A SHUFFLED COMPLEX EVOLUTION ALGORITHM FOR THE MULTIDIMENSIONAL KNAPSACK PROBLEM USING CORE CONCEPT

MARCOS BARONI & FLÁVIO VAREJÃO
UNIVERSIDADE FEDERAL DO ESÍRITO SANTO, VITÓRIA, ES, BRAZIL

THE PROBLEM (MKP)

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:

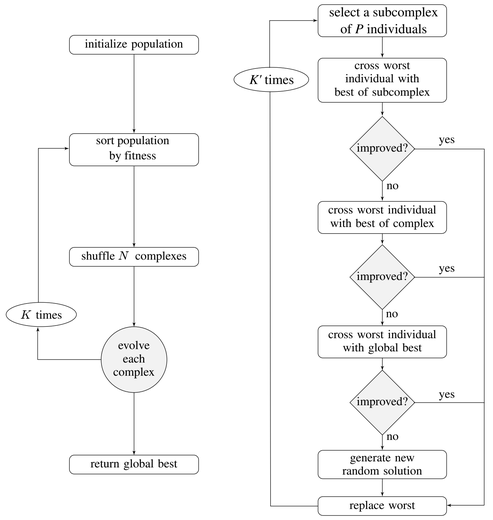
$$\max \sum_{j=1}^{n} p_j x_j$$
$$\text{s. to } \sum_{j=1}^{n} w_{ij} x_j \leqslant c_i \quad i \in \{1,\ldots,m\}$$
$$x_j \in \{0,1\}, \quad j \in \{1,\ldots,n\}.$$

The work address the application of a metaheuristic called shuffled complex evolution (SCE) to the MKP.

THE META-HEURISTIC (SCE)

The shuffled complex evolution is a population based evolutionary optimization algorithm that regards a natural evolution happening simultaneously in N independent communities (or complexes).

Initialy $N * M$ individuals are randomly taken from the feasible solution space and sorted according to their fitness. Subsequently a shuffling process places the 1st in the first complex, the 2nd in second complex, individual Nth goes to Nth complex, individual $M+1$ goes back to the first complex, etc.



The SCE algorithm.

Evolving stage.

The next step after shuffling the complexes is to evolve each complex through a 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, prioritizing individuals with better fitness.

Then the worst individual from the subcomplex is identified to be replaced by a new solution generated by its crossing with best individual of the subcomplex. If the new solution has not improved, the best individual of the complex is considered for crossing and latter the best individual of whole population. If all the crossing steps couldn't improve the worst solution, it is replaced by a new random solution.

THE CORE CONCEPT FOR MKP

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.

$$e_j = \frac{p_j}{w_j}, \tag{1}$$

The knapsack problem considers items $j = 1,\ldots,n$, 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. The KP Core problem (KPC) is defined as

$$\text{maximize} \sum_{j \in C} p_j x_j + \tilde{p} \tag{2}$$
$$\text{subject to} \sum_{j \in C} w_j x_j \leqslant c - \tilde{w} \tag{3}$$
$$x_j \in \{0,1\}, \quad j \in C. \tag{4}$$

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. Asterisks indicate free variables associated to the items in the approximate core.

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, 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 **??**.

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

$$a = \min\{j | x_j^* = 0\}, \quad b = \max\{j | x_j^* = 1\}. \tag{5}$$

In contrast to KP, the solution of the LP-relaxation of MKP in general does not consists of a single fractional split item.

THE SCE FOR 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 the creation of a new random solution (Algorithm 1) and the crossing procedure of two solutions (Algorithm 2).

Algorithm 1 Generation of a new random solution.

```
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
```

Algorithm 2 Crossing procedure used on SCE algorithm.

```
1:  procedure CROSSING(x^w : worst individual,
            x^b : better individual, c : n. of genes to be carried)
2:      v ← shuffle(1, 2, ..., n)
3:      for i ← 1 : c do
4:          j ← v_i
5:          x_j^w ← x_j^b                       ▷ 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
```

RESULTS

Two main tests was considered: (a) using the well-known set of problems defined by Chu and Beasley [1] and (b) a large set of randomly generated instances using uniform distribution. The number of constraints m varies among 5, 10 and 30, and the number of variables n varies among 100, 250 and 500.

		time (s)		quality (%)	
n	**m**	**SCE**	**SCEcr**	**SCE**	**SCEcr**
100	**5**	1.31(0.03)	0.17(0.00)	97.60(0.56)	99.83(0.02)
	10	1.43(0.04)	0.26(0.00)	96.96(0.99)	99.75(0.04)
	30	1.75(0.08)	1.01(0.04)	96.66(0.66)	98.89(0.11)
250	**5**	2.87(0.09)	0.69(0.01)	94.98(0.33)	99.92(0.00)
	10	3.08(0.09)	0.83(0.01)	94.95(0.35)	99.75(0.00)
	30	3.82(0.14)	1.45(0.05)	94.65(0.39)	98.89(0.04)
500	**5**	5.74(0.14)	1.23(0.01)	93.73(0.28)	99.86(0.00)
	10	5.85(0.34)	1.33(0.03)	93.65(0.25)	99.71(0.00)
	30	6.17(1.12)	1.84(0.19)	93.30(0.34)	99.28(0.01)

SCEcr performance on Chu-Beasley problems.

			time (s)		quality (%)	
#	**n**	**m**	**SCE**	**SCEcr**	**SCE**	**SCEcr**
01	100	15	1.47(0.00)	0.08(0.0)	97.66(0.03)	99.24(0.02)
02	100	25	1.61(0.00)	0.09(0.0)	97.94(0.04)	98.94(0.09)
03	150	25	2.51(0.01)	0.09(0.0)	97.22(0.04)	99.09(0.02)
04	150	50	3.56(0.03)	0.09(0.0)	97.40(0.04)	98.52(0.02)
05	200	25	3.55(0.01)	0.09(0.0)	96.88(0.03)	99.28(0.01)
06	200	50	4.81(0.09)	0.10(0.0)	97.68(0.02)	98.90(0.03)
07	500	25	7.30(0.09)	0.10(0.0)	97.12(0.01)	99.54(0.00)
08	500	50	12.20(0.47)	0.11(0.0)	97.27(0.01)	99.33(0.01)
09	1500	25	24.61(1.73)	0.12(0.0)	95.40(0.01)	98.22(0.00)
10	1500	50	33.79(2.44)	0.13(0.0)	97.50(0.00)	99.64(0.00)
11	2500	100	121.28(194.74)	0.15(0.0)	97.95(0.00)	99.70(0.00)

SCEcr performance on Glover-Kochenberger problems.

For the set of random instances all best known solution was found by the solver SCIP running for at least 10 minutes. SCIP is an open-source integer programming solver which implements the branch-and-cut algorithm.

EXPERIMENTAL SETUP

A batch of preliminary tests was driven to find the best parameters for the problem:

	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 of evolving process
c	$n/5$	# of genes carried from parent in crossing

All the experiments was run on a 3.40GHz computer with 4GB of RAM. SCE lgorithm was implemented in C programming language.

CONCLUSIONS

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 REMARKS

Future work includes the investigation of different crossing procedures, the use of local search in the process of evolving complexes and the application of problem reduction procedures for the MKP.

futer works....

REFERENCES

References

[1] P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4(1):63–86, June 1998.

[2] Qingyun Duan, Soroosh Sorooshian, and Vijai Gupta. Effective and efficient global optimization for conceptual rainfall-runoff models. *Water resources research*, 28(4):1015–1031, 1992.

ACKNOWLEDGEMENT

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