Military Technical College
Kobry El-Kobbah,
Cairo, Egypt

7th International Conference
on Electrical Engineering
ICEENG 2010

A New Genetic Algorithm for Multiple-Choice Multidimensional Knapsack Problem

By

A. F. Heikal * M. H. Rasmy ** A. A. Tharwat ** M. A. El-Beltagy **

Abstract:

A New Genetic Algorithm (NGA) for solving the Multiple Choice Multidimensional Knapsack Problem (MMKP) is presented in this paper. The MMKP can be applied to solve a wide variety of real life problems i.e., in any area where tasks must be scheduled or budgeted. This paper introduces NGA algorithm that hybridize the solution construction mechanism of GA operators (hybrid selection operator, hybrid cross-over operator and new hybrid mutation operator) for permutation encoding genetic algorithm. In addition we present a strong initial population is created by the Maximizing Value per Resources Consumption (MVRC) heuristic algorithm. The experimental results show that the method is very efficient and competitive to solve the MMKP compared with the better existing methods

Keywords:

Genetic Algorithm, MMKP, MVRC, Heuristic Algorithms.

* Egyptian Armed Forces
** Faculty of Computers and Information, Cairo University, Cairo, Egypt

1. Introduction:

The Multiple Choice Multidimensional Knapsack Problem (MMKP) is a combinatorial optimization problem and its one of the most complex members of the Knapsack Problem (KP) family. Actually, the MMKP is the combine aspects of the Multiple Choice Knapsack Problem MCKP and the Multidimensional Knapsack Problem MDKP. The (MCKP) is one kind of KP where the picking criteria for items are restricted. In this variant of KP there are one or more groups of items with the constraint that exactly one item has to be picked from each group. The (MDKP) is another KP, where the resources are multidimensional, i.e. there are multiple resource constraints for the knapsack.
The (MMKP) could be stated as follows: We are given m classes with each class i containing n_i items. The j_{th} item of class i has profit p_{ij} . Each item has l dimensions of weight, and the weight at dimension k is denoted as w_{ijk}. The knapsack has capacity ck on each dimension k. The goal is to select one item in each class to maximize the sum of their profits and to keep the total weight on each dimension no more than the corresponding capacity. It is generally considered that the profits, weights and the knapsack capacities take non-negative values, thus we will not explicitly state this constraint in the formulation.
Formally, MMKP could be expressed with an integer programming model
Maximize,

$$z = \sum_{i=1}^{m} \sum_{j=1}^{n_i} p_{ij} x_{ij} \tag{1}$$

Subject to:

$$\sum_{i=1}^{m} \sum_{j=1}^{n_i} w_{ijk} x_{ij} \le c_k , \quad k \in \{1, \dots, l\} \tag{2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \ , \ i \in \{1, \dots, m\} \tag{3}$$

$$x_{ij} \in \{0,1\} \ , \quad i \in \{1, \dots, m\} \ , \quad j \in \{1, \dots, n_i\} \tag{4}$$

Equation (1) provides the profit of selecting an item from every class, a value to be maximized. Equation (2) ensures the resource capacity of knapsack k is not exceeded while Equation (3) ensures selecting a single item from each of the i classes. Equation (4) is the binary selection requirement on decision variable x_{ij} such that $x_{ij} = 1$ if selected and $x_{ij} = 0$ if not selected.
Since the MMKP is a NP-hard problem, the computation time to reach the optimal solution increases exponentially with the size of the problem [1]. There are two types of solution methods: exact algorithms capable to produce the optimal solutions for some problem instances within a reasonable computational time, and approximate procedures or heuristics capable to produce “good” (near-optimal) solutions within small computational time.
Several heuristic algorithms have been proposed for MMKP. The first heuristic is proposed by Moser et al. [2] based on Lagrange multiplier method. Then HEU is proposed by Khan et al. [1, 3]. Akbar et al. [4] present two heuristic algorithms a

modified version of HEU named M-HEU and Incremental Heuristic I-HEU. Later, Hifi et al. proposed heuristic algorithms CPCCP, DerAlgo and MGLS in [5] and MRLS in [6]. Another heuristic HMMKP, proposed by Hernandez et al. [7]. A convex hull based method C-HEU was proposed by Akbar et al. [8]. Two algorithms based on the column generation method have been proposed recently by Cherfi and Hifi [9]. Chantzara et al. [10] presented the heuristic Maximizing Value per Resources Consumption (MVRC). Shahriar et all [11] presented MP-HEU, a multiprocessor algorithm based on a M-HEU heuristic. Hiremath [12] developed and examined three new greedy heuristic approaches TYPE, CH1, and CH2. Different meta-heuristics have already been applied to solve the MMKP like Tabu Search (TS) [12, 13], Simulated Annealing (SA) [14], Genetic Algorithm (GA) [15] , Ant Colony Optimization (ACO) and (ACO &PR) [16].
These heuristics and meta-heuristics are able to obtain fairly good solutions for large MMKP instances. In comparison, exact algorithms have not received a lot of attention. Indeed, only very few papers have been proposed in the literature, these papers are based on the branch-and-bound principle. [3, 17, 18].
MMKP has its application in many resource management problems, budgeting problem, adaptive multimedia problem, logistic scheduling problem, network design problem, warehouse container storage problem, resource allocation problem for satellites, airlift loading problem, and any area where tasks must be scheduled or budgeted. For more detailed see [12].

