A shuffled complex evolution algorithm for the multidimensional knapsack problem using core concept

Marcos Daniel Valadão Baroni
Departamento de Informática
Universidade Federal do Espírito Santo
Vitória, Espírito Santo, Brazil
Email: mbaroni@ninfa.inf.ufes.br

Flávio Miguel Varejão
Departamento de Informática
Universidade Federal do Espírito Santo
Vitória, Espírito Santo, Brazil
Email: fvarejao@ninfa.inf.ufes.br

***Abstract*—This work addresses the application of a population based evolutionary algorithm called shuffled complex evolution (SCE) in the core of multidimensional knapsack problem (MKP). The core of the MKP is a set of items which are hard to decide if they are or not selected in good solutions. This concept is used to reduce the original size of MKP instances. The performance of the SCE applied to the reduced MKP is verified through computational experiments using well-known instances from literature and randomly generated problems as well. The approach proved to be very effective in finding near optimal solutions demanding a very small amount of processing time.**

I. Introduction

The Multidimensional Knapsack Problem (MKP) is a strongly NP-hard combinatorial optimization problem which can be viewed as a resource allocation problem and defined as follows:

$$\begin{aligned} &\text{maximize} \sum_{j=1}^{n} p_j x_j \\ &\text{subject to} \sum_{j=1}^{n} w_{ij} x_j \leqslant c_i \quad i \in \{1,\dots,m\} \\ &\qquad x_j \in \{0,1\}, \quad j \in \{1,\dots,n\}. \end{aligned}$$

The problem can be interpreted as a set of n items with profits p_j and a set of m resources with capacities c_i. Each item j consumes an amount w_{ij} from each resource i, if selected. The objective is to select a subset of items with maximum total profit, not exceeding the defined resource capacities. The decision variable x_j indicates if j-th item is selected.

The multidimensional knapsack problem can be applied on budget planning scenarios, subset project selections, cutting stock problems, task scheduling, allocation of processors and databases in distributed computer programs. The problem is a generalization of the well-known knapsack problem (KP) in which $m = 1$.

The MKP is a NP-Hard problem significantly harder to solve in practice than the KP. Despite the existence of a fully polynomial approximation scheme (FPAS) for the KP, finding a FPAS for the MKP is NP-hard for $m \geqslant 2$ [1]. Due its simple definition but challenging difficulty the MKP is often used to to verify the efficiency of novel metaheuristics.

In this paper we address the application of a metaheuristic called shuffled complex evolution (SCE) to the multidimensional knapsack problem. The SCE is a metaheuristic, proposed by Duan in [2], which combines the ideas of a controlled random search with the concepts of competitive evolution and shuffling. The SCE algorithm has been successfully used to solve several problems like flow shop scheduling [3] and project management [4].

The reminder of the paper is organized as follows: Section III presents the shuffled complex evolution algorithm and proposes its application on the multidimensional knapsack problem. Section V comprises several computational experiments. In section VI we make our concluding remarks about the experimental results.

II. The core concept

The core concept was first presented for the one-dimensional 0-1 knapsack problem (KP), leading to very successful KP algorithms. The main idea is to reduce the original problem by only considering a set of items for which it is hard to decide if they will occur or not in an optimal solution. This set of items is named core. The variables for all items outside the core are fixed to certain values.

The KP considers items $j = 1,\dots,n$, associated profits p_j and associated weights w_j. A subset of these items has to be selected and packed into a knapsack having capacity c. The total profit of the items in the knapsack has to be maximized, while the total weight is not allowed to exceed c.

If the items are sorted according to descreasing efficiency values

$$e_j = \frac{p_j}{w_j},$$

it is well known that the solution of the LP-relaxation consists in general of three consecutive parts: The fisrt part contains variables set to 1, the second pare consists of at

most on split item s, whose corresponding LP-values is factional, and finally the remaining variables, which are always set to zero, form the third part. For most instance of KP (except those with a very special structure) the integer optimial solution closely corresponds to this partitioning in the sense that it contains most of the highly efficient items of the first part, some items with medium efficiencies near the split item, and almost no items with low efficiencies from the third part. Items of medium efficiency constitute the so called core.

Balas and Zemel [5] gave the following precise definition of the core of a KP, based on the knowledge of an optimal integer solution x^*. Assume that the items are sorted according to descreasing efficiencies and let

$$a := \min\{j|x_k^* = 0|\}, \quad b := \max\{j|x_j^* = 1\}.$$

The core is given by the items in the interval $C = \{a, \ldots, b\}$. It is obvious that the split item is always part of the core.

