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

***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. 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:

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

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. It is considered an integer programming problem (IP) since its variables x_i are restricted to be integers.

The multidimensional knapsack problem can be applied on budget planning scenarios and project selections [1], cutting stock problems [2], loading problems [3], allocation of processors and databases in distributed computer programs [4].

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.

Its well known that the hardness of a NP-hard problem grows exponentially over its size. Thereupon, a suitable approach for tackling NP-hard problems is to reduce their size through some variable fixing procedure. Despite not guaranteeing optimality of the solution, an efficient variable fixing procedure may provide near optimal solutions through a small computational effort.

The SCE is a metaheuristic proposed by Duan in [5] 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 [6], project management [7] and MKP [8]. This work addresses the development of an hybrid heuristic using an efficient variable fixing procedure and the SCE metaheuristic, as an improvement proposal for the work in [8]

The reminder of the paper is organized as follows: Section II defines the core concept for the MKP and its application in the problem reduction. Section III presents the shuffled complex evolution algorithm and proposes its application on the MKP. Section IV comprises several computational experiments over well-known instances from literature. In section V we make our concluding remarks about the developed methods and the experimental results.

II. 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.

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.

Before defining the core of the KP it is worth to note the solution structure of its LP-relaxation. The LP-relaxation of an integer programming problem arises by replacing the constraint that each variable must be integer by a constraint that allows continuity of the variable. The LP-relaxation of a KP is found by replacing the constraint $x_i \in \{0,1\}$ by $0 \leqslant x_i \leqslant 1$ for all $i \in \{1, \ldots, n\}$. If the items are sorted according to decreasing efficiency values

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

it is known that the solution of the LP-relaxation consists of three consecutive parts: the first part contains variables set

to 1, the second pare consists of at most one split item s, whose corresponding LP-values is fractional, 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 optimal 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 core.

Balas and Zemel [9] 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 decreasing efficiencies and let

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

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

Figure 1 illustrates an exact core for an hypothetical KP instance with 13 items. First row represents the efficiency value e of each item. The items are sorted by descending order of efficiency. Second row represent solution array of the LP-relaxation. The third row represents the exact solution for the original problem. The 8-th item is the split item. Notice that the split item is within the exact core.

split item ↓

	1	2	3	4	5	6	7	8	9	10	11	12	13
efficiency	1.8	1.8	1.7	1.7	1.6	1.4	1.3	1.2	1.1	0.9	0.5	0.4	0.2
LP-value	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.6	0.0	0.0	0.0	0.0	0.0
IP-value	1.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0

exact core = {5,6,...,10}

Figure 1: Example of exact core for a hypothetical KP instance.

The KP Core problem (KPC) is defined as

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

with $\tilde{p} = \sum_{j=1}^{a-1} p_j$ and $\tilde{w} = \sum_{j=1}^{a-1} w_j$ respectively quantifying the total profit and the total weights of items fixed as selected. The solution of KPC would suffice to compute the 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 2 exemplifies an approximate 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. Asterisks indicate free variables associated to the items in the approximate core.

split item ↓

	1	2	3	4	5	6	7	8	9	10	11	12	13
efficiency	1.8	1.8	1.7	1.7	1.6	1.4	1.3	1.2	1.1	0.9	0.5	0.4	0.2
LP-value	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.6	0.0	0.0	0.0	0.0	0.0

core = {6,7,..,10}

variable fixing (KPC)	1	2	3	4	5	6	7	8	9	10	11	12	13
	1.0	1.0	1.0	1.0	1.0	*	*	*	*	*	0.0	0.0	0.0

Figure 2: Example of core for a hypothetical KP instance with $n = 13$ and approximate 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 [10], 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.

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

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

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

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

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 an approximate core.

B. The Dual-variable Efficiency Measure

For defining a reasonable efficiency measure for MKP consider the most obvious form of efficiency which is a direct generalization of the one-dimensional case:

$$e_j(simple) = \frac{p_j}{\sum_{i=1}^{m} w_{ij}}$$

In this definition different orders of magnitude of the constraints are not considered and a single constraint may

dominate the others. This drawback can be avoided by introducing relevance values r_i for every constraint:

$$e_j(relevance) = \frac{p_j}{\sum_{i=1}^{m} r_i w_{ij}}$$

Several proposals for setting the relevance values r_i were discussed and tested by Puchinger, Raidl and Pferschy in [10]. According to their work setting the relevance values r_i to the values of an optimal solution to the dual problem of the MKP's LP-relaxation, as suggested in [11], achieved the best results and will be the one considered in this work for the development of the hybrid heuristic.

While the original MKP can be seen as a resource allocation problem, the dual problem of the MKP can be seen as a resource valuation problem. For this reason the values of the dual solution is related to the *importance* of each resource.

The split interval resulting from the efficient measure considered can be precisely characterized. Let x^{LP} be the optimal solution of the LP-relaxation of MKP. Then the following relation holds, as proved in [10]:

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

From this relation we can note that all variables in the split interval will have efficiency measure $e_j = 1$, while less and more efficient ones will have $e_j < 1$ and $e_j > 1$ respectively.

Figure 3 exemplifies an approximate core of a hypothetical MKP with 13 variables and 3 dimensions. Notice that the LP-relaxation solution has now 3 fractional variables that defines the center of the split interval.

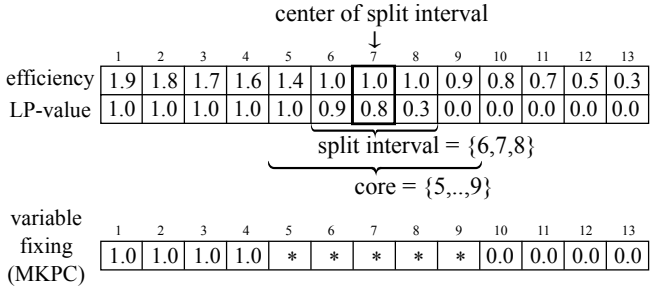

Figure 3: Example of core for a hypothetical MKP instance with $n = 13$, $m = 3$ and approximate core size of 5 ($\delta = 2$).

The following section presents the shuffled complex evolution algorithm and proposes its application on the MKP.

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 subsection III-A the SCE is explained in more details, while subsection III-B considers its application to 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
```

Figure 4: Generation of a new random solution for the MKP.

A. 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 $(M + 1)$-th 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 are 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. 5 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. 6 shows the SCE algorithm in a flowchart diagram in which the stop condition is a fixed amount of K evolving steps.

B. The shuffled complex evolution for the MKP

As it can be noted in its description the SCE is easily applied to any optimization problem. The only steps needed to

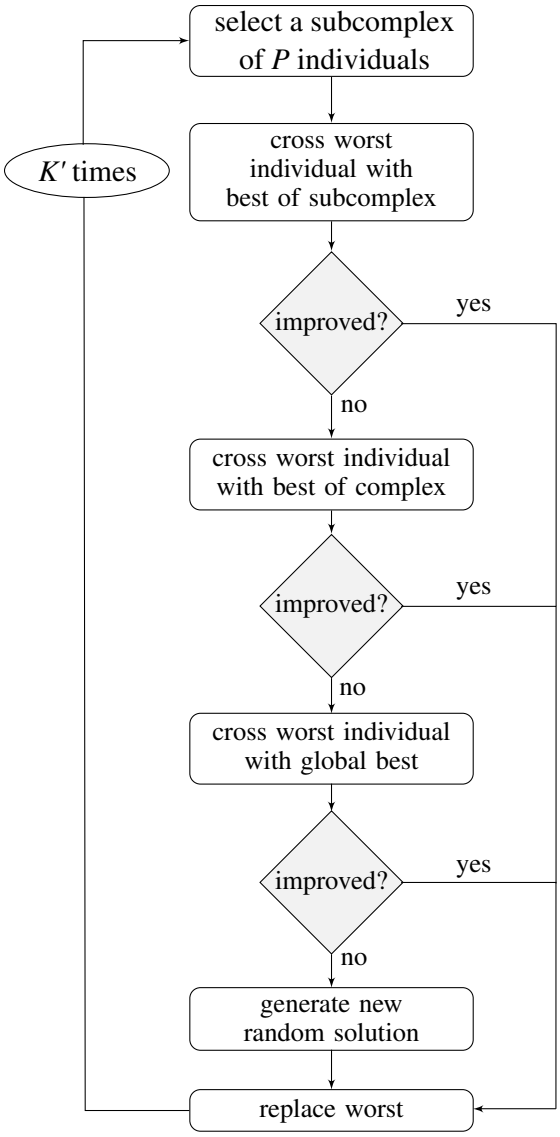

Figure 5: The evolving stage of SCE for a single complex.

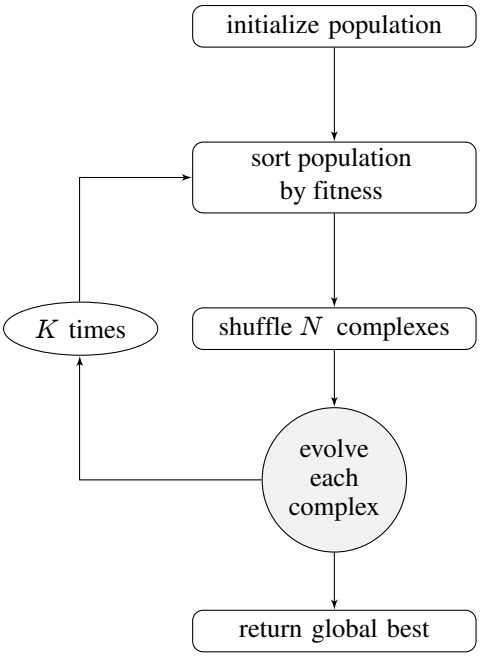

Figure 6: The shuffled complex evolution algorithm.

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: $x_j^w \leftarrow x_j^b$ ▷ 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**

Figure 7: Crossing procedure used on SCE algorithm.

be specified is (a) the creation of a new random solution and (b) the crossing procedure of two solutions. The procedures presented in this section are based on the work in [8]. These two procedures for the MKP are respectively presented by algorithms on Fig. 4 and Fig. 7.

To construct a new random solution (Fig. 4) 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. 7) 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 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. Computational experiments

To verify the efficiency of the proposed hybrid metaheuristic, several computational experiments was executed considering the original SCE for the MKP proposed in [8] and the hybrid SCE, proposed in this work. For brevity the hybrid SCE proposed in this work will be referred to as SCEcr .

For the experiments, two sets of MKP instances was used: a first set composed by 270 instances provided by Chu and Beasley ([11]) and a second set composed by 11 instances provided by Glover and Kochenberger in [12]. These instances are all available at [13].

The best known solution for all instances were taken from the literature and were found by different algorithms which we had no access to the implementation. In mosts cases those are exact algorithms which took minutes or hours of execution time.

The set of MKP instances provided by Chu and Beasley was generated using a procedure suggested by Freville and Plateau [14], 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:

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

A. Experimental results

All the experiments were run on a Intel^R Core i5-3570 CPU @3.40GHz computer with 4GB of RAM. The original SCE and SCEcr was both implemented in C programming language.

For the variable fixing procedure used on SCEcr , the range size of the approximate core was $|C| = m + \frac{n}{10}$ for all instances. In all instances the parameters used for SCE and SCEcr were the same recommended in [8] which was found after a batch test using Chu and Beasley instances:

- $N = 20$: 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 and SCEcr on the Chu-Beasley set of instances. Each instance in the set was executed 10 times for each algorithm. Columns *n*, *m* and α shows the parameters used on each instance generation. The *time* column shows the average execution time of the algorithms (lower is better). The *quality* column shows the average ratio of the solution found and the best known solution from literature ([15], [16]) of each instance (higher is better).

			time (s)		quality (%)	
n	**m**	α	**SCE**	**SCEcr**	**SCE**	**SCEcr**
100	5	0.25	0.79	0.15	96.5	99.73
		0.50	0.81	0.15	97.4	99.86
		0.75	0.83	0.13	98.9	99.91
	10	0.25	0.75	0.22	95.7	99.53
		0.50	0.93	0.22	96.7	99.76
		0.75	0.89	0.22	98.5	99.96
	30	0.25	1.01	1.00	95.4	99.02
		0.50	1.07	0.79	96.4	99.21
		0.75	0.99	0.77	98.2	99.52
	average		0.90	**0.41**	97.5	**99.50**
250	5	0.25	1.72	0.54	93.2	99.87
		0.50	1.75	0.55	94.9	99.94
		0.75	1.78	0.51	97.6	99.95
	10	0.25	1.84	0.67	93.1	99.57
		0.50	1.84	0.62	94.6	99.80
		0.75	1.81	0.64	97.2	99.88
	30	0.25	2.21	1.16	93.2	99.46
		0.50	2.21	0.96	94.2	99.36
		0.75	2.31	1.03	96.6	99.59
	average		1.94	**0.74**	95.0	**99.60**
500	5	0.25	3.16	0.85	91.4	99.77
		0.50	3.18	0.86	93.4	99.87
		0.75	3.34	0.84	96.4	99.92
	10	0.25	3.39	0.99	91.7	99.49
		0.50	3.37	0.97	93.1	99.78
		0.75	3.44	0.95	96.2	99.83
	30	0.25	3.83	1.53	91.4	99.75
		0.50	3.90	1.48	92.6	99.42
		0.75	3.99	1.42	96.0	99.68
	average		3.51	**1.10**	93.58	**99.61**

Table I: SCE and SCEcr performance on Chu-Beasley problems.

It can be observed that the SCEcr had faster convergence speed, achieving higher quality solutions in all cases, achieving at least 99.02% of best known, in less than 2 seconds for every instance.