2. Genetic Algorithms:

Genetic algorithms are an evolutionary technique that use crossover and mutation operators to solve optimization problems using a survival of the fittest idea. They have been used successfully in a variety of different NP-hard problems. The technique does not ensure an optimal solution, however it usually gives good approximations in a reasonable amount of time. This, therefore, would be a good algorithm to try on the MMKP problem, one of the most famous NP-hard problems.
In 1975, Holland developed this idea in his book “Adaptation in natural and artificial systems”. He described how to apply the principles of natural evolution to optimization problems and built the first Genetic Algorithms. Holland’s theory has been further developed and now Genetic Algorithms (GAs) stand up as a powerful tool for solving search and optimization problems. Genetic algorithms are stochastic global search and optimization methods that mimic the metaphor of natural biological evolution. During the course of evolution, natural populations evolve according to the principle of natural selection and ”survival of the fittest”. Individuals which are more successful in adapting to their environment will have a better chance of surviving and reproducing, whilst individual which are less fit will be eliminated [19-20].
A GA simulates these processes by taking an initial population of individuals and applying genetic algorithm in each reproduction. In optimization terms each individual in the population is encoded into a string or chromosome which represents a possible

solution to a given problem. The fitness of an individual is evaluated with respect to a given objective function. Highly fit individuals or solutions have opportunities to reproduce by exchanging pieces of their genetic information, in a crossover procedure, with other highly fit individuals. This produces new "offspring" solutions (i.e., children), which share some characteristics taken from both parents.
In general, a genetic algorithm has five basic components, as summarized by Michalewicz [21] and the basic steps of a simple GA are shown in Figure 1. [22]
1. A genetic representation of solutions to the problem
2. A way to create an initial population of solutions
3. An evaluation function rating solutions in terms of their fitness
4. Genetic operators that alter the genetic composition of children during reproduction
5. Values for the parameters of genetic algorithms

```
Genetic Algorithm
{
    Generate initial population $P_t$
    Evaluate population $P_t$
    While stopping criteria not satisfied Repeat
    {
        Select elements from $P_t$ to copy into $P_{t+1}$
        Crossover elements of $P_t$ and put into $P_{t+1}$
        Mutation elements of $P_t$ and put into $P_{t+1}$
        Evaluate new population $P_{t+1}$
        $P_t = P_{t+1}$
    }
}
```

Figure (1): Basic steps of a simple GA

A more comprehensive overview of GAs can be found in (S.N.Sivanandam, S.N.Deepa (2008) [20], Mitsuo Gen, Runwei Cheng (1999) [21], Michalewicz, Z., (1996) [22], Melanie Mitchell (1996) [23], Glover, F. and G. Kochenberger (2003) [24]).

3. Handling Constraints with Genetic Algorithms

A variety of constraint handling methods have been suggested by many researchers [25]. The most popular constraint handling method among users is penalty function methods [26]. The idea behind penalty-based methods is to transform a constrained optimization problem into an unconstrained one by adding (or subtracting) a certain value to/from the

objective function based on the amount of constraint violation present in a certain solution.
In classical optimization, two kinds of penalty functions are considered: exterior and interior. In the case of exterior methods, we start with an infeasible solution and from there we move towards the feasible region. In the case of interior methods, the penalty term is chosen such that its value will be small at points away from the constraint boundaries and will tend to infinity as the constraint boundaries are approached. Then, if we start from a feasible point, the subsequent points generated will always lie within the feasible region since the constraint boundaries act as barriers during the optimization process.
There are several Methods proposed in GAs to handle constrained optimization problems based on penalty functions (Death Penalty - Static Penalties - Dynamic Penalties - Annealing Penalties - Adaptive Penalties - Segregated GA - Co-evolutionary Penalties) [27, 28]. The basic approach in penalty functions is to define the fitness value of an individual i by extending the domain of the objective function $f(S)$ using two **ways**:
1. $Fitn(S) = f(S) + Pen1(S)$.
2. $Fitn(S) = f(S) \times Pen2(S)$.
Where $Pen1(S)$ and $Pen2(S)$ represent a penalty for an infeasible individual. It is assumed that if S is feasible then $Pen1 = 0$ and $Pen2 = 1$ (i.e. we do not penalize feasible individual).

4. A New Genetic Algorithm (NGA) for the MMKP

In this section we describe the NGA algorithm; this algorithm hybridize the solution construction mechanism of GA operator (hybrid selection operator, hybrid cross-over operator and new hybrid mutation operator) and use the Maximizing Value per Resources Consumption (MVRC) heuristic algorithm to generate the initial population of the GA. Our proposed algorithm consists of the following sections.

4.1. Representation and Fitness Function

The first step in designing a genetic algorithm for a particular problem is to devise a suitable representation scheme (Encoding Process), i.e., a way to represent individuals in the GA population [29]. This process can be performed using bits, numbers, trees, arrays, lists or any other objects. The encoding depends mainly on solving the problem, There are many types of representation the problem such that (Binary Encoding, Octal Encoding, Hexadecimal Encoding, Permutation Encoding (Real Number Coding), Value Encoding, Tree Encoding).
Hence, in our representation we encode the problem by the Permutation Encoding which every chromosome is a string of numbers, which represents the index of items that select from each group. This representation of an individual's chromosome (solution) for the MMKP is illustrated in Figure 2. Note that a string $S = \{S_1, S_2, \ldots .S_n\}$, $S_i \in \{1,2,\ldots, n_i\}$, $i \in \{1, \ldots, m\}$, where n_i number of items in group i and m number of group in the knapsack problem.

i	1	2	3	4	5	……	*m*-1	*m*
S[*i*]	2	3	3	1	4	……	5	2

Figure (2): *Example of Permutation Encoding*

This encoding ensures that only one item is selected from each group in the knapsack problem. Thus, the whole of covering constraint set (3) of the MMKP formulation in section 1 is implicitly fulfilled by this encoding.
The objective function is directly chosen as the fitness function. Since the objective function is to be maximized, the larger the fitness function, the better the chromosome is. The population might have infeasible individuals, so the fitness function of infeasible individuals must be penalized in some way [30]. So that we use penalty value (*Pen)* that discussed in section 3 to handle constraint set (2) of the MMKP formulation in section 1, therefore a fitness function is defined as **follows**:

$$Fitness(S) = \sum_{i=1}^{m}\sum_{j=1}^{n_i} \left(p_{ij}\, x_{ij} + Pen\left(Min\left(c_k - w_{ijk}\, x_{ij},\ 0\right)\right)\right),\quad k \in \{1, \dots, l\} \tag{5}$$

4.2 Generation of the Initial Population:

While genetic algorithms are generally started with an initial population that is generated randomly, some research has been conducted into using special techniques to produce a higher quality initial population. Such an approach is designed to give the GA a good start and speed up the evolutionary process [31].
In this section, instead of selecting the Initial Population according to randomly generated, we present a strong initial population is created by the heuristic algorithm Maximizing Value per Resources Consumption (MVRC). The initial population is produced, a heuristic is used to creating the initial population based on producing an initial solution of (MVRC) algorithm [10] by selecting the item with the highest V-ARC of each group. Considering item *ij* with value P_{ij} , resources usage $W_{ij} = (w_{ij1},\dots, w_{ijk})$, Aggregate Resources Consumption (ARC_{ij}). We define the Value-per unit of V-$ARC_{ij} = P_{ij} / ARC_{ij}$, where the Aggregate Resources Consumption (ARC) is:

$$ARC_{ij} = \frac{W_{ij1} * C_1 + \dots + W_{ijk} * C_k}{\sqrt{C_1^2 + \dots + C_k^2}} \tag{6}$$

We generate the initial Population Size chromosomes by Algorithm1, which illustrated in Figure 3.

```
Algorithm 1:
For Individual = 1 To PopulationSize        /* Generate Individuals */
     For Group = 1 To NGroup          /* Generate items in each Group , NGroup is
                                           number of groups in knapsack */
     Do
          Ck1 = Int((NofItem) * Rnd) + 1
          Ck2 = Int((NofItem) * Rnd) + 1
      Loop Until Ck1 <> Ck2
      If V-ARC(Group, Ck1) > V-ARC (Group, Ck2) Then CK = Ck1 Else CK = Ck2
      Items(Individual, Group) = CK         /* Select item with the highest V-ARC */
      Next Group
Next Individual
```

Figure (3): *MVRC Algorithm to Generate the Initial Population*

4.3 Parent Selection

The effect of selection is to return a probabilistically selected parent. Although this selection procedure is stochastic, it does not imply GA employ a directionless search. The chance of each parent being selected is in some way related to its fitness. Parent selection is the task of assigning reproductive opportunities to each individual in the population. Typically in a GA we need to generate two parents who will have (one or two) children. In this section we present a Hybrid selection method.

4.3.1. Hybrid Selection Method

The hybrid selection method consists of the combination of tow type of selection methods or more [32], here we use both Roulette wheel Selection Method (RWSM) and Tournament Selection Method (TSM). We designed these types of hybrid selections, 50% of the population size adopts RSM procedure where as the RWSM procedure is used in the remaining 50% of the population size. In other words, the offspring is generated using these procedures, 50% using TSM and another 50% using RWSM.

4.3.2. Roulette Wheel Selection Method (RWSM)

Each individual in the population is assigned a space on the roulette wheel, which is proportional to the individual relative fitness. Individuals with the largest portion on the wheel have the greatest probability to be selected as parent generation for the next generation.

4.3.3. Tournament Selection Method (TSM)

In tournament selection, a number Tour of individuals is chosen randomly from the population and the best individual from this group is selected as a parent. This process is repeated as often as individuals to choose. These selected parents produce uniform offspring at random. The parameter for tournament selection is the tournament size Tour. Tour takes values ranging from 2 – Nind (number of individuals in population).

4.4 Crossover Operator

Crossover is the main genetic operator and consists of swapping chromosome parts between individuals. Cross-over is not performed on every pair of individuals; its requency being controlled by a crossover probability (*Pc*). There are several crossover methods available and here we use hybrid combination of one-point crossover method, tow-point crossover method, and uniform crossover method [32, 33].

4.4.1 One-Point Crossover

The procedure of one-point crossover is to randomly generate a number (less than or equal to the chromosome length) as the crossover position. Then, keep the bits before the number unchanged and swap the bits after the crossover position between the two parents. See Figure 4.

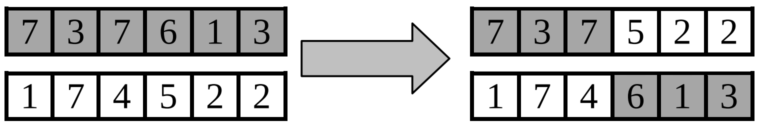


Figure 4: *One-point Crossover*

4.4.2 Tow-Point Crossover

The procedure of two-point crossover is similar to that of one-point crossover except that we must select two positions and only the bits between the two positions are swapped. This crossover method can preserve the first and the last parts of a chromosome and just swap the middle part. See Figure 5.

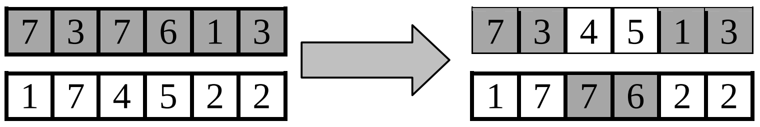
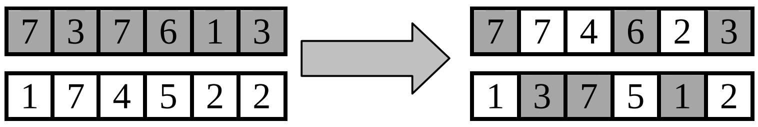


Figure 5: *Tow-point Crossover*

4.4.2 Uniform Crossover

The procedure of uniform crossover: each gene of the first parent has a 0.5 probability of swapping with the corresponding gene of the second parent. See Figure 6.



Random Number (**0.2**, 0.7, 0.9, **0.4**, 0.6, **0.1**)

Figure 6: *Uniform Crossover*

4.5 Mutation Operator

After crossover, the strings are subjected to mutation. Mutation helps escape from local minima's trap and maintains diversity in the population. It also keeps the gene pool well stocked, and thus ensuring ergodicity. Mutation is not performed on each individual; its requency being controlled by a mutation probability (*Pm*). There are several mutation methods available and here we use hybrid combination of insert Mutation method, Inverse Mutation method, Swap mutation method and Reversing Mutation method.

4.5.1. Swap Mutation

Two random positions of the string are chosen and the bits corresponding to those positions are interchanged. This is shown in Figure 7.

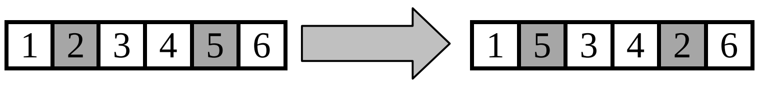


Figure 7: *Swap mutation*

4.5.2. Reversing Mutation

A random position is chosen and the bits next to that position are reversed and child chromosome is produced. This is shown in Figure 8.

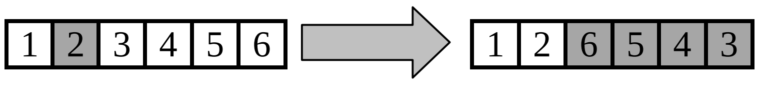


Figure 7*:* *Reversing mutation*

4.5.3. Insert Mutation

Two random positions of the string are chosen, and then move the second to follow the first and shifting the rest to accommodate. This is shown in Figure 9.

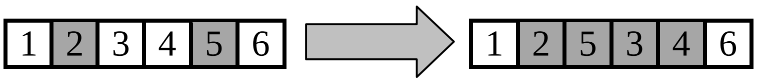


Figure 9: *Insert mutation*

4.5.4. Inversion Mutation

Two random positions of the string are chosen, and then invert the substring between them. This is shown in Figure 10.

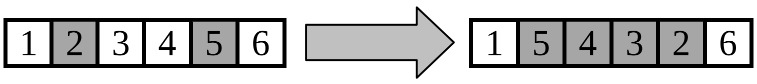


Figure 10: *Inversion mutation*

F. Djannaty and S. Doostdar [31] present A novel mutation operator is used which is a kind of variable mutation rate. the following mutation rate was:
Mutation rate = (0.06 * (Iter / PopSize) 10) + 0.01 (7)
Where Iter denotes the current iteration number and PopSize is the number of individuals in the initial population.

4.6 Replacement Techniques.

Once the new offspring solutions are created using crossover and mutation, we need to introduce them into the parental population. There are many ways we can approach this. Bear in mind that the parent chromosomes have already been selected according to their fitness, so we are hoping that the children (which includes parents which did not undergo crossover) are among the fittest in the population and so we would hope that the population will gradually, on average, increase its fitness. Some of the most common replacement techniques are outlined below.

4.6.1 Delete-All Technique

This technique deletes all the members of the current population and replaces them with the same number of chromosomes that have just been created. This is probably the most common technique and will be the technique of choice for most people due to its relative ease of implementation. It is also parameter-free, which is not the case for some other methods.

4.6.2 Steady-State Technique.

This technique deletes n old members and replaces them with n new members. The number to delete and replace, n, at any one time is a parameter to this deletion technique. Another consideration for this technique is deciding which members to delete from the current population. Do you delete the worst individuals, pick them at random or delete the chromos omes that you used as parents? Again, this is a parameter to this technique.

4.6.3 Steady-State-No-Duplicates Technique.

This is the same as the steady-state technique but the algorithm checks that no duplicate chromosomes are added to the population. This adds to the computational overhead but can mean that more of the search space is explored.
Here we use the steady-state-no-duplicates technique to replace the population with keep 50% best individual to next generation

4.7 Stopping Criterion

There are several Stopping Criterion methods available (Maximum generations, Elapsed time, No change in fitness,….) and here we use Maximum generations to stop. i.e If the algorithm is executed to the maximal number of generations MAXGEN, then stop. The best chromosome found in the last population is then taken as the approximate global optimal solution. Here we use (No change in fitness) to stop the program

4.8. Algorithmic Outline

The outline of the proposed algorithm NGA for solving MMKP problem is shown in Algorithm 3. See Figure 11

5. Computational Results

In this section, we present computational results of our proposed algorithm, which was coded in the visual basic and executed on a computer P4 Intel Core 2 Duo E7300 @ 2.66 GHz CPU Speed and 3GB RAM. To evaluate validity of our proposed algorithm for the MMKP problem, the performance of our algorithms was tested on benchmark problems.
The benchmark problems are thirteen test problems labeled I01 to I13 and generated by Khan et al. [1]. Benchmark problems can be downloaded from the OR Library at http://www.laria.u-picardie.fr/hifi/OR-Benchmark/MMKP/ [34].
Many parameters exist in the NGA algorithm, and the values of these affect directly or indirectly the final solution quality. The Initial parameters setting for the NGA are shown in Table 1.

```
Algorithm 3. A NGA for the MMKP
0: Data. Choose the Initial parameters setting for the NGA.
1: Set $Gen = 1$;  /* Iteration counter */
2: Generate initialize population by Algorithm 1
3: Evaluate $P(Gen) = \{ fitn(S_1), \ldots, fitn(S_n)\}$; /* Use penalty function */
4: Find $S^* \in P(Gen)$ s.t $f(S^*) \geq f(S), \forall S \in P(Gen)$ and $S^*$ is feasible
5: While (gen < MAXGEN ) /* MAXGEN is Maximum number of generations */
6: Select {P1; P2} = $\phi$ P(Gen);  /* $\phi$= Hybrid RWSM and TSM */
7: Crossover $C = \Omega c (P1; P2)$;  /* $\Omega c$ = Hybrid one-point, tow-point, and uniform crossover */
8: Mutate $C \leftarrow \Omega m$; /* $\Omega m$ = Hybrid insert, inverse, swap and reversing Mutation */
9: Evaluate $f(c)$ ;  /* Use penalty function */
10: Find $S` \in P(Gen)$ s.t $f(S`) \geq f(S), \forall S \in P(Gen)$ and $S`$ is feasible
11: Replace $S` \leftarrow C$; /* Steady-state-no-duplicates and Keep 50% best individuals to next generation */
12: if  $f(C) \geq f(S^*)$ then $S^* \leftarrow C$; Endif /* update best solution $S^*$ found */
13: $Gen = Gen + 1$;
14: End While
15: Return $S^*, f(S^*)$.
```

***Figure 11:** The outline of the proposed algorithm NGA for solving MMKP*

***Table (1):** Initial parameter setting for the NGA*

No	Parameter / Strategy	Initial Setting
1	Number of generation	2000
2	Population Size	The population size varied for all 13 problem instances. (100-1000)
3	Initiation Population	Initial population is created by (MVRC).
4	Selection Type	Hybrid combination. (50% using RWSM and 50% using TSM)
5	Crossover Type	Hybrid combination (one-point crossover, tow-point crossover, and uniform crossover) with probability 1/3 for each.
6	Crossover Probability	0.9
7	Mutation Type	Hybrid combination (insert mutation, inverse mutation, swap mutation and reversing mutation) with probability 0.25 for each.
8	Mutation Probability	Mutation probability = (0.06 * (Iter / PopSize) 10) + 0.01
9	Replacement Strategy	Steady-state-no-duplicates technique with keep 50% best individual to next generation
10	Stopping Criteria	No improvement for 100 generations
11	Penalty weight	10

To save a lot of memory space and reduce runtime, first, We performed run for each problem with 10 sample population size (100-200-300-400-500-600-700-800-1000) to determine what the suitable population size for each problem is.
Second, We performed 10 independent runs for each benchmark problem. The statistical results of NGA are summarized in Table 2.
We compare NGA with a number of the better methods available for the (MMKP), such as MOSER algorithm [2], HEU algorithm [1], CPCCP and DerAlgo algorithms [5], MRLS algorithm [6], greedy heuristic approaches (CH1, and CH2) [12], meta-heuristic approaches (FLTS, FanTabu, and CCFT) [12], ACO and ACO&PR [16]. The results of comparison are described in Table 3.
Figure 12 shows the Comparison of NGA with other Heuristics based on Number of Problems Solved on the MMKP Test Sets and Figure 13 summarizes the average relative percentage error of NGA and other Heuristics for the MMKP Test Sets
The results show that the proposed NGA has a higher solution quality and lower relative percentage error in comparison to the other heuristics. NGA reaches the optimal solution more often and provides better performance than other heuristics.

6. Conclusions:

In this paper, we propose an NGA algorithm, this algorithm hybridize the solution construction mechanism of GA operator (hybrid selection operator, hybrid cross-over operator and new hybrid mutation operator) and use the heuristic algorithm (MVRC) to generate the initial population of the GA. The experimental results show that the proposed NGA has a higher solution quality and lower relative percentage error in comparison to the other heuristics. NGA reaches the optimal solution more often and provides better performance than other heuristics.