The KP Core problem (KPC) is defined as

$$\begin{aligned} &\text{maximize} \sum_{j \in C} p_j x_j + \tilde{p} \\ &\text{subject to} \sum_{j \in C} w_j x_j \leqslant c - \tilde{w} \\ &\qquad x_j \in \{0,1\}, \quad j \in C. \end{aligned}$$

with $\tilde{p} = \sum_{j=1}^{a-1} p_j$ and $\tilde{w} = \sum_{j=1}^{a-1} w_j$. The solution of KPC would suffice to compute teh optimal solution of KP, which however, has to be already partially known to determine C. Nevertheless an approximate core $C = \{s-\delta, \ldots, s+\delta\}$, of fixed size $|C| = 2\delta+1$ is considered for a heuristic reduction of the problem.

Figure 1 exemplifies the core of a hypothetical KP instance with 13 items. The first row represents the efficiency value of each item and the second row represents the value of each variable on the LP-relaxation optimal solution. The items are sorted in descending order of efficiency value. The last row illustrates the variable fixing after the defined core. An asterisk indicates a free variable.

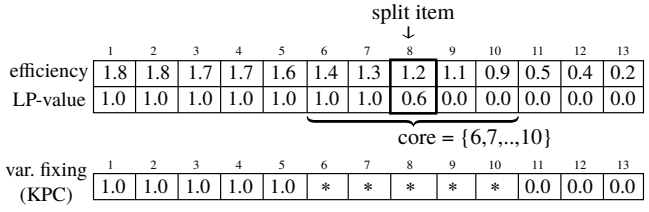

Figure 1: Example of core for a hypothetical KP instance with $n = 13$ and core size of 5 ($\delta = 2$).

A. The Core Concept for MKP

The previous definition of the core for KP can be extended to MKP without major difficulties, once an efficiency measure is defined for the MKP, as addressed in [6]. Even though a proper efficiency measure for MKP is not obvious due the multidimensional weight of the items. A well accepted efficiency measure is discussed in Section II-B.

Let x^* be an optimal solution and assume that the items are sorted in descending order after a given efficiency. Then let

$$a = \min\{j|x_j^* = 0\}, \quad b = \max\{j|x_j^* = 1\}.$$

The core is given by the items in the interval $C = \{a, \ldots, b\}$, and the multidimensional knapsack core problem (MKPC) defined as

$$\begin{aligned} &\text{maximize} \sum_{j \in C} p_j x_j + \tilde{p} \\ &\text{subject to} \sum_{j \in C} w_{ij} x_j \leqslant c_i - \tilde{w}_i, \quad i = 1, \ldots, m \\ &\qquad x_j \in \{0,1\}, \quad j \in C. \end{aligned}$$

with $\tilde{p} = \sum_{j=1}^{a-1} p_j$ and $\tilde{w}_i = \sum_{j=1}^{a-1} w_{ij}, i = 1, \ldots, m$.

In contrast to KP, the solution of the LP-relaxation of MKP in general does not consists of a single fractional split item. But up to m fractional values give rise to a whole *split interval* $S = \{s_1, \ldots, s_m\}$, where s_1 and s_m are respectively the first and the last index of variables with fractional values after sorting by the given efficiency measure. Once the split interval is defined, a central index value $s = \lfloor \frac{s_1+s_m}{2} \rfloor$ can be used as the center of a approximate core.

B. The Dual-variable Efficiency Measure

In [6] several proposals for efficiency measure was investigated and...

$$x_l^{LP} = \begin{cases} 1 & \text{if } e_j > 1, \\ \in [0,1] & \text{if } e_j = 1, \\ 0 & \text{if } e_j < 1. \end{cases}$$

$$e = \frac{p_j}{\sum_{i=1}^{m} r_i w_{ij}}$$

where r_i are set to the values of an optimal solution to the dual problem of the MKP's LP-relaxation, as suggested in [7].

