Mathematical and Computational Applications, Vol. 18, No. 3, pp. 486-494, 2013

A GENETIC ALGORITHM TO SOLVE THE MULTIDIMENSIONAL KNAPSACK PROBLEM

Murat Ersen Berberler[1], Asli Guler[2,*] and Urfat G. Nurıyev[3]

[1]Department of Computer Science, Dokuz Eylul University, Izmir, Turkey
[2]Department of Mathematics, Yasar University, Izmir, Turkey
[3]Department of Mathematics, Ege University, Izmir, Turkey
[1]murat.berberler@deu.edu.tr [2]asli.guler@yasar.edu.tr [3]urfat.nuriyev@ege.edu.tr

Abstract- In this paper, The Multidimensional Knapsack Problem (MKP) which occurs in many different applications is studied and a genetic algorithm to solve the MKP is proposed. Unlike the technique of the classical genetic algorithm, initial population is not randomly generated in the proposed algorithm, thus the solution space is scanned more efficiently. Moreover, the algorithm is written in C programming language and is tested on randomly generated instances. It is seen that the algorithm yields optimal solutions for all instances.

Key Words- Multidimensional Knapsack Problem, Genetic Algorithm, Heuristic Approach, Evolutionary Algorithms

1. INTRODUCTION

Knapsack problems have been intensively studied recently due to its simple structure and the more complex problems can be solved through knapsack problems. The problems such as capital budgeting, cargo loading and project selection problem can be modeled by knapsack problems [4]. The multidimensional knapsack problem (MKP) is special case of the classical 0-1 knapsack problem, and it has more than one constraint. The MKP is a well-studied, NP-hard combinatorial optimization problem occurring in many different applications and there is no FPTAS for two dimensional knapsack problem unless P=NP, [9].

The MKP can be stated as follows:

Consider a set of projects (j = 1,...,n) and a set of resources (i = 1,...,m). Each project has assigned a profit $p_j > 0$ and resource consumption values $w_{ij} > 0$. The problem is to find a subset of all projects that leads to the maximum possible profit and not exceeding given resource limits c_i [13].

It is seen that there are more constraints unlike the general KP. The problem cen be defined by the following integer linear programming:

$$\text{maximize} \quad \sum_{j=1}^{n} p_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_{ij} x_j \le c_i, \qquad i = 1,...m,$$

$$x_j \in \{0,1\}, \qquad j = 1,...,n.$$

Here,

p_j : profit of project *j*,

w_{ij} : consumption of project *j* from resource *i*,

c_i : capacity of resource *i*,

$$x_j = \begin{cases} 1, & \text{if project } j \text{ is selected,} \\ 0, & \text{otherwise.} \end{cases}$$

It is assumed, without loss of generality, that p_j, w_{ij} and c_i are pozitive integers, besides

$$w_{ij} \le c_i, \; j = 1,...,n$$

$$\sum_{j=1}^{n} w_{ij} \ge c_i, \quad i = 1,...,m \; .$$

MKP is a particular difficult problem of integer programming since the constraint matrix consisting of w_{ij} is dense. On the other hand, there is already a feasible solution at hand for MKP, namely $x_j = 0, \; j = 1,...,n$, whereas finding a feasible solution can be as hard as finding an optimal solution in general integer programming [11].

The first examples have been exhibited by Lorie and Savage and by Manne and Markowitz as a capital budgeting model. There is a comprehensive overview of the results for the MKP by Kellerer et al. [9]. A recent review of the MKP was given by Fr´eville [3]. Besides the method currently yielding the best results, at least for commonly used benchmark instances, was described by Vasquez and Hao [16] and has recently been refined by Vasquez and Vimont [17]. It is a hybrid approach based on tabu search. Moreover, there are studies of Gilmore and Gomory [9]; Weingartner and Ness [9]; Shih [9]; Gavish and Pirkul [5]; Glover and Kochenberger [9]; Chu and Beasley [2], Raidl and Gottlieb [7, 14] and Puchinger et al. [12] in the litarature.

2. GENETIC ALGORITHMS FOR MKP

Genetic Algorithms (GA), which find application in bioinformatics, phylogenetics, computational science, engineering, economics, chemistry, manufacturing, mathematics, physics, pharmacometrics and other fields are search algorithms based on natural selection and genetics. These algorithms belong to the larger class of evolutionary algorithms (EA), that generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. It can be said that the strongest individuals in a population will have a better chance to transfer their genes to the next generation.

In a genetic algorithm, a population of candidate solutions to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.

The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated, the more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population, [6].

The reproduction can be done in three ways :

- *Pure Reproduction* - The individual is copied directly into the next generation
- *Crossover* - Two individuals are selected and their genes are crossed at some point, as the first part of the new individual comes from one parent and the last part
  from the other.
- *Mutation* - An individual is selected, and one bit is changed.

Evolutionary algorithms is an important subject of metaheuristics. The early papers have not successfully proved that genetic algorithms were an effective heuristic tool for the MKP. Khuri *et al.* [10] extended previous work for the single constraint knapsack problem. A similar study is given in Battiti and Tecchioli [1]. Thiel and Voss showed that a standard GA using a direct search in the complete search space is not able to obtain good solutions for the MKP, except for small problems [15]. Moreover, they investigated the combination of GA with tabu search and obtained promising results. Chu and Beasley gave the first successful implementation of GA's by restricting the genetic algorithms to search only the feasible search space. Finally, Haul and Voβ enhanced the performance of GA's by using surrogate constraints [8].

3. A NEW GENETIC ALGORITHM FOR MKP

The steps of the algorithm are as follows:

[GA1] Each of *m* constraints is handled seperately and its optimal solution is found by dynamic programming method. The total frequencies of occurrence of items that are located in the solution vectors are found, then they are sorted in descending order and index sequence I is obtained.

[GA2] The first *n* elements of the initial population are established in a way that the item concerning the current index is taken as long as it does not exceed knapsack capacities starting with the i^{th} element of index sequence I ($1 \le i \le m$) at each step.

[GA3] Each of m constraints is handled seperately and p_j / w_{ij} , $(1 \le i \le m)$, values are calculated. The relaxed solutions of each constraint are found, then index sequence J is obtained by sorting the frequencies of entering the solution of each item in descending order.

[GA4] The other n elements of the initial population are established in a way that the item concerning the current index is taken as long as it does not exceed knapsack capacities starting with the j^{th} element of index sequence J $(1 \le j \le n)$.

[GA5] The coefficients of the objective function, p_j , are sorted in descending order and index sequence K is obtained.

[GA6] Each individual of the population consisting of 2*n elements is crossed with all other individuals. If there is an item which can be taken for the generated individual, the item concerning the current index is taken as long as it does not exceed knapsack capacities starting with the first element of index sequence K$(1 \le k \le n)$. The individual that has the maximum value of the objective function in the population is assigned as the record.

[GA7] Step [GA6] is repeated until the iteration number is *n.*

[GA8] The record is written and the algorithm ends.

Unlike the technique of the classical genetic algorithm, initial population is not randomly generated in this algorithm through the steps [GA1]...[GA4], thus the solution space is scanned much more efficiently.

4. COMPUTATIONAL EXPERIMENTS

Computational experiments have been carried out generating random problems for $1 \le w_{ij} \le 100$, $1 \le p_j \le 100$, m:10,20,…,100 and n:10,20,…,100. In all instances, the capacity of each knapsack (c_i) in each constraint is obtained by taking 25 percent off total weight of the items.

The optimal values of the problems have been found by GAMS IDE and shown in Table 1.

Table 1. Optimal values of the problems found by GAMS IDE

m \ n	10	20	30	40	50	60	70	80	90	100
10	95	339	484	809	991	1297	1478	1705	1788	2241
20	70	289	530	802	953	1231	1325	1660	1838	2011
30	86	303	506	735	904	1139	1420	1618	1803	1911
40	90	262	543	710	867	1042	1310	1616	1815	2067
50	93	280	443	688	859	1098	1289	1522	1741	1973
60	59	285	489	567	872	1137	1288	1532	1692	1970
70	99	270	439	729	902	1056	1321	1503	1786	1867
80	80	278	445	701	888	1050	1373	1481	1594	1961
90	80	259	476	591	864	1058	1260	1454	1738	1905
100	98	216	479	689	882	1028	1193	1519	1645	1932

The algorithm has been written in C language and it has been observed that the proposed algorithm yields optimal results when it is run for 100 problems. The solution times are given in Table 2.

Table 2. The solution times of problems

m \ n	10	20	30	40	50	60	70	80	90	100
10	0,000	0,015	0,125	0,328	0,765	1,609	3,406	5,281	8,968	14,390
20	0,000	0,031	0,156	0,500	1,000	2,281	6,063	7,781	11,718	16,781
30	0,000	0,031	0,203	0,531	1,453	2,765	6,046	9,812	13,671	23,484
40	0,000	0,046	0,218	0,671	1,765	3,171	6,656	10,515	19,625	28,906
50	0,000	0,046	0,250	0,735	2,187	3,812	8,125	11,687	20,812	31,593
60	0,000	0,062	0,281	0,828	2,265	4,531	8,046	15,609	22,500	42,015
70	0,000	0,062	0,296	1,000	2,625	4,937	9,921	19,250	31,687	42,703
80	0,000	0,062	0,343	1,078	2,656	4,859	11,218	21,718	31,203	44,705
90	0,000	0,078	0,359	1,187	2,718	5,546	12,265	21,765	31,468	49,281
100	0,000	0,078	0,375	1,218	3,343	7,281	13,453	23,046	39,125	53,360

The parameters which affect the running time of the algorithm are *m*, *n* and c_i. Figure 1 shows the time increment with respect to parameter *m*, and Figure 2 shows the time increment with respect to parameter *n*.