It can be also noticed that SCEcr executed in much less processing time than original SCE. This is due the variable fixing procedure which reduced the problem size, resulting in less genes operations during the evolving procedures. The variable fixing procedure also brought robustness for the method, as the quality of the solution found increased in case of larger instances while on original SCE the quality decreased considerably.

Table II shows the performance of SCE and SCEcr on the Glover-Kochenberger set of instances. Each instance in the set was executed 10 times for each algorithm. Columns *n* and *m* indicate the size of each instance. The *time* column shows the average execution (lower is better). The *quality* column shows the average ratio of the solution found and the best known solution of each instance. It can be noticed that SCEcr achieved high quality solutions, at least 99.10% of best known solution, spending very small amount of processing time, compared to the time taken to find the best

			time (s)		quality (%)	
#	**n**	**m**	**SCE**	**SCEcr**	**SCE**	**SCEcr**
01	100	15	1.48	0.31	97.67	99.68
02	100	25	1.60	0.47	97.93	99.51
03	150	25	2.48	0.79	97.25	99.60
04	150	50	3.44	1.61	97.38	99.10
05	200	25	3.30	0.83	96.89	99.73
06	200	50	4.26	1.67	97.69	99.30
07	500	25	6.74	1.27	97.12	99.72
08	500	50	11.23	2.06	97.29	99.62
09	1500	25	22.28	1.83	96.90	99.32
10	1500	50	31.60	5.25	97.49	99.76
11	2500	100	107.76	11.94	97.94	99.77
average			17.83	**2.55**	97.41	**99.46**

Table II: SCEcr performance on Glover-Kochenberger problems.

known solutions.

V. Conclusions and future remarks

In this paper we addressed the development of a hybrid heuristic for the MKP implementing a population based algorithm called shuffled complex evolution for solving the multidimensional knapsack problem assisted by an efficient variable fixing procedure. Its performance was verified 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 able to achieve fast convergence ratio, finding good quality near optimal solutions, demanding small amount of computational time.

The application of the core concept, through a variable fixing procedure for MKP, proved to be efficient to reduce the size of the problems which provided fast execution time, producing higher quality solutions.

SCEcr algorithm presented faster convergence speed, achieving higher quality solutions in all cases, achieving at least 99.02% of best known, in less than 2 seconds for every instance. The variable fixing procedure also brought robustness for the method, as the quality of the solution found increased in case of larger instances.

The hybrid heuristic developed in this papter is very well suitable if it is necessary to compute a good solution in a small processing time. The hybrid heuristic could achieved 99.61% on average of quality of the best known solution for the 270 Chu-Beasley instances and 99.46% on average for the Glover-Kochenberger instances.

Future works include the investigation of different crossing procedures and the use of local search in the process of evolving complexes. Besides we want to investigate the possibility of using the efficiency measure concept directly on metaheuristics.

References

[1] C. McMillan and D. Plaine, "Resource allocation via 0–1 programming," *Decision Sciences*, vol. 4, pp. 119–132, 1973.

[2] P. C. Gilmore and R. E. Gomory, "The Theory and Computation of Knapsack Functions," *Operations Research*, vol. 14, pp. 1045–1074, 1966.

[3] W. Shih, "A Branch and Bound Method for the Multiconstraint Zero-One Knapsack Problem," *Journal of The Operational Research Society*, vol. 30, pp. 369–378, 1979.

[4] B. Gavish and H. Pirkul, "Allocation of data bases and processors in a distributed computing system," 1982.

[5] 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.

[6] 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.

[7] 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.

[8] M. D. V. Baroni and F. M. Varejão, "A shuffled complex evolution algorithm for the multidimensional knapsack problem," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. Springer, 2015, pp. 768–775.

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

[10] 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.

[11] 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

[12] F. Glover and G. A. Kochenberger, "Critical event tabu search for multidimensional knapsack problems," in *Meta-Heuristics*. Springer, 1996, pp. 407–427.

[13] J. E. Beasley. (2012) Or library. [Online]. Available: http://www.brunel.ac.uk/~mastjjb/jeb/info.html

[14] 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.

[15] Y. Vimont, S. Boussier, and M. Vasquez, "Reduced costs propagation in an efficient implicit enumeration for the 01 multidimensional knapsack problem," *Journal of Combinatorial Optimization*, vol. 15, no. 2, pp. 165–178, 2008.

[16] F. Della Croce and A. Grosso, "Improved core problem based heuristics for the 0/1 multi-dimensional knapsack problem," *Computers & Operations Research*, vol. 39, no. 1, pp. 27–31, 2012.