A New Genetic Algorithm for the 0-1 Knapsack Problem

[*1]Asli Guler, [2]Murat Ersen Berberler, [3]Urfat G. Nuriyev
[1]Department of Computer Programming, Yasar University, Izmir, Turkey
[2]Department of Computer Science, Dokuz Eylul University, Izmir, Turkey
[3]Department of Mathematics, Ege University, Izmir, Turkey

Received Date : 2016-07-15 Accepted Date: 2016-10-09

Abstract

In this paper, the 0-1 Knapsack Problem (KP) which occurs in many different applications is studied and a new genetic algorithm to solve the KP is proposed. In our methodology, *n* items are represented by *n* genes on a bit array that compactly stores the values 0 or 1. When calculating fitness values of items, coefficients of items whose values are 1 in the bit array are summed. Roulette wheel method is used for choosing parents; in this way it is provided that strong individuals produce more children. Crossover is applied in such a way that roulette wheel is rotated on genes with the same index of parents, that the dominant parent can transfer its genes to the child. Mutation is applied for randomly chosen 25% genes and this process is repeated for the number of individuals. The algorithm is written in C programming language and is tested on randomly generated instances. In order to find the optimal solutions of these problems, a program that has been written by dynamic programming technique is used. It is seen that the algorithm yields optimal solutions for all instances.

Keywords: One-dimensional 0-1 knapsack problem, Genetic algorithm, Heuristic approach, Dynamic programming.

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 [1]. The Knapsack Problem (KP) is a well-studied, NP-complete combinatorial optimization problem occurring in many different applications. The KP cannot be solved in linear amount of time however its solution can be verified in linear time [2].

The problem has been intensively studied in the last 20 years both because of its theoretical interest and its wide practical applicability in operations research, computer science, engineering and management science. Because of the increasing number of the potential applications, numerous algorithms have been developed to solve the KP especially for large problem sizes.

Several faster algorithms have been produced which provide good solutions and approximate solutions. For example, Sahni [3] introduced approximation algorithms to 0-1 KP. Ibarra and Kim [4] developed a fully polynomial approximation scheme for the KP. Martello and Toth [5] designed a quite effective algorithm for large-size problems, which is based on the use of a greedy algorithm for solving large KP. Horowitz et al. [6] also presented a very simple algorithm solving the KP using a greedy method.

Also knapsack problem is a sub problem of one-dimensional cutting stock problem [7]. Therefore, KP is also a very important problem for operations research. This paper aims to solve the problems with large sizes in minimum amount of time with near-optimal values.

The KP can be stated as follows:
Given a set of items (i = 1, . . . , n), each with a weight $w_i > 0$ and a profit $p_i > 0$, the aim is to determine the number of each item to include in a collection so that the total weight is less than a given limit and the total profit is as large as possible.

The problem can be defined by the following integer linear programming:

$$\text{maximize} \quad \sum_{i=1}^{n} p_i x_i$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \le c,$$

$$x_i \in \{0,1\}, \qquad i = 1,\ldots,n$$

Here,

p_i : profit of item i,

w_i : weight of item i,

c : capacity of the knapsack,

*Corresponding author: Address: Department of Computer Programming, Yasar University, Izmir, Turkey, Turkey**.** E-mail address: asli.guler@yasar.edu.tr, Phone: +90 05335602317.

Doi:10.21541/apjes.14020

$$x_i = \begin{cases} 1, & \text{if item } i \text{ is selected,} \\ 0, & \text{otherwise.} \end{cases}$$

It is assumed, without loss of generality, that p_i, w_i and c are positive integers, besides

$$w_i \le c, \quad i = 1,...,n$$

$$\sum_{i=1}^{n} w_i \ge c .$$

2. Genetic algorithms

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 [8].

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 [9].

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.

The genetic algorithm approach has been found quite effective in obtaining the solution of a large variety of complex optimization problems, such as multidimensional knapsack (Chu & Beasley, [10]), subset-sum (Spillman, [11]) and bin packing (Falkenauer, [12]; Hussain & Sastry, [13]). Khuri et al. [14] extended previous work for the single constraint knapsack problem. A similar study was given in Battiti and Tecchioli [15]. Chu and Beasley gave the first successful implementation of GA's by restricting the genetic algorithms to search only the feasible search space. Moreover, there are recent studies for 0-1 KP where GA's are used. For example, Khan proposed a new algorithm which uses masked mutation [16], while Zhao et. al. prefer greedy strategy for their genetic algorithm [17]. In the study of Neoh et. al. [18] genetic algorithm and particle swarm optimization are combined to form a new model to solve 0-1 knapsack problem.

3. A new genetic algorithm for KP

In our methodology, *n* items are represented by *n* genes on a bit array that compactly stores the values 0 or 1. If i^{th} element is 1, it means that item *i* is put into the knapsack. When calculating fitness values of items, coefficients of items whose values are 1 in the bit array are summed. Roulette wheel method is used for choosing parents; in this way it is provided that strong individuals produce more children.

Crossover is applied in such a way that roulette wheel is rotated on genes with the same index of parents, that the dominant parent can transfer its genes to the child. Mutation is applied for randomly chosen 25% genes and this process is repeated for the number of individuals.

The steps of the algorithm knapGA are as follows:

- **[GA1]** p_i / w_i, values are calculated, then they are sorted in descending order and index sequence *I* is obtained.
- **[GA2]** *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 ($1 \le i \le n$) of index sequence *I* at each step.
- **[GA3]** The coefficients of the objective function, p_i are sorted in descending order and the first individual is stored as the record. Then, the upper half of the population is transferred to the next generation, without any change, due to elitism.
- **[GA4]** To create the other half of the new generation, roulette wheel method is used and parents are chosen. Roulette wheel is rotated again for all genes of the individual that is being created. In this way, the probability that dominant

parent transfers its genes to the child is taken into consideration.

- **[GA5]** Mutation is applied for randomly chosen 25% genes of the new generation, and new individuals are created by checking the feasible solution (not exceeding the capacity of the knapsack).
- **[GA6]** Steps [GA3], [GA4] and [GA5] are repeated until the iteration number is 10.
- **[GA7]** 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 step [GA2], thus the solution space is scanned much more efficiently. The flow chart of the algorithm knapGA can be seen in Chart 1.

4. Computational experiments

Computational experiments have been carried out by generating random problems for $1 \le w_i \le 100$ and $1 \le p_i \le 100$. In all instances, the capacity of the knapsack is obtