solution of the MKP is generated from the problem MKP($e - y, \{j\}$) by applying the following procedure (lines 3–5).

- Step1: solve the LP-relaxation of the problem MKP($e - y, \{j\}$) and retain an optimal solution $\bar{x}$ (note that it is possible to use another relaxation).
- Step2: construct the reduced problem obtained from the MKP and from $\bar{x}$ by fixing all the variables with the value 0 or 1 in $\bar{x}$.
- Step3: solve exactly the reduced problem obtained in Step2.

An optimal solution of the reduced problem is then added in the population P and the process is repeated. To define solution y in (1), the above procedure is applied to the original problem (when $j = 0$ in line 2) in order to obtain an initial feasible solution for the MKP. This solution is then used to construct the reduced problem associated with every problem variable (when $j > 0$ in line 2).

Algorithm 1 The relaxation-based generator

Require: The MKP instance
Ensure: a solution set P
1: $y = 0$; $J = \oslash$; $P = \oslash$;
2: **for** $j = 0$ to n **do**
3: Let $\bar{x}$ be an optimal solution of the LP-relaxation of MKP($e - y, J$);
4: $I = \{j \in N : \bar{x}_j \in \{0,1\}\}$;
5: Let x be an optimal solution of MKP($\bar{x}, I$);
6: **if** $x \notin P$ **then**
7: $P = P \cup \{x\}$;
8: **end if**
9: **if** $j = 0$ **then**
10: $y = x$;
11: **end if**
12: $J = \{j + 1\}$;
13: **end for**
14: return P

Obviously, since this generator requires the solving of several reduced problems, its temporal complexity will increase compared to the previous generators. However, with the linear programming relaxation, the reduced problems have at most m variables, which is generally small for the MKP compared to n. To give an idea of the practical complexity of the reduced problems solved in Algorithm 1, on average, the first reduced problem generated from $\bar{x}$ can be solved exactly in less than 3 seconds for the largest instances described in Section 5 (with $n = 500$ and $m = 30$) with a branch-and-bound method. It is however worth noting that these instances would require several hours to be solved exactly using a branch-and-bound method [6].

The Algorithm 1 can be repeated while the best generated solution is improved since the reduced problems in lines 3–4 depend on the initial solution. It is also possible to apply the following classic reduction property.

Proposition 1 *For every $j \in N$ and every feasible solution y of the MKP, if $v(MKP(e - y, \{j\})) \leq p^T y$, then either $x_j = y_j$ in any optimal solution of the MKP, or y is an optimal solution of the MKP, where $v(P)$ is the optimal value of problem P.*

Proof It is trivial to note that the reduced problem MKP$(e - y, \{j\})$ corresponds to the initial problem in which only variable x_j is set to the value $1 - y_j$. If the optimal value of this problem is less than the value of y, it is not possible to obtain a better value than $c^T y$ by setting $x_j = y_j$. □

Proposition 1 remains valid if we replace $v(\text{MKP}(e - y, \{j\}))$ by an upper bound of the problem MKP$(e - y, \{j\})$. In our experiments we used the linear programming relaxation of these problems.

2.5 Evaluation

Preliminary results showed us that the relaxation-based generator produces elite populations in terms of quality. Thus this generator is used to generate an initial population of the algorithm.

When applying the scatter search algorithm, it happens that the reference set converges (when there is no new solution to add). At this moment it is necessary to apply a diversification generator to restart the search. In this section we compare the generators described above (i.e. the random, the sequential and the dichotomous generator) according to their performance. The aim is to apply the most efficient generator among the 3 candidates. We evaluate the performance of a generator according to two criteria : the quality and the diversity of the population generated.

We have first to explain how we evaluate the diversity of a population. To measure the diversity between two solutions x and y, we use the Hamming distance $d(x, y)$:

$$d(x, y) = \sum_{j \in N} |x_j - y_j| .$$

Then the value

$$d(x, P) = Min\{d(x, y) : y \in P\}$$

is used to measure the distance between solution x and population P.

Table 1 Evaluation of the diversification generators

	δd	δp	$\delta p + \delta d$
Random	0.37	0.49	0.86
Seq.	0.05	0.63	0.68
Dicho.	0.90	0.37	1.27

Let $\Omega = \{P_1, P_2, \ldots, P_k\}$ be a set of populations. To evaluate the diversity of population P_i among Ω, we calculate the following value $\delta d(P_i)$:

$$\delta d(P_i) = \frac{d(P_i) - d_{\min}}{d_{\max} - d_{\min}},$$

where $d(P_i) = \sum_{x \in P_i} d(x, P_i)$, $d_{\min} = min_{j=1,\ldots,k}\{d(P_j)\}$, and $d_{\max} = \max_{j=1,\ldots,k}\{d(P_j)\}$.

In the same way we can evaluate the quality of population P_i among Ω by calculating the following value $\delta p(P_i)$:

$$\delta p(P_i) = \frac{p(P_i) - p_{\min}}{p_{\max} - p_{\min}},$$

where $p(P_i) = \sum_{x \in P_i} p^T x$, $p_{\min} = \min_{j=1,\ldots,k}\{p(P_j)\}$, and $p_{\max} = \max_{j=1,\ldots,k}\{p(P_j)\}$.

Campos et al. [3] used a similar method to evaluate the performance of a generator for the linear ordering problem.

We evaluated the generators on a set of 55 small MKP instances [1], with several seed solutions for the sequential and the dichotomous generators, and several runs for the random generator. Then we calculated the previous values δd, δp and the value $\delta p + \delta d$ obtained for each generator on the 55 instances. We give in Table 1 the average values obtained respectively by the random generator, the sequential generator and the dichotomous generator.

Table 1 shows that the dichotomous generator obtained, on average, the best population in terms of diversity, which can be explained by the fact that every generated solution maximizes the minimum Hamming distance between all the other generated solutions in the population (see Glover [14]). If both criteria are considered, then this generator obtains much better results than the sequential and random generators. For these reasons, we decided to use the dichotomous generator to restart the search when the reference set converges.

3 Other Components of the Scatter Search Algorithm

3.1 Improving the Solutions

After the diversity generation method, the next component is the improvement method. As the improvement phase is called regularly while the scatter search algorithm is running, we chose to use a simple local search method based on the heuristic k-opt. In this method, k objects are retrieved from the solution and then, if possible, one or more other objects are added. This k-opt heuristic, with $k = 1$, was applied to the feasible solutions during our experimentation. When a solution was infeasible, a simple greedy heuristic was applied to transform it into a feasible

solution, or at least into a solution near the feasible domain, given the use of infeasible solutions explained below. The greedy heuristic is based on the *efficiency* E_j of variable x_j, which is defined as:

$$E_j = \frac{p_j}{\sum_{i \in M} \mu_i a_{ij}} \tag{2}$$

where μ is a multiplier of dimension m. In practice, μ_i is the shadow price of the ith constraint in the linear programming relaxation of the problem. A list of E_j values is maintained in increasing order. In the greedy algorithm, variables are eliminated starting from the top of the list. The same principle is applied in the local search algorithm, but the items are inserted into the current solution starting from the bottom of the list.

3.2 Updating the Reference Set

The third component is the reference set update method, which creates and maintains the reference set during the search. We propose using three types of solutions: (1) elite solutions, chosen according to their objective value only; (2) diverse solutions, chosen according to their diversity; and (3) infeasible solutions, chosen according to their distance from the feasible domain. Infeasible solutions are introduced in the reference set because visiting this kind of solutions has proved to be useful for solving the MKP, in particular algorithms (see for example, Glover and Kochenberger [15], Hanafi and Fréville [18], Li and Curry [22]). The results presented in Section 5 show that using infeasible solutions can also be effective in a scatter search context.

The diversity of the reference set is based on the Hamming distance (see Section 2.5). Then, the diversity $d(P)$ of the population P is defined as:

$$d(P) = Max\{Min\{d(x, y) : y \in P\} : x \in P\}$$

This max–min measure has already been used with success by Marti et al. [24]. A solution is inserted into P if it increases the diversity of P, or in other words, if the value $d(P)$ is increased or if the minimum distance between the candidate and the other solutions in the reference set is larger than the current minimum distance between two solutions in the reference set.

3.3 Generating the Subsets

The fourth SC component is the subset generation method. The solution combination methods in scatter search are not limited to combining just two solutions and therefore the subset generation method in its more general form consists of creating subsets of different sizes. As proposed by Glover [14], 4 types of solution subsets are used in our method, each containing 2, 3, 4 and k solutions respectively ($k = 5$ up to the size of the reference set).

Since the total number of subsets associated with the reference set increases very quickly with the number of solutions in the reference set, it is typically recommended to only create a part of the subsets as follows: all the subsets of type 1

(with 2 elements) are generated. Then the subsets of type 2 are derived from the previous subsets by augmenting each 2-element subset to include the best solution not in this subset. The same principle is applied for generating the subsets of type 3. Finally subsets of type 4 consist of the best k elements, for $k = 5$ to the size of the reference set.

The fact that the combination methods (see the next section) used are not based on random ensures that the combined solution(s) obtained for a given subset does not change if the combination method is applied more than once. Therefore, once a given subset is created, there is no merit in creating it again. It is why we chose a dynamic method to avoid this situation. The dynamic management is realized using two vectors. The first maintains the last iteration in which a position in the reference set was changed ($Iter$), and the second maintains the last iteration in which a subset type was generated ($Subset$). It is easy to determine whether or not a solution associated with an index j in the reference set ($1 \leq j \leq$ number of solutions in the reference set) has to be considered for a given type of subset i ($1 \leq i \leq 4$) by checking whether $Subset[i] \leq Iter[j]$. If this condition is satisfied, then the solution stored in location j has been modified since the last time the subsets of type i were generated.

3.4 Combining the Solutions

The fifth component of the scatter search algorithm is the solution combination method. Our algorithm uses two combination methods depending on the size of the solution subsets. The first combination method is based on a score in the range [0; 1] associated with every variable of the problem for a given solution subset S. When the score of variable x_j is greater than 0.5, then $x_j = 1$ in the new solution. Laguna and Marti [21] have defined a score s_j of variable x_j for the knapsack problem as follows:

$$s_j = \frac{\sum_{x \in S} p^T x \times x_j}{\sum_{x \in S} p^T x} \tag{3}$$

We propose a new score s_j obtained from formula (3) by integrating the objective coefficient p_j to accentuate the impact of variable x_j on the solutions in the subset S as follows:

$$s_j = \frac{\sum_{x \in S} (p^T x \times x_j + (1 - 2x_j) p_j)}{\sum_{x \in S} p^T x}$$

It is obvious that the score-based combination can produce one of the initial solutions, particularly for type 1 subsets with only two solutions. We thus decided to apply this method to type 2, 3 and 4 subsets.

For type 1 subsets, we implemented an algorithm based on star paths [13]. Glover et al. [16] showed that the combination of star paths with scatter search allows generating a set of diverse solutions for the 0–1 mixed integer programming problems. A star path is defined as a sequence of integer solutions between an initial solution x_1 and a final solution x_2.

A set of solutions is generated on the path according to a sequence of directional roundings. A directional rounding is defined relative to a base point (not necessarily

an integer), x^0, and an initial point (x_1 in our case). The directional rounding associated with x^0 and x^1 generates the solution y such that, for all $j \in N$:

$$y_j = \begin{cases} 1 & \text{if } x_j^1 > x_j^0 \\ 0 & \text{if } x_j^1 < x_j^0 \\ x_j^0 & \text{if } x_j^1 = x_j^0 \text{ and } x_j^0 \in \{0, 1\} \\ \mathrm{r}(x_j^0) & \text{otherwise} \end{cases}$$

where $\mathrm{r}(x_j^0)$ refers to simple rounding.

A star path is defined as a set of vectors associated with a line between x_1 and the final solution x_2 obtained from a sequence of directional roundings using x^0 as a base point (for more details, the reader is referred to Glover et al. [16]).

In our experiments, we used an optimal solution of the linear programming relaxation as base point and the two solutions of the subset as initial solution and final solution. We implemented a simple version of the star-path method because, given that type 1 subsets are more numerous than other subsets, we needed a quick algorithm to avoid slowing down the SC algorithm. In practice, we retained a small number of solutions (the best ones) generated along the star path.

4 Additional Components to Enhance Scatter Search

A set of preliminary experiments was conducted on the set of small instances mentioned in Section 2.5 in order to try to adjust some of the algorithm parameters. We compared the algorithms' performance in terms of quality and diversity. Our results are summarized below, but more details are available in Wilbaut [29].