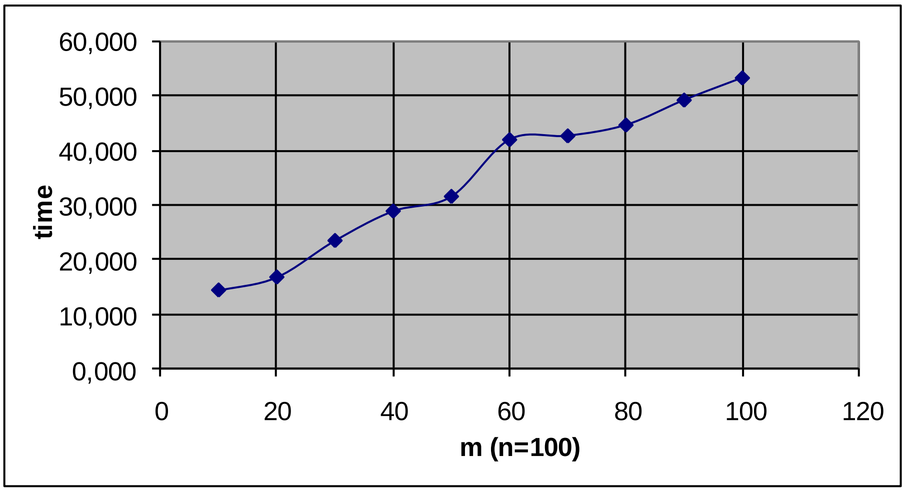

Figure 1. The time increment with respect to parameter *m*

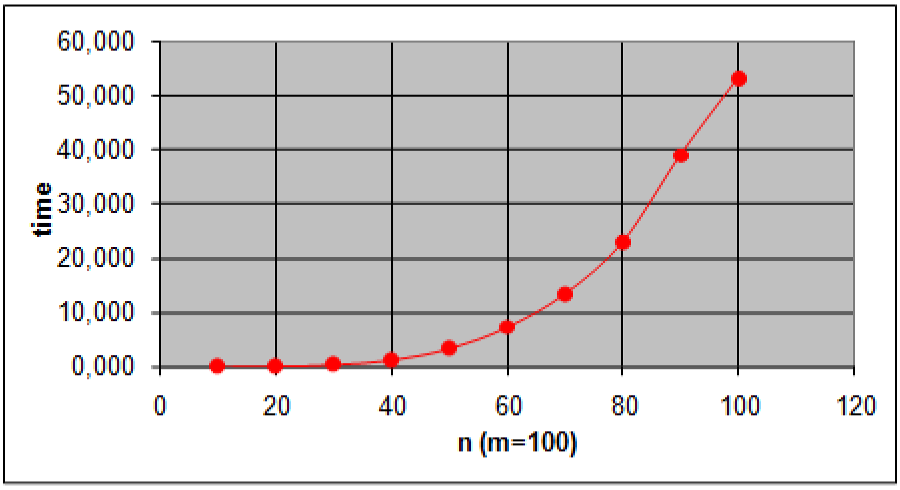

Figure 2. The time increment with respect to parameter *n*

As it is seen in Table 2 and the figures, while parameter *m* affects the running time of the program linearly, parameter *n* affects the time 3rd degree parabolically.

In order to observe how the capacity of the knapsack affects the running time, computational experiments have been carried out for n=100, *m*:10,20,…100. The values of c_i are determined by taking 25 percent, 45 percent, 55 percent, 75 percent off total weight of the items in i^{th} constraint. The optimal values are shown in Table 3, and the running times are given in Table 4 and Figure 3.

Table 3. Optimal values

		n = 100			
		25%	45%	55%	75%
m	10	2241	3597	3460	4991
	20	2011	3243	3992	4811
	30	1911	3219	3888	4601
	40	2067	3426	3720	4401
	50	1973	3366	3971	5072
	60	1970	3339	3990	4867
	70	1867	3257	3993	4691
	80	1961	3089	3902	4285
	90	1905	3358	3758	4461
	100	1932	3280	3696	4807

Table 4. Running times

		n = 100			
		25%	45%	55%	75%
m	10	14,390	15,968	16,109	19,187
	20	16,781	23,343	25,625	30,265
	30	23,484	31,875	34,312	41,546
	40	28,906	38,875	43,187	52,718
	50	31,593	44,093	53,594	64,063
	60	42,015	56,875	62,750	75,234
	70	42,703	59,406	70,360	86,656
	80	44,705	69,046	78,531	98,750
	90	49,281	79,593	83,140	109,796
	100	53,360	78,047	95,250	122,780

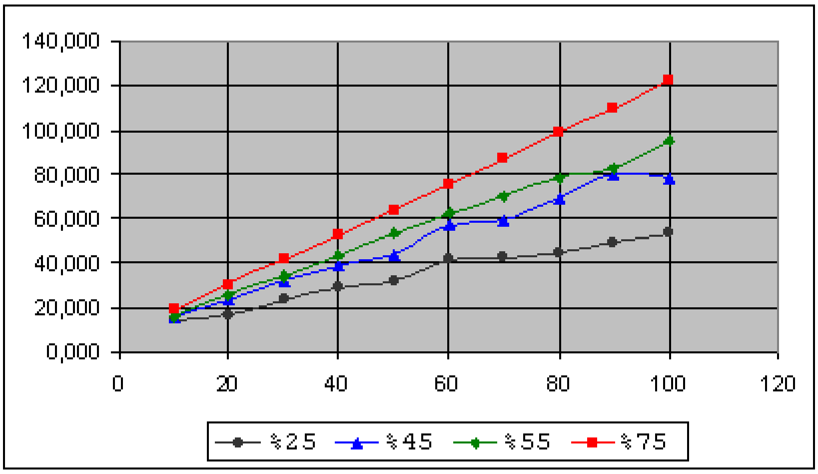

Figure 3. Running times

The properties of the computer that has been used in computational experiments are Intel CORE 2 CPU (2.8 GHz) and 3 GB RAM, besides all problems and source codes are available in the adress http://fen.ege.edu.tr/~murateb/mknapGA/.

