A shuffled frog leaping algorithm for the multidimensional knapsack problem

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*—The abstract goes here.**

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:

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

The problem can be interpreted as a set of n itens 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 ([2]) which combines the ideas of a controlled random search with the concepts of competitive evolution and shuffling.

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

II. The shuffled complex evolution for the MKP

The shuffled complex evolution ([2]) is a population based evolutionary optimization algorithm that regards a natural evolution happenning 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 (suffled) 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 subcomplex is replaced by a new random solution taken from the feasible solution space. This last step is

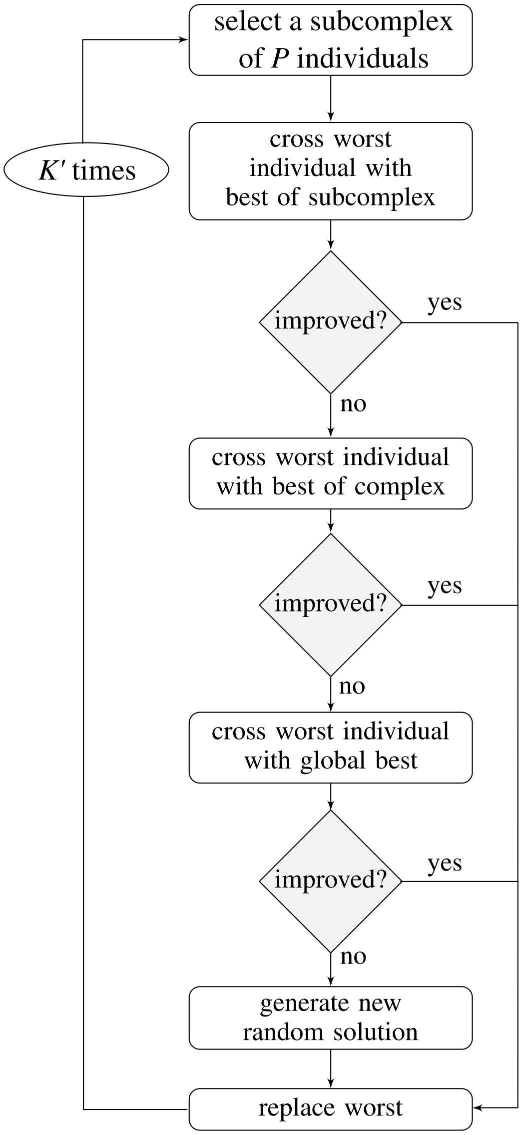

Fig. 1: The evolving stage of SCE for a single complex.

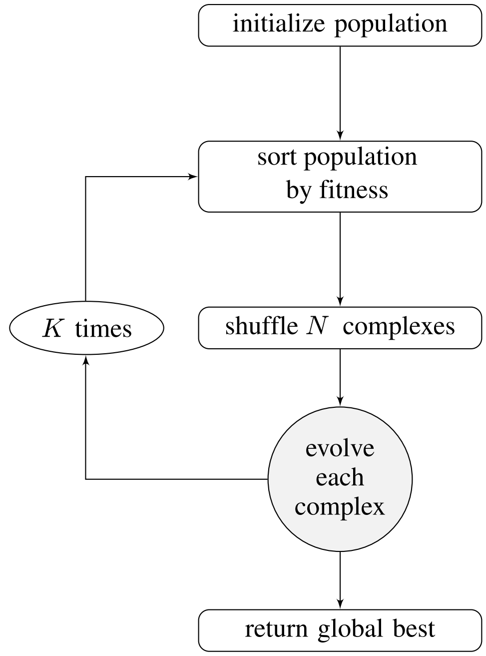

Fig. 2: The shuffled complex evolution algorithm.

```
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
```

Fig. 3: Generation of a new random solution for the MKP.

important to prevent the algorithm becoming trapped in local minima. Fig. 1 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. 2 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. 3 and Fig. 4.

To construct a new random solution (Fig. 3) 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. 4) 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 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 chosen genes are now 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 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).

procedure CROSSING(x^w : worst individual, x^b : better individual, c)
  $v \leftarrow$ shuffle$(1, 2, \ldots, n)$
  for $i \leftarrow 1 : c$ **do**
    $j \leftarrow v_i$
    $x_j^w \leftarrow x_j^b$ ▷ carriage of gene
  end for
  if s^w is not feasible **then**
    repair s^w
  end if
  update s^w fitness
  return s^w
end procedure

Fig. 4: Crossing procedure used on SCE algorithm.

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

A. *The Chu-Beasly instances*

The set of MKP instances provided by Chu and Beasly was generated using a procedure suggested by Freville and Plateau [4], 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 difference 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.

IV. CONCLUSIONS AND FUTURE REMARKS

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] 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

[4] 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.