A shuffled complex evolution algorithm for the multidimensional knapsack problem

Marcos Daniel Valadão Baroni* and Flávio Miguel Varejão

Universidade Federal do Espírito Santo,
Av. Fernando Ferrari, 514, Goiabeiras, Vitória, ES, Brazil
http://ninfa.inf.ufes.br/

Abstract. This work addresses the application of a population based evolutionary algorithm called shuffled complex evolution (SCE) in the multidimensional knapsack problem. The SCE regards a natural evolution happening simultaneously in independent communities. The performance of the SCE algorithm is verified through computational experiments using well-known problems from literature and randomly generated problem as well. The SCE proved to be very effective in finding good solutions demanding a very small amount of processing time.

Keywords: Multidimensional knapsack problem, Meta-heuristics, Artificial Intelligence

1 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:

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

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.

* Research supported by Fundação de Amparo à Pesquisa do Espírito Santo.

The multidimensional knapsack problem can be applied on budget planning scenarios and project selections [8], cutting stock problems [7], loading problems [10], allocation of processors and databases in distributed computer programs [6].

The problem is a generalization of the well-known knapsack problem (KP) in which $m = 1$. However it is a NP-hard problem significantly harder to solve in practice than the KP. Due its simple definition but challenging difficulty of solving, the MKP is often used to to verify the efficiency of novel metaheuristics. A good review for the MKP is given by [5].

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 [3], 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 [11] and project management [4].

The reminder of the paper is organized as follows: Section 2 presents the shuffled complex evolution algorithm and proposes its application on the MKP. Section 3 comprises several computational experiments. In section 4 we make our concluding remarks about the experimental results.

2 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.

2.1 The shuffled complex evolution

In the SCE a population of $N*M$ individuals is randomly taken from the feasible solution space. After this initialization 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. The individuals in each complex is sorted by descending order of fitness quality. 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. 1(b) presents the evolving procedure described above in a flowchart diagram.

After evolving all the N complexes the whole population is again sorted by descending order of fitness quality and the process continues until a stop condition is satisfied. Fig. 1(a) shows the SCE algorithm in a flowchart diagram.

2.2 The shuffled complex evolution for the MKP

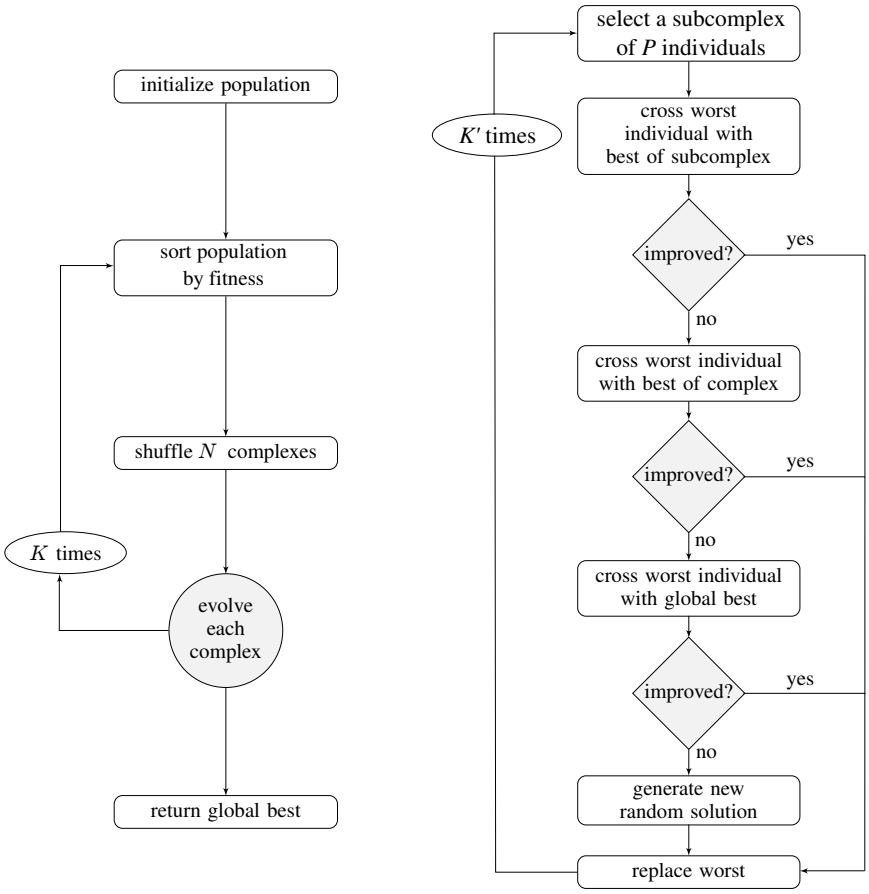
As it can be noted in its description the SCE is easly applied to any optimization problem. The only steps needed to be specified is (a) the creation of a new random solution and (b) the crossing procedure of two solutions. These two procedures are respectively presented by Algorithm. 1 and Algorithm 2.