5. CONCLUSION

In this paper, The Multidimensional Knapsack Problem (MKP) which occurs in many different applications such as capital budgeting, cargo loading, project selection and which is an NP-hard problem has been studied. A new genetic algorithm to solve the MKP has been proposed. Unlike the technique of the classical genetic algorithm, initial population is not randomly generated in the proposed algorithm, thus the solution space is scanned more efficiently. Moreover, the algorithm is written in C programming language and is tested on randomly generated instances. It is seen that the algorithm yields optimal solutions for all instances. The properties of the computer that has been used in computational experiments are Intel CORE 2 CPU (2.8 GHz) and 3 GB RAM. As it is seen in Table 2 and the figures, while parameter m affects the running time of the program linearly, parameter n affects the time parabolically. Furthermore, problems have been generated in order to observe how the capacity of the knapsack affects the running time and the results have been given in the tables and figures.

6. REFERENCES

1. R Battiti, G. Tecchiolli, Parallel biased search for combinatorial optimization: Genetic algorithms and tabu search, *Microprocessors and Microsystems* **16**, 351–367, 1992.
2. P. C. Chu and J. E. Beasley, A genetic algorithm for the multidimensional knapsack problem**,** *Journal of Heuristics* **4**, 63-86, 1998.
3. A. Freville, The multidimensional 0–1 knapsack problem: An overview, *European Journal of Operational Research* **155,** 1–21, 2004.
4. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 338p, 1979.
5. B. Gavish, H. Pirkul, Efficient Algorithms for Solving Multiconstraint Zero-One Knapsack Problems to Optimality, *Mathematical Programming* **31**, 78–105, 1985.
6. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison–Wesley, 1989.
7. J. Gotlieb, On the effectivity of evolutionary algorithms for multidimensional knapsack problem, *Proceedings of the 4th European Conference of Artificial Evolution*, Dunkerque, France, LNCS no. **1829**, 23–27, 1999.
8. C. Haul, S. Voss, *Using surrogate constraints in genetic algorithms for solving multidimensional knapsack problems*, D.L Woodruff (Ed.), Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search, Kluwer Academic Publishers, pp. 235–251, 1998.
9. H. Kellerer, U. Pferschy, D. Pisinger, *Knapsack Problems*, Springer, Berlin, 546p, 2004.
10. S. Khuri, T. Back, J. Heitkotter, The Zero/One Multiple Knapsack Problem and Genetic Algorithms, *ACM Symposium on Applied Computing*, 188–193, ACM Press, 1994.

11. E. Lin, *A bibliographical survey on some well-known non-standard knapsack problems*, INFOR, 36:274-317, 1998.
12. J. Puchinger, G. R. Raidl, U. Pferschy, The Multidimensional Knapsack Problem: Structure and Algorithms, *INFORMS Journal on Computing* **22:2,** 250–265, 2010.
13. R. Raidl Gunther, An improved genetic algorithm for the multiconstrained 0–1 knapsack problem, D. Fogel, et al., eds., *Proceedings of the 5th IEEE International Conference on Evolutionary Computation*. IEEE Press, 207–211, 1998.
14. R. Raidl Gunther, Jens Gottlieb, Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem, *Evolutionary Computation Journal* **13:4**, 441–475, 2005.
15. J. Thiel, S. Voss, Some experiences on solving multiconstraint zero–one knapsack problems with genetic algorithms, *INFOR* **32**, 226–242, 1994.
16. M. Vasquez, J.-K. Hao, A hybrid approach for the 0–1 multidimensional knapsack problem, *Proceedings of the Int. Joint Conference on Artificial Intelligence*, Seattle, Washington, 328–333, 2001.
17. M. Vasquez, V. Yannick, Improved results on the 0–1 multidimensional knapsack problem, *European Journal of Operational Research* **165,** 70–81, 2005.