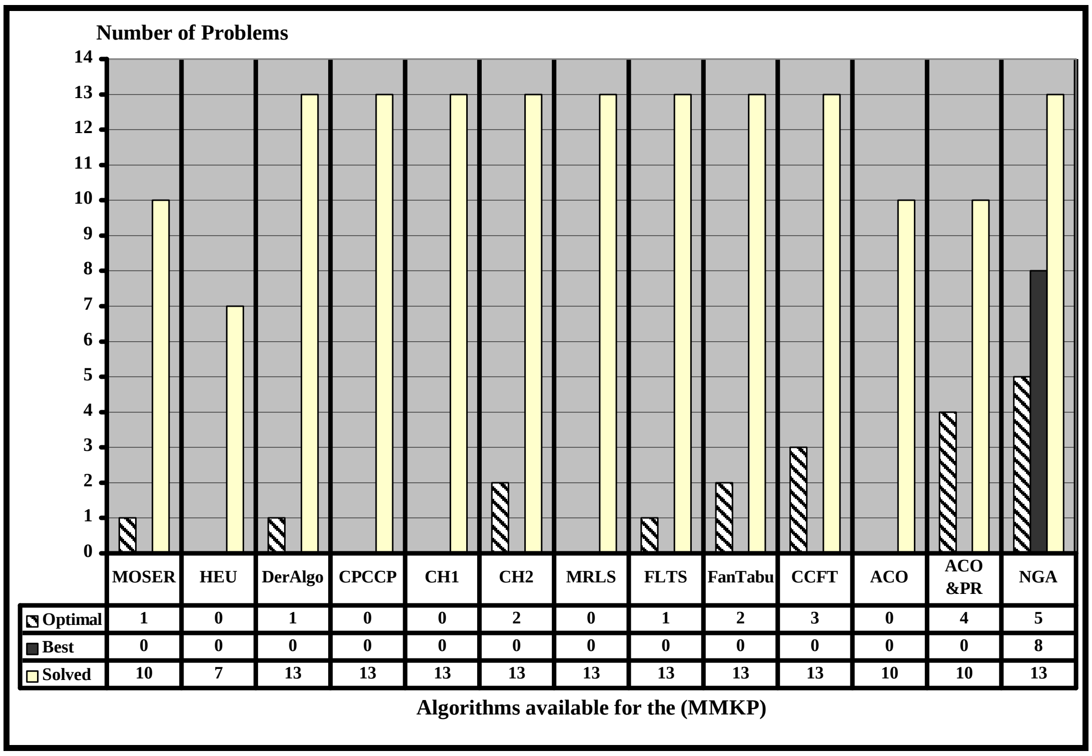
***Table (2):** Solution Quality for NGA on MMKP Problems*

Problem File	Problem size (m,n,l)	population size	NGA Solution	Mean Fitness	Stander Deviation	Optimality Percent	Relative Error	CPU Time	Generation Number	Memory Usage
I01	(5,5,5)	100	173	159.2	13.9	100%	0%	0.63	21	12012
I02	(10,5,5)	100	364	339.7	24.35	100%	0%	1.015	79	12020
I03	(15,10,10)	600	1602	1563	31.99	100%	0%	94	1104	12104
I04	(20,10,10)	500	3591	3541.5	33.55	99.8%	0.2%	48	646	12428
I05	(25,10,10)	100	3905.7	3797.4	108.8	100%	0%	0.594	50	12040
I06	(30,10,10)	100	4799.3	4665.2	134.6	100%	0%	1.75	125	12096
I07	(100,10,10)	800	24443	24318.1	114.9	99.4%	0.6%	478	1128	16076
I08	(150,10,10)	800	36605	36408.2	192.9	99.3%	0.7%	604	960	16240
I09	(200,10,10)	900	48747	48473.5	255.6	99.1%	0.9%	991	1080	18204
I10	(250,10,10)	1000	60787	60173.9	613.6	98.9%	1.1%	1534	1183	19168
I11	(300,10,10)	1000	73210	72685.9	521.4	99.2%	0.8%	2873	1771	19204
I12	(350,10,10)	1000	85216	84758.3	454.95	99 %	1.0%	3108	1653	21664
I13	(400,10,10)	1000	97500	96680	820.68	99 %	1.2%	3996	1855	21708

Memory Usage in KB , CPU Time in Second
Optimality Percent = (NGA Solution / Exact Solution) * 100
Relative Error = ((Exact Solution - NGA Solution) / Exact Solution) * 100
(m,n,l) represents (groups, items, resources) in knapsack problems
Exact Solution according to (Hiremath [12])