We began by varying the types of solutions in the reference set. Our results clearly demonstrated that retaining only elite or diverse solutions makes the quality of the results deteriorate, whereas using both elite and diverse solutions improves the quality of the results. This outcome confirms the lesson #6 in Laguna and Armentano [20], indicating that it is necessary balance the number of elite and diverse solutions in the reference set. In addition, it appears that integrating infeasible solutions into the reference set produces even better results, which validates the potential of using the information contained in infeasible MKP solutions. However, the simple version of scatter search was not able to solve all the instances considered in these experiments. To allow all the instances to be solved, we propose to strengthen the use of memory and the use of intensification phases into the algorithm.

4.1 Using Memory

Search memory is often used in tabu search [11, 12]. We propose using frequency memory to modify the order of the variables during the improvement phases, which would prevent multiple visits to the same local optimum when the local search-based heuristics are applied. This is achieved by sorting the variables according to their efficiency and memory information. The aim is to take into account the history of the search to guide the improvement phases.

The value f_j, corresponding to the number of times the variable x_j is fixed to value 0 or 1 in the reference set for all the variables, is maintained, and thus the vector f of dimension n sums all the solutions recorded in the reference set. Then the value FR_j, associated with variable x_j, is computed according to its frequency and efficiency:

$$FR_j = \alpha R_j + (1 - \alpha) F_j \tag{4}$$

where $R_j = \frac{E_j}{\sum_{j \in N} E_j}$, $F_j = \frac{f_j}{\sum_{j \in N} f_j}$ and $\alpha \in [0; 1]$.

Initially, the value of α is set to 1 because there is no information from the frequency. The value of α then decreases during the search (each time the reference set converges) to give more importance to the search memory ($\alpha = \alpha - 0.1$, up to $\alpha = 0.2$). The values FR_j are used during the improvement phases, with formula (4) replacing formula (2) when sorting the variables.

4.2 Defining Other Intensification Phases

We also propose integrating adapted intensification phases into the algorithm so that the information associated with the reference set can be used before restarting the search. Two procedures are applied to generate elite solutions from the search history before generating a new reference set when this set converges.

The first procedure, which is described in Algorithm 2, solves exactly a small size problem generated from the frequency memory described above, starting from an initial null solution ($y = 0$). In the first step, an attempt is made to set n^1 variables to value 1 while still respecting the knapsack constraints [lines 2–4; variables are considered in the decreasing order of (4)]. Next, a subset of variables is released according to a parameter (n^* in Algorithm 2, though we used the value 10 in our experiments). The most undecided variables when the variables are sorted according to (4) are chosen to be released. In lines 10–11, the reduced problem is finally solved.

It is possible to refine the number of variables to be set to value 1 (n^1 in Algorithm 2) by computing bounds on the sum of the variables in an optimal solution of the MKP. A lower (respectively an upper) bound on this sum can be obtained by solving the following linear program [LS; resp. (US)]:

$$\text{(LS)} \begin{cases} \underline{n} = \min \sum_{j=1}^{n} x_j \\ \text{s.t.: } Ax \le c \\ \quad 1 + px^* \le px \\ \quad x_j \in [0, 1], \, j \in N \end{cases} \qquad \text{(US)} \begin{cases} \overline{n} = \max \sum_{j=1}^{n} x_j \\ \text{s.t.: } Ax \le c \\ \quad 1 + px^* \le px \\ \quad x_j \in [0, 1], \, j \in N \end{cases}$$

where x^* is the best feasible solution (see, for example, Fréville and Plateau [7] or Vasquez and Hao [27]).

When we apply Algorithm 2, the value of parameter n^1 is chosen in the range $[\underline{n}; \overline{n}]$.