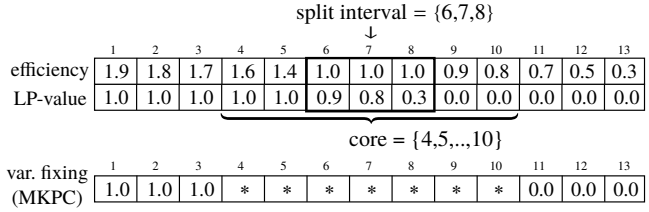

Figure 2: Example of core for a hypothetical MKP instace with $n = 13$, $m = 3$ and core size of 7 ($\delta = 3$).

[6]

III. The shuffled complex evolution for the MKP

The shuffled complex evolution is a population based evolutionary optimization algorithm that regards a natural evolution happening simultaneously in independent communities. The algorithm works with a population partitioned in N complexes, each one having M individuals. In the next Subsection the SCE is explained in more details. In the later Subsection the application of SCE to the multidimensional knapsack problem is considered.

A. The shuffled complex evolution

In the SCE a population of $N*M$ individuals is randomly taken from the feasible solution space. After this initializing the population is sorted by descending order according to their fitness and the best global solution is identified. The entire population is then partitioned (shuffled) into N complexes, each containing M individuals. In this shuffling process the first individual goes to the first complex, the second individual goes to the second complex, individual N goes to N-th complex, individual $M+1$ goes back to the first complex, etc.

The next step after shuffling the complexes is to evolve each complex through a given fixed amount of K' steps. In each step a subcomplex of P individuals is selected from the complex using a triangular probability distribution, where the i-th individual has a probability $p_i = \frac{2(n+1-i)}{n(n+1)}$ of being selected. The use of triangular distribution is intended to prioritize individuals with better fitness, supporting the algorithm convergence rate.

After the selection of the subcomplex, its worst individual is identified to be replaced by a new generated solution. This new solution is generated by the crossing of the worst individual and an other individual with better fitness. At first the best individual of the subcomplex is considered for the crossing. If the new solution is not better than the worst one, the best individual of the complex is considered for a crossing. If the latter crossing did not result in any improvement, the best individual of whole population is considered. Finally, if all the crossing steps couldn't generate a better individual, the worst individual of the subcomplex is replaced by a new random solution taken from the feasible solution space. This last step is important to prevent the algorithm becoming trapped in local minima. Fig. 3 presents the procedure described above in a flowchart diagram.

After evolving all the N complexes the whole population is again sorted by descending and the process continues until a stop condition is satisfied. Fig. 4 shows the SCE algorithm in a flowchart diagram.

B. The shuffled complex evolution for the MKP

As it can be noted in its description the SCE is easly applied to any optimization problem. The only steps needed to be specified is (a) the creation of a new random solution and (b) the crossing procedure of two solutions. These two procedures are respectively presented by Fig. 5 and Fig. 6.

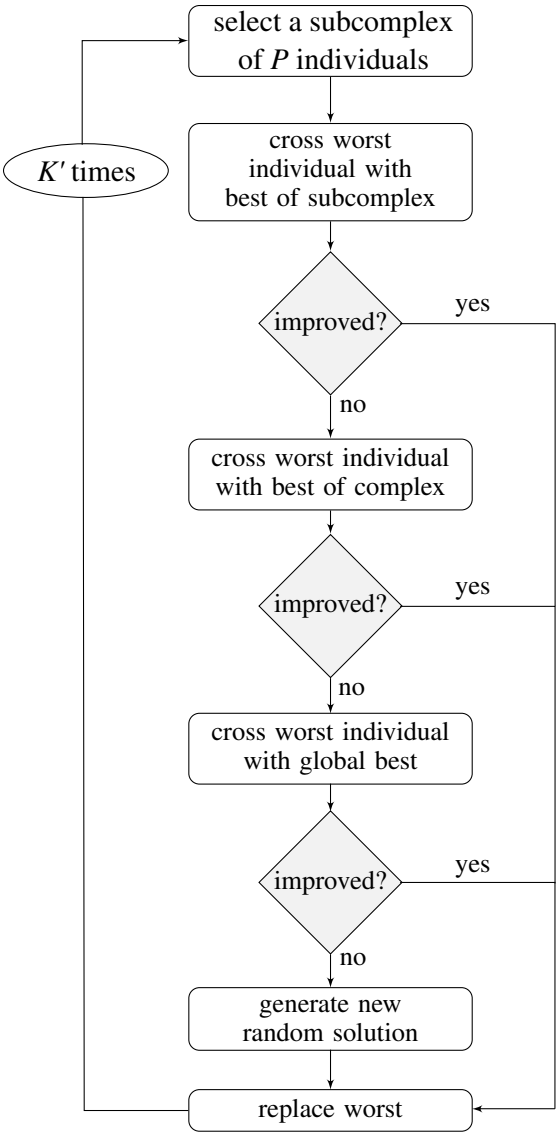

Figure 3: The evolving stage of SCE for a single complex.

To construct a new random solution (Fig. 5) the items are at first shuffled in random order and stored in a list (line 2). A new empty solution is then defined (line 3). The algorithm iteratively tries to fill the solution's knapsack with the an item taken from the list (lines 4-9). The feasibility of the solution is then checked: if the item insertion let the solution unfeasible (line 6) its removed from knapsack (line 7). After trying to place all available items the new solution is returned.

The crossing procedure (Fig. 6) takes as input the worst solution taken from the subcomplex $x^w = (x_1^w, x_2^w, \ldots, x_n^w)$, the selected better solution $x_b = (x_1^b, x_2^b, \ldots, x_n^b)$ and the number c of genes that will be carried from the better

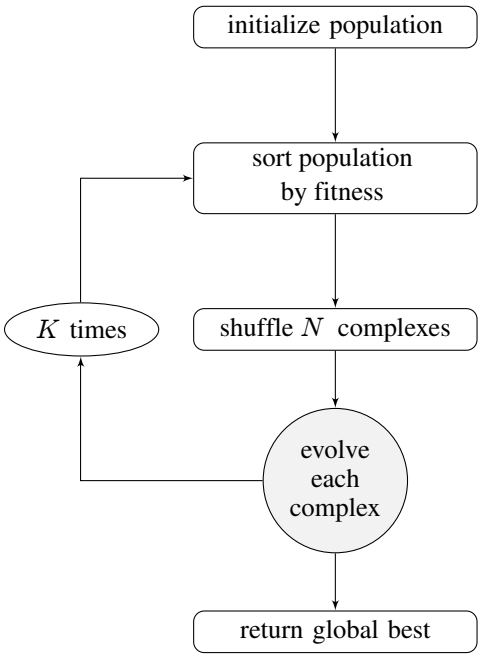

Figure 4: The shuffled complex evolution algorithm.

1: **procedure** NEW RANDOM SOLUTION
2: $v \leftarrow$ shuffle$(1, 2, \ldots, n)$
3: $s \leftarrow \emptyset$ ▷ empty solution
4: **for** $i \leftarrow 1 : n$ **do**
5: $s \leftarrow s \cup \{v_i\}$ ▷ adding item
6: **if** s is not feasible **then** ▷ checking feasibility
7: $s \leftarrow s - \{v_i\}$
8: **end if**
9: **end for**
10: return s
11: **end procedure**

Figure 5: Generation of a new random solution for the MKP.

1: **procedure** CROSSING(x^w : worst individual, x^b : better individual, c)
2: $v \leftarrow$ shuffle$(1, 2, \ldots, n)$
3: **for** $i \leftarrow 1 : c$ **do**
4: $j \leftarrow v_i$
5: $x^w_j \leftarrow x^b_j$ ▷ gene carriage
6: **end for**
7: **if** s^w is not feasible **then**
8: repair s^w
9: **end if**
10: update s^w fitness
11: return s^w
12: **end procedure**

Figure 6: Crossing procedure used on SCE algorithm.

solution. The c parameter will control how similar the worst individual will be from the given better individual. At first the items are shuffled in random order and stored in a list (line 2). Then c randomly chosen genes are carried from the better individual to the worst individual (line 5). At the end of steps the feasibility of the solution is checked (line 7) and the solution is repaired if needed. The repair stage is a greedy procedure that iteratively removes the item that less decreases the objective function. Finally the fitness of the generated solution is updated (line 10) and returned (line 11).

IV. CORE APPLICATION...

V. COMPUTATIONAL EXPERIMENTS

For the computational experiments a batch of tests was driven to find the best parameters for the problem. Afterwards two main tests was considered: (a) using the well-known set of problems defined by Chu and Beasley ([7]) and (b) a large set of randomly generated instances using uniform distribution.

A. The Chu-Beasley instances

The set of MKP instances provided by Chu and Beasley was generated using a procedure suggested by Freville and Plateau [8], which attempts to generate instances hard to solve. The number of constraints m varies among 5, 10 and 30, and the number of variables n varies among 100, 250 and 500.

The w_{ij} were integer numbers drawn from the discrete uniform distribution $U(0, 1000)$. The capacity coefficient c_i were set using $b_i = \alpha \sum_{j=1}^{n} w_{ij}$ where α is a tightness ratio and varies among 0.25, 0.5 and 0.75. For each combination of (m, n, α) parameters, 10 random problems was generated, totaling 270 problems. The profit p_j of the items were correlated to w_{ij} and generated as follows:

$$p_j = \sum_{i=1}^{m} \frac{w_{ij}}{m} + 500 q_j \qquad j = 1, \ldots, n$$

B. The set of random instances

The second set of instances is composed by problems generated using a similar setup. The only differences is that the profit p_j is also drawn from a discrete uniform distribution $U(0, 1000)$. For each combination of (m, n, α) parameter, 600 random problems was generated, totaling 16200 problems.

C. Experimental results

All the experiments was run on a Intel^R Core i5-3570 CPU @3.40GHz computer with 4GB of RAM. The SCE algorithm for MKP was implemented in C programming language. For the set of random instance all best known solution was found by the solver SCIP 3.0.1 running for at least 10 minutes.

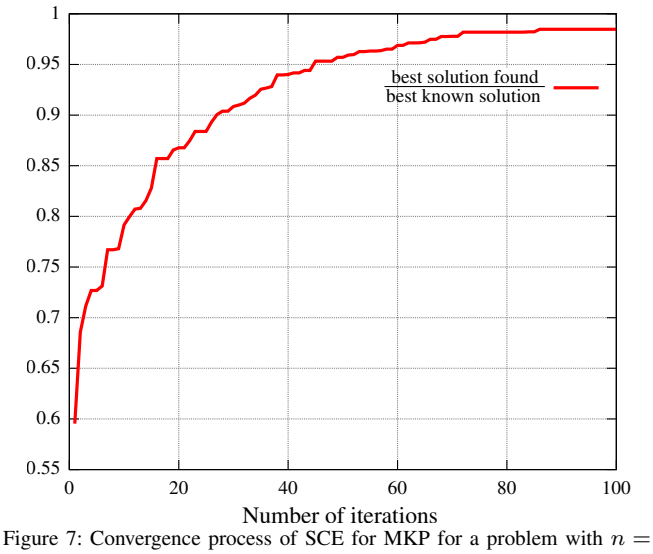

Figure 7: Convergence process of SCE for MKP for a problem with $n = 500$, $m = 30$ and $t = 0.50$.

After a previous test batch the following parameters for SCE was defined and used in all executions of SCE:

- $N = 20$: number of complexes;
- $M = 20$: number of individuals in each complex;
- $P = 5$: number of individuals in each subcomplex;
- $K = 300$: number of algorithm iterations;
- $K' = 20$: number of iterations used in the complex evolving process;
- $c = n/5$: number of genes carried from parent in crossing process.

Table I shows the performance of the SCE on the Chu-Beasley set of instance. Each instance in the set was executed 10 times on SCE. The *SCE time* column shows the average execution time of SCE algorithm. The *gap* column shows the average ratio of the solution found by SCE and the best known solution of each instance. It can be observed that the SCE has a fast convergence speed, achieving high quality solutions in few seconds.

The fast convergence speed of SCE for MKP can be noticed in Fig. 7. The figure shows for each iterations step, the quality of best solution found for the first 100 iterations. The problem instance used was taken from the second set of problem (random instances). The best known solution was found with 600s of execution on SCIP solver and the execution of the SCE algorithm expended 1.1 seconds.

n	m	α	SCE time (s)	gap (%)
100	5	0.25	0.79	96.5
		0.5	0.81	97.4
		0.75	0.83	98.9
	10	0.25	0.75	95.7
		0.5	0.93	96.7
		0.75	0.89	98.5
	30	0.25	1.01	95.4
		0.5	1.07	96.4
		0.75	0.99	98.2
			average gap	**97.1**
n	**m**	α	**SCE time (s)**	**gap (%)**
250	5	0.25	1.72	93.2
		0.5	1.75	94.9
		0.75	1.78	97.6
	10	0.25	1.84	93.1
		0.5	1.84	94.6
		0.75	1.81	97.2
	30	0.25	2.21	93.2
		0.5	2.21	94.2
		0.75	2.31	96.6
			average gap	**95.0**
n	**m**	α	**SCE time (s)**	**gap (%)**
500	5	0.25	3.16	91.4
		0.5	3.18	93.4
		0.75	3.34	96.4
	10	0.25	3.39	91.7
		0.5	3.37	93.1
		0.75	3.44	96.2
	30	0.25	3.83	91.4
		0.5	3.90	92.6
		0.75	3.99	96.0
			average gap	**93.6**

Table I: SCE performance on Chu-Beasley problems.