***Table (3):** Compare NGA with other Algorithms*

Problem File	Exact Solution	MOSER	HEU	DerAlgo	CPCCP	CH1	CH2	MRLS	FLTS	FanTabu	CCFT	ACO	ACO &PR	NGA
I01	173	151	154	159	159	167	173	161	158	169	173	169	173	173
I02	364	291	354	312	312	332	364	354	351	354	352	353	364	364
I03	1602	1464	1518	1432	1407	1509	1572	1496	1445	1557	1518	1521	1556	1602
I04	3597	3375	3297	3322	3322	3369	3461	3435	3350	3473	3419	3383.2	3452	3590
I05	3905.7	3905.7	3894.5	3905.7	3889.9	3905	3905	3847.3	3905.7	3905.7	3905.7	3800.1	3905.7	3905.7
I06	4799.3	4115.2	4788.2	4723.1	4723.1	4689	4799	4680.6	4793.2	4799.3	4799.3	4723.4	4799.3	4799.3
I07	24587	23556	-	23480	23237	23529	23711	23828	23547	23691	23739	23600	23938.2	24443
I08	36877	35375	34338	35525	35403	35691	35816	35685	35487	35684	35698	35405	35997	36605
I09	49167	47205	-	47471	47154	47687	47647	47574	47107	47202	47491	47225	47928	48747
I10	61437	58648	-	59039	58990	59703	59351	59361	59108	58964	59549	58824.3	59846	60787
I11	73773	-	-	71018	70685	71761	71405	71565	70549	70555	71651	-	-	73210
I12	86071	-	-	83154	82754	83701	83174	83314	82114	81833	83358	-	-	85216
I13	98429	-	-	94628	84465	95432	94934	95076	91551	94168	94874	-	-	97500



	MOSER	HEU	DerAlgo	CPCCP	CH1	CH2	MRLS	FLTS	FanTabu	CCFT	ACO	ACO &PR	NGA
▧ Optimal	1	0	1	0	0	2	0	1	2	3	0	4	5
■ Best	0	0	0	0	0	0	0	0	0	0	0	0	8
□ Solved	10	7	13	13	13	13	13	13	13	13	10	10	13

Figure 12: Comparison of NGA based on Number of Problems Solved on MMKP Test Problems

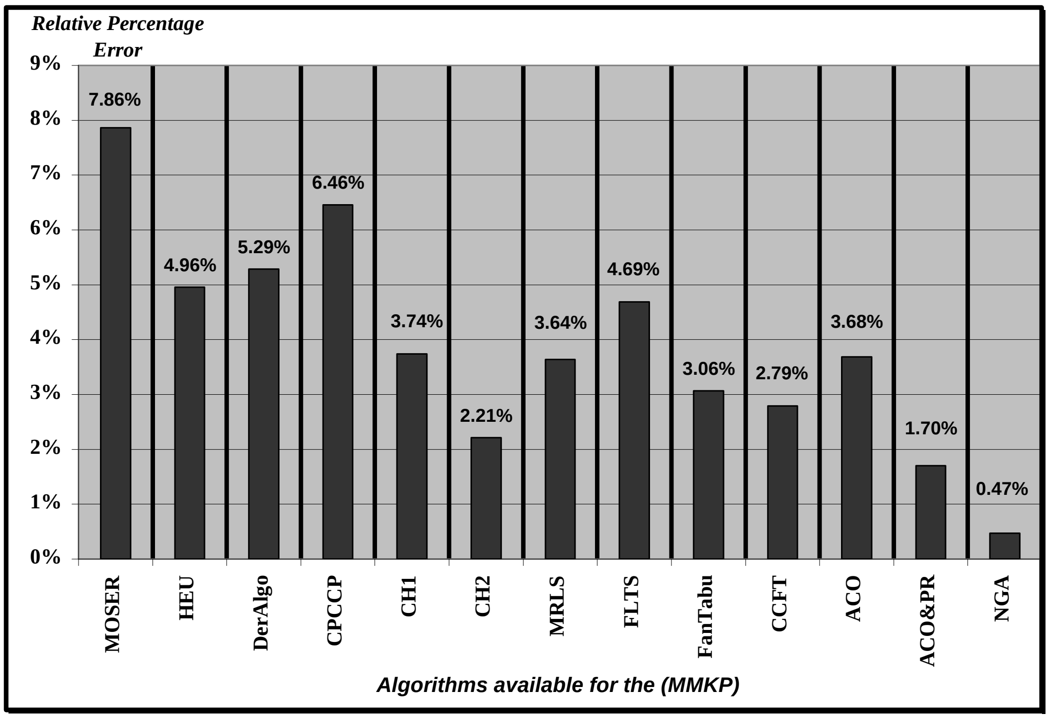


Figure 13: *Percentage Relative Error of NGA and other Heuristics for the MMKP Test Sets*

References:

[1] S. Khan, K. F. Li, E. G. Manning, M. M. Akbar (2002), *Solving the knapsack problem for adaptive multimedia systems*, Studia Informatica, Special Issue on Combinatorial Problems 2(2), 157–178.

[2] M. Moser, D. P. Jokanovi_c, N. Shiratori, *An algorithm for the multidimensional multiple-choice knapsack problem*, IEICE transactions on fundamentals of electronics, communications and computer sciences 80 (3) (1997) 582-589.

[3] S. Khan, *Quality adaptation in a multisession multimedia system: Model, algorithms and architecture*, Ph.D. thesis, University of Victoria (1998).

[4] Akbar, M. M., O. Ergun, and A. P. Punnen (2001), *Heuristic solutions for the multiple-choice multidimensional knapsack problem*. In International Conference on Computer Science, San Francisco, USA.

[5] M. Hifi, M. Michrafy, A. Sbihi, *Heuristic algorithms for the multiple-choice multidimensional knapsack problem*, J Operat Res Soc 55 (12) (2004) 1323-1332.

[6] M. Hifi, M. Michrafy, A. Sbihi, *A reactive local search-based algorithm for the multiple-choice multi-dimensional knapsack problem*, Computational Optimization and Applications 33 (2-3) (2006) 271-285.