We also introduce another intensification process that applies a path relinking between the reference set solutions. Path relinking is a metaheuristic that is related to scatter search [14]. Since we did not want to slow the algorithm down, we imple-

Algorithm 2 Intensification by solving a reduced problem

Require: The MKP instance, an expected number of variables set at '1' n^1, the size of the reduced problem n^*
Ensure: a feasible solution y
1: $y = 0$;
2: **for** $j = 1$ to n **do**
3: **if** $Ay + A^j \leq c$ **then**
4: $y_j = 1$ (where A^j corresponds to $(a_{1j}; a_{2j}; \ldots; a_{mj})$;
5: **end if**
6: **end for**
7: **if** $ey < n^1$ **then**
8: $n^* = n^* + n^1 - ey$;
9: **end if**
10: Let z be an optimal solution of the reduced problem MKP(y, $N - \tilde{N}$)
11: where $\tilde{N} = \{\lfloor n/2 \rfloor - \lfloor n^*/2 \rfloor; \ldots; \lfloor n/2 \rfloor + \lfloor n^*/2 \rfloor\}$, when the variables are sorted according to (4)
12: return z

mented a simple version, as shown in Algorithm 3. To create the path (lines 2–3), Algorithm 3 modifies every distinct variable between the initial solution x and the final solution y one after the other, and then it activates a local search around the solutions generated on that path (line 5, where LS(z, F) corresponds to a call to the local search algorithm on solution z, but only for subset F).

The results of Algorithm 3 depend partially on the order of the variables in line 2 and partially on the local search algorithm used in line 5. In our implementation, we considered the variables set to value 1 (resp. 0) in the decreasing (resp. increasing) order of (4). The local search algorithm is based on the same principle as the one used in the improvement phase (see Section 3.1). Algorithm 3 also maintains all the solutions generated between x and y in a pool P. In practice, the size of P is limited to 10 elements (i.e., the best solutions), and the local search algorithm does not need to be applied to every solution, but only when the best solution on the path is improved.

Algorithm 3 Intensification with path relinking

Require: the initial solution x and the final solution y
Ensure: the set of generated solutions P
1: $z = x$; $P = \oslash$; $F = \oslash$;
2: **for** $j \in N : x_j \neq y_j$ **do**
3: $z_j = 1 - z_j$;
4: $F = F \cup \{j\}$;
5: $z^* = \text{LS}(z, F)$;
6: add z^* to P;
7: **end for**
8: return P;

The two intensification methods described above are applied each time the reference set converges. Then, when we integrated the use of memory and these processes into the scatter search algorithm, we were able to solve exactly all the small MKP instances used during the preliminary experiments.

Before reporting our computational results in the next section, we would like to review the components used in all our scatter search implementations:

- The dichotomous generator is used to construct a new reference set when it converges (Section 2.3).
- The reference set is composed of 40 (resp. 20) solutions when $n = 100$ (resp. $n > 100$).
- A dynamic management is used to generate the solution subsets (Section 3.3).
- A score-based combination method is used to combine subsets with more than two solutions, and a star path combination method is used for type 1 subsets (Section 3.4).

5 Computational Results

All the results presented in this paper were obtained with a Pentium IV 3.4GHz, using algorithms coded in C++ language. The algorithms were validated on a set of correlated instances available in the OR-Library [1], composed of 270 MKP instances generated according to the procedure proposed by Fréville and Plateau [8]. The instances were generated by varying combinations of constraints ($m = 5, 10, 30$) and variables ($n = 100, 250, 500$), with 30 instances generated for every n-m combination.

We first compare in Table 2 the results obtained by the relaxation-based generator used alone (rows "GP") and the scatter search algorithm used alone (rows "SC") with other optimisation methods proposed for solving the MKP. Here the initial population of the scatter search algorithm is obtained by applying the dichotomous generator. In Table 2, rows "GAP" correspond to the average percent deviation from the optimum values of the LP-relaxations (since the optimum solution values are not known for all the instances). Row "GAP_I" reports the results when elite, diverse and infeasible solutions were used in the scatter search implementation (whereas row "GAP" refers to the results when only elite and diverse solutions are used). Rows "CPU" report the average computational times in seconds of the algorithms. Rows "GA_{CB}" and "GAP_{HV}" present the average difference from the LP-value for genetic algorithms proposed by Chu and Beasley [4] and Haul and Voß [19], respectively. We consider these algorithms useful for comparison purposes since they

Table 2 Results of the new generator and scatter search on the OR-Library

Size of	n	100	250	500	100	250	500	100	250	500
the Pb.	m	5			10			30		
GP	GAP	0.66	0.18	0.06	1.10	0.36	0.15	1.73	0.68	0.34
	CPU	1	4	60	1	6	81	15	86	430
SC	GAP_I	0.59	0.19	0.08	0.96	0.38	0.21	1.71	0.82	0.55
	GAP	0.59	0.18	0.08	0.98	0.39	0.21	1.71	0.83	0.56
	CPU	21	54	422	26	69	341	39	87	342
	GA_{CB}	0.58	0.14	0.05	0.94	0.30	0.13	1.69	0.67	0.35
	GA_{HV}	0.72	0.36	0.34	1.26	0.74	0.64	2.14	1.36	1.20
	ADP	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.97	0.52
	Ra-Ps	0.60	0.17	0.09	1.17	0.45	0.20	2.23	1.38	0.82

are also population-based methods and have obtained valuable results for the MKP. Row ADP gives the results obtained with the approximate dynamic programming method proposed by Bertsimas and Demir [2], and Ra-Ps reports the results obtained with the metaheuristic, randomized priority search, recently proposed by Moraga et al. [25]. Each value in Table 2 is an average over 30 instances.

The primary conclusions that can be drawn from Table 2 are as follows. First, the new generator obtains good quality solutions, particularly for the larger instances. It dominates all the other methods reported in Table 2 excepted the Chu and Beasley's genetic algorithm and the RA-Ps when $m = 5$ and $n < 500$. In addition, the computational effort associated with this generator is not excessive, even for the largest instances ($m = 30$). Though it is difficult to compare the execution times of several algorithms since they were not all executed on the same computer, it is nonetheless noteworthy that the RA-PS algorithm's run times ranged from 7 to 35 min per instance on a Pentium IV 1.6Ghz compared to run times ranged from 1 to 7 min for our generator on a Pentium IV 3.4GHz. Thus, it appears that our generator could be very useful for generating an initial elite population for the scatter search algorithm, as well as for solving the MKP if used only as a heuristic.

Second, the scatter search algorithm obtained encouraging results when applied alone. It dominates the Haul and Voß's genetic algorithm and obtains better solutions than the RA-Ps algorithm for a large part of the instances. It also obtains better solutions than the Bertsimas and Demir's approach for $n = 250$ and $m = 30$, but it is not the case for the largest instances. However, the Chu and Beasley's algorithm clearly dominates this implementation of scatter search. Table 2 also shows that when infeasible solutions are used in the reference set, on average, better results are obtained. More precisely, the version with infeasible solutions obtains 127 best solutions, whereas the version without infeasible solutions finds 65 best solutions only (the other 78 solutions are the same).

Table 3 presents the final results obtained with the scatter search algorithm. In this version, scatter search is applied from with the initial population generated

Table 3 Final results of scatter search on the OR-Library

Size of	n	100	250	500	100	250	500	100	250	500
the Pb.	m	5			10			30		
GP	GAP	0.66	0.18	0.06	1.10	0.36	0.15	1.73	0.68	0.34
	CPU	1	4	60	1	6	81	15	86	430
FSC	GAP	0.58	0.15	0.05	0.94	0.31	0.13	1.68	0.66	0.33
	CPU	25	52	205	45	195	872	81	308	1338
	GA_{CB}	0.58	0.14	0.05	0.94	0.30	0.13	1.69	0.67	0.35
	HTS	N/A	N/A	0.04	N/A	N/A	0.10	N/A	N/A	0.28
	Cplex	0.58	0.14	0.05	0.94	0.29	0.12	1.70	0.64	0.33

by the new relaxation-based generator. It is why we recall the results obtained by this generator (rows GP). Then we give the results of the final scatter search (rows "FSC") in which we added the following components:

– Use memory information to reorder the variables when the reference set converges (Section 4.1).
– Apply the adapted intensification phases each time the reference set converges to take into account the memory information contained in the reference set (Section 4.2).

The number of iterations of the scatter search algorithm was set according to the size of the instances and is included in [2*n;5*n]. As in Table 2, all the CPU times are in seconds, and the quality of the solutions is compared to the LP-value of the instance. In this table we also introduce in row "VV" the results obtained by the hybrid tabu search-based algorithm developed by Vasquez and Vimont [28], and we report in row "Cplex" the results obtained with the software CPLEX of Ilog. We allowed CPLEX the same computational time to solve the instance that we allowed our own algorithm.

Rows FSC show the positive impact of our specific components on the performance of the scatter search algorithm. If we compare the rows GAP of GP and FSC, we can conclude that our modifications allow the algorithm to improve the initial population given by our relaxation-based generator, and thus obtain good final solutions. If the results of the final solutions are compared in terms of quality with the results of other algorithms, the Vasquez and Vimont's algorithm is better for the largest instances with $n = 500$. However, the computational effort needed to produce these solutions (several hours on a Pentium IV 2GHz) is far in excess of the computational effort needed by our algorithm. The results mentioned by Chu and Beasley and those obtained by CPLEX seem to be on the same order of magnitude. To give an idea of the performance of our algorithm when compared to the Chu and Beasley's algorithm (resp. CPLEX), our algorithm obtains the same objective function value for 128 (resp. 120) instances and better ones for 67 (resp. 29) instances. In other words, 195 (resp. 149) solutions have at least the same objective value over the 270 instances. The average difference between our method and the Chu and Beasley's one (resp. CPLEX) is about −0.01 (resp. 0.01).

With respect to computational time, as shown in Table 3, our algorithm does not require an excessive effort since, on average, it does not exceed 25 min for the largest instances (note that row CPU of FSC reports the total computational effort for the algorithm, including the effort for the relaxation-based generator). The CPU times for the scatter search algorithm also depend on the size of the problem to which it is applied. By applying the reduction procedure (Lemma 1 in Section 2), the variables can be set definitively at their optimal values, particularly for $m = 5$.

These results show that scatter search can be applied efficiently for solving the MKP and can even rival other methods. In particular, it seems that both introducing more memory and producing an initial elite population enhance the results of scatter search.

6 Conclusions

In this paper, we proposed a scatter search algorithm for solving the 0–1 multidimensional knapsack problem. To our knowledge, this is the first time that the scatter search approach has been applied to this problem. Our algorithm is constructed along the same general lines proposed by Glover [14]. We first proposed a new relaxation-based generator that solves a series of small problems in order to obtain an elite population. According to our computational results, this generator clearly guides the scatter search algorithm to visit elite solutions more quickly. Then we proposed to enhance the scatter search algorithm by using infeasible solutions in the reference set, by using search memory information and by integrating adapted intensification phases. The results obtained on a classic set of correlated instances show that our algorithm outperforms other population-based methods for solving the multidimensional knapsack problem. The final compromise between the quality of the solutions and the computational effort seems to be worthwhile.

One direction for future research is to investigate the possibilities of integrating more search memory, particularly during the diversification phases. This can be done either by modifying the existing generators or by proposing new ones. It would also be possible to integrate memory into the combination methods. Another direction for research may be to explore the influence of the number of elite, diverse and infeasible solutions in the reference set. Modifying the proportion of the different types of solutions may have an impact on the search. It would also be worthwhile to propose an adaptive component, which regularizes the number of each type of solutions according to the evolution of the search.

Acknowledgements We would like to thank the referees for their constructive comments that improved both the content as well as the presentation of the paper.

References

1. Beasley, J.E.: OR-Library: distributing test problems by electronic mail. J. Oper. Res. Soc. **41**(11), 1069–1072 (1990)
2. Bertsimas, D., Demir, R.: An approximate dynamic programming approach to multidimensional knapsack problems. Manage. Sci. **48**(4), 550–565 (2002)
3. Campos, V., Glover, F., Laguna, M., Martí, R.: An experimental evaluation of a scatter search for the linear ordering problem. J. Glob. Optim. **21**, 397–414 (2001)
4. Chu, P., Beasley, J.E.: A genetic algorithm for the multidimensional knapsack problem. J. Heuristics **4**, 63–86 (1998)
5. Fréville, A.: The multidimensional 0–1 knapsack problem: an overview. Eur. J. Oper. Res. **155**(1), 1–21 (2004)
6. Fréville, A., Hanafi, S.: The multidimensional 0–1 knapsack problem—bounds and computational aspects. Ann. Oper. Res. **139**(1), 195–227 (2005)
7. Fréville, A., Plateau, G.: Sac-à-dos multidimensionel en variables 0–1: Encadrement de la somme des variables à l'optimum. RAIRO Oper. Res. **27**, 169–187 (1993)
8. Fréville, A., Plateau, G.: An efficient preprocessing procedure for the multidimensional knapsack problem. Discrete Appl. Math. **49**, 189–212 (1994)
9. Gilmore, P.C., Gomory, R.E.: The theory and computation of knapsack functions. Oper. Res. **14**, 1045–1075 (1966)
10. Glover, F.: Heuristics for integer programming using surrogate constraints. Decis. Sci. **8**, 156–166 (1977)
11. Glover, F.: Tabu search—part I. ORSA J. Comput. **1**(3), 190–206 (1989)
12. Glover, F.: Tabu search—part II. ORSA J. Comput. **2**(1), 4–32 (1990)