Algorithm 1 Generation of a new random solution for the MKP.

```
1: procedure NEW RANDOM SOLUTION
2:     v ← shuffle(1, 2, ..., n)
3:     s ← ∅                                   ▷ empty solution
4:     for i ← 1 : n do
5:         s ← s ∪ {v_i}                       ▷ adding item
6:         if s is not feasible then           ▷ checking feasibility
7:             s ← s − {v_i}
8:         end if
9:     end for
10:    return s
11: end procedure
```

To construct a new random solution (Algorithm 1) 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 (Algorithm 2) 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 $x_b =$



(a) The SCE algorithm overview. (b) Evolving stage for a single complex.

Fig. 1. Flowchart representing the shuffled complex evolution algorithm.

$(x_1^b, x_2^b, \ldots, x_n^b)$ and the number c of genes that will be carried from the better 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).

Algorithm 2 Crossing procedure used on SCE algorithm.

```
1: procedure CROSSING(x^w : worst individual, x^b : better individual, c)
2:     v ← shuffle(1, 2, ..., n)
3:     for i ← 1 : c do
4:         j ← v_i
5:         x^w_j ← 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
```

3 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 [2] and (b) a large set of randomly generated instances using uniform distribution.

The set of MKP instances provided by Chu and Beasley was generated using a procedure suggested by Freville and Plateau [5], 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:

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

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.

All the experiments was run on a Intel 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. SCIP [1] is an open-source integer programming solver which implements the branch-and-cut algorithm [9].

After a previous test batch parameters for SCE was defined as shown in Table 1 and used in all executions of SCE.

Parameter	Value	Description
N	20	# of complexes
M	20	# of individuals in each complex
P	5	# of individuals in each subcomplex
K	300	# of algorithm iterations
K'	20	# of iterations used in the complex evolving process
c	$n/5$	# of genes carried from parent in crossing process

Table 1. Parameters used in SCE algorithm.

Table 2 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 t* column shows the average execution time of SCE. 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.

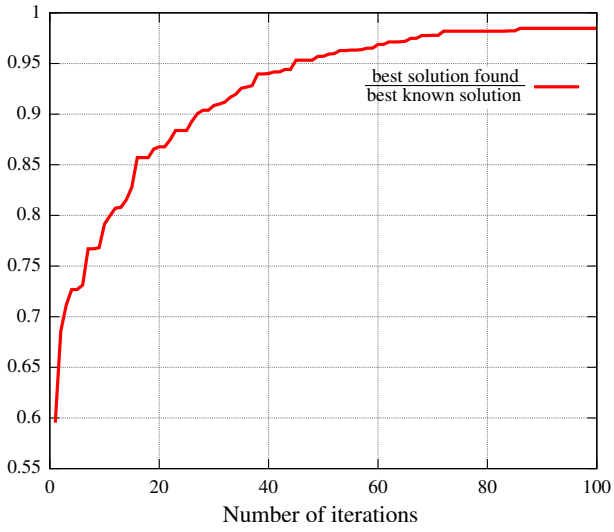


Fig. 2. Convergence process of SCE for MKP for a problem with $n = 500$, $m = 30$ and $t = 0.50$.

The fast convergence speed of SCE for MKP can be noticed in Fig. 2. 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 t (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 t (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 t (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 2. SCE performance on Chu-Beasley problems.

n	m	α	SCIP t (s)	SCE t (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 t (s)**	**SCE t (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 t (s)**	**SCE t (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 3. SCE performance on the random generated problems.

4 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. Achterberg, T.: Scip: Solving constraint integer programs. Mathematical Programming Computation 1(1), 1–41 (2009), http://mpc.zib.de/index.php/MPC/article/view/4
2. Chu, P.C., Beasley, J.E.: A genetic algorithm for the multidimensional knapsack problem. Journal of Heuristics 4(1), 63–86 (Jun 1998), http://dx.doi.org/10.1023/A:1009642405419
3. Duan, Q., Sorooshian, S., Gupta, V.: Effective and efficient global optimization for conceptual rainfall-runoff models. Water resources research 28(4), 1015–1031 (1992)
4. Elbeltagi, E., Hegazy, T., Grierson, D.: A modified shuffled frog-leaping optimization algorithm: applications to project management. Structure and Infrastructure Engineering 3(1), 53–60 (2007)
5. Freville, A., Plateau, G.: An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem. Discrete Applied Mathematics 49(1), 189–212 (1994)
6. Gavish, B., Pirkul, H.: Allocation of data bases and processors in a distributed computing system (1982)
7. Gilmore, P.C., Gomory, R.E.: The Theory and Computation of Knapsack Functions. Operations Research 14, 1045–1074 (1966)
8. McMillan, C., Plaine, D.: Resource allocation via 0–1 programming. Decision Sciences 4, 119–132 (1973)
9. Padberg, M., Rinaldi, G.: A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. SIAM review 33(1), 60–100 (1991)
10. Shih, W.: A Branch and Bound Method for the Multiconstraint Zero-One Knapsack Problem. Journal of The Operational Research Society 30, 369–378 (1979)
11. Zhao, F., Zhang, J., Wang, J., Zhang, C.: A shuffled complex evolution algorithm with opposition-based learning for a permutation flow shop scheduling problem. International Journal of Computer Integrated Manufacturing (ahead-of-print), 1–16 (2014)