[7] Parra-Hernandez, R. and N. Dimopoulos (2005). *A new heuristic for solving the multi-choice multidimensional knapsack problem*. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans 35(5), 708–717.

[8] M. M. Akbar, M. S. Rahman, M. Kaykobad, E. G. Manning, G. C. Shoja, *Solving the multidimensional multiple-choice knapsack problem by constructing convex hulls*, Comput. Oper. Res. 33 (5) (2006) 1259-1273.

[9] N. Cherfi, M. Hifi, *A column generation method for the multiple-choice multidimensional knapsack problem*, Comput. Optim. App. online first, Springer Netherlands (2008).

[10] M. Chantzara, M. E. Anagnostou, *MVRC heuristic for solving the multi-choice multiconstraint knapsack problem*, in: International Conference on Computational Science (1), 2006, pp. 579-587.

[11] A.Z.M. Shahriar, M.M. Akbar, M.S. Rahman and M.A.H. Newton. *A multiprocessor based heuristic for multi-dimensional multiple-choice knapsack problem*. Journal of Supercomput (2008) 43: 257–280

[12] Hiremath, Chaitr, *New Heuristic and Metaheuristic Approaches Applied to The Multiple-choice Multidimensional Knapsack Problem*. Engineering PhD, 2008, Wright State University.

[13] Li, V. C. and G. L. Curry (2005). *Solving multidimensional knapsack problems with generalized upper bound constraints using critical event tabu search*. Computers and Operations Research 32(4), 825–848.

[14] http://shah.freeshell.org/samultichoiceknapsack/

[15] Shubhashis K. Shil, A. B. M. Sarowar Sattar, Md. Waselul Haque Sadid, Dr. Md. Shahid Uz Zaman, *Solving Multidimensional Multiple Choice Knapsack Problem by Genetic Algorithm & Measuring it's Performance*, International Conference on Electronics, Computer and Communication (ICECC 2008). University of Rajshahi, Bangladesh

[16] Xiaoxia Zhang and Lixin Tang, *A new ACO&PR algorithm for multiple-choice multidimensional knapsack problem*. J Control and decision Vol. 24 No.5 May 2009
[17] A. Sbihi, *A best first search exact algorithm for the multiple-choice multidimensional knapsack problem*, Journal of Combinatorial Optimization 13 (4) (2007) 337-351.
[18] M.R. Razzazi and T. Ghasemi, *An Exact Algorithm for the Multiple-Choice Multidimensional Knapsack Based on the Core*, Springer-Verlag Berlin Heidelberg 2008, CCIS 6, pp. 275–282.
[19] Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press.
[20] S.N.Sivanandam, S.N.Deepa., *Introduction to Genetic Algorithms*, Springer-Verlag, New York, 2008.
[21] Mitsuo Gen, Runwei Cheng, *Genetic Algorithms and Engineering Optimization*, John Wiley and Sons, Inc, New York, 1999.
[22] Michalewicz, Z., *Genetic Algorithm + Data Structure = Evolution Programs*, 3rd edition, Springer-Verlag, New York, 1996.
[23] Melanie Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, 1996.
[24] Glover, F. and G. Kochenberger. (2003) *Handbook of Metaheuristics*. kluwer Academinc Publisher
[25] Michalewicz, Z. and Schoenauer, M. (1996). *Evolutionary Algorithms for Constrained Parameter Optimization Problems*. Evolutionary Computation, 4(1):1–32.
[26] Coello, C.A.C. (2002). *Theoretical and Numerical Constraint Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art*. Computer Methods in Applied Mechanics and Engineering, 191(11-12):1245–1287, January.
[27] Coello, C.A.C., Cruz-Cort´es, N, *Hybridizing a Genetic Algorithm with an Artificial Immune System for Global Optimization*. Engineering Optimization 36(5), 607–634 (2004)
[28] Özgür Yeniay, *Penalty Function Methods for Constrained Optimization with Genetic Algorithms*, Mathematical and Computational Applications, Vol. 10, No. 1, pp. 45-56, 2005.
[29] Chu, P.C. and J.E. Beasley. (1998) " *A Genetic Algorithm for the Multidimensional Knapsack Problem*," Journal of Heuristic 4, 63-86.
[30] Hoff, A., et al, (1996). " *Genetic Algorithms for 0/1 Multidimensional Knapsack Problems*." Working Paper, Molde College, Britveien 2, 6400 Molde, Norway.
[31] F. Djannaty and S. Doostdar. " *A Hybrid Genetic Algorithm for the Multidimensional Knapsack Problem*," Int. J. Contemp. Math. Sciences, Vol. 3, 2008, no. 9, 443 - 456.
[32] M. S. Arumugam, M.V.C. Rao and R. Palaniappan, *New hybrid genetic operators for real coded genetic algorithm to compute optimal control of a class of hybrid systems*, Journal of Applied Soft Computing 6 (2005) 38–52.
[33] T. P. Hong, H.S.Wang, and T. T.Y. Juang. " *A Dynamic Crossover Genetic Algorithm*," Proceedings of National Computer Symposium 1995, 596 - 602.
[34] Hifi, M. (2006). MMKP problems available at the website. http://www.laria.u-picardie.fr/hifi/OR-Benchmark/MMKP/.