VI. Conclusions and Future Remarks

In this work we addressed the application of the shuffled complex evolution (SCE) to the multidimensional knapsack problem and investigated it performance through several computational experiments.

The SCE algorithm, which combines the ideas of a controlled random search with the concepts of competitive evolution proved to be very effective in finding good solution for hard instances of MKP, demanding a very small amount of processing time to reach high quality solutions for MKP.

Future work includes the investigation of different crossing procedures and the use of local search in the process of evolving complexes.

References

[1] M. J. Magazine and M.-S. Chern, "A note on approximation schemes for multidimensional knapsack problems," *Mathematics of Operations Research*, vol. 9, no. 2, pp. 244–247, 1984.

[2] Q. Duan, S. Sorooshian, and V. Gupta, "Effective and efficient global optimization for conceptual rainfall-runoff models," *Water resources research*, vol. 28, no. 4, pp. 1015–1031, 1992.

[3] F. Zhao, J. Zhang, J. Wang, and C. Zhang, "A shuffled complex evolution algorithm with opposition-based learning for a permutation flow shop scheduling problem," *International Journal of Computer Integrated Manufacturing*, no. ahead-of-print, pp. 1–16, 2014.

n	m	α	SCIP time (s)	SCE time (s)	gap (%)
100	10	0.25	0.93	0.41	98.3
		0.50	0.28	0.39	99.3
		0.75	0.09	0.37	99.8
	20	0.25	3.15	0.41	98.2
		0.50	0.71	0.40	99.3
		0.75	0.16	0.37	99.8
	30	0.25	7.26	0.42	98.3
		0.50	1.47	0.42	99.3
		0.75	0.25	0.38	99.8
				average gap	**99.1**
n	**m**	α	**SCIP time (s)**	**SCE time (s)**	**gap (%)**
250	10	0.25	58.20	1.10	97.2
		0.50	8.51	1.04	98.9
		0.75	0.51	0.90	99.7
	20	0.25	227.94	1.11	97.6
		0.50	43.69	1.02	99.0
		0.75	1.59	0.90	99.8
	30	0.25	270.48	1.20	97.7
		0.50	88.73	1.09	99.0
		0.75	2.90	0.94	99.8
				average gap	**98.7**
n	**m**	α	**SCIP time (s)**	**SCE time (s)**	**gap (%)**
500	10	0.25	278.85	2.23	96.1
		0.50	177.32	2.14	98.4
		0.75	8.47	1.87	99.6
	20	0.25	284.11	2.30	96.7
		0.50	275.68	2.16	98.6
		0.75	33.67	1.90	99.7
	30	0.25	283.78	2.50	96.9
		0.50	283.54	2.32	98.7
		0.75	71.66	1.96	99.7
				average gap	**98.3**

Table II: SCE performance on the random generated problems.

[4] E. Elbeltagi, T. Hegazy, and D. Grierson, "A modified shuffled frog-leaping optimization algorithm: applications to project management," *Structure and Infrastructure Engineering*, vol. 3, no. 1, pp. 53–60, 2007.

[5] E. Balas and E. Zemel, "An algorithm for large zero-one knapsack problems," *operations Research*, vol. 28, no. 5, pp. 1130–1154, 1980.

[6] J. Puchinger, G. R. Raidl, and U. Pferschy, "The core concept for the multidimensional knapsack problem," in *Evolutionary Computation in Combinatorial Optimization*. Springer, 2006, pp. 195–208.

[7] P. C. Chu and J. E. Beasley, "A genetic algorithm for the multidimensional knapsack problem," *Journal of Heuristics*, vol. 4, no. 1, pp. 63–86, Jun. 1998. [Online]. Available: http://dx.doi.org/10.1023/A:1009642405419

[8] A. Freville and G. Plateau, "An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem," *Discrete Applied Mathematics*, vol. 49, no. 1, pp. 189–212, 1994.