13. Glover, F.: Scatter search and star paths: beyond the genetic metaphor. OR Spektrum **17**, 125–137 (1995)
14. Glover, F.: A template for scatter search and path relinking. In: Hao, J., Lutton, E., Ronald, E., Schoenauer, M., Snyers, D. (eds.) Artificial Evolution, vol. 1363 of Lecture Notes in Computer Science, pp. 13–54. Springer (1998)
15. Glover, F., Kochenberger, G.: Critical event tabu search for multidimensional knapsack problems. In: Osman, I., Kelly, J. (eds.) Meta Heuristics: Theory and Applications, pp. 407–427. Kluwer Academic Publishers (1996)
16. Glover, F., Lokketangen, A., Woodruff, D.: Scatter search to generate diverse MIP solutions. In: Laguna, M., Gonzlez-Velarde, J. (eds.) OR Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research, pp. 299–317. Kluwer Academic Publishers (2000)
17. Gomes da Silva, C., Climaco, J., Figueira, J.: A scatter search method for the bi-criteria multi-dimensional {0,1}-knapsack problem using surrogate relaxation. J. Math. Model. Algorithms **3**(3), 183–208 (2004)
18. Hanafi, S., Fréville, A.: An efficient tabu search approach for the 0–1 multidimensional knapsack problem. Eur. J. Oper. Res. **106**, 659–675 (1998)
19. Haul, C., Voß, S.: Using surrogate constraints in genetic algorithms for solving multidimensional knapsack problems. In: Woodruff, D. (ed.) Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search: Interface in Computer Science And Operations Research, pp. 235–251. Kluwer Academic Publishers (1998)
20. Laguna, M., Armentano, V.: Lessons from applying and experimenting with scatter search. In: Rego, C., Alidaee, B. (eds.) Metaheuristic Optimization via Adaptive Memory and Evolution: Tabu Search and Scatter Search, pp. 229–246. Kluwer Academic Publishers (2005)
21. Laguna, M., Marti, R.: Scatter Search: Methodology and Implementations in C. Kluwer Academic Publishers (2003)
22. Li, V., Curry, G.: Solving multidimensional knapsack problems with generalized upper bound constraints using critical event tabu search. Comput. Oper. Res. **32**(4), 825–848 (2005)
23. Lorie, J.H., Savage, L.J.: Three problems in capital rationing. J. Bus. **28**, 229–239 (1955)
24. Marti, R., Lourenco, H., Laguna, M.: Assigning proctors to exams with scatter search. In: Laguna, M., Gonzlez-Velarde, J. (eds.) Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research, pp. 215–227. Kluwer Academic Publishers (2000)
25. Moraga, R.J., DePuy, G.W., Whitehouse, G.E.: Meta-RaPS approach for the 0–1 multidimensional knapsack problem. Comput. Ind. Eng. **48**, 83–96 (2005)
26. Vasquez, M., Hao, J.K.: A logic-constrained knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. Comput. Optim. Appl. **20**, 137–157 (2001a)
27. Vasquez, M., Hao, J.K.: Une approche hybride pour le sac à dos multidimensionnel en variables 0–1. RAIRO Oper. Res. **35**, 415–438 (2001b)
28. Vasquez, M., Vimont, Y.: Improved results on the 0–1 multidimensional knapsack problem. Eur. J. Oper. Res. **165**, 70–81 (2005)
29. Wilbaut, C.: Heuristiques hybrides pour la résolution de problèmes en nombres entiers mixtes. Ph.D. thesis, Université de Valenciennes et du Hainaut Cambrésis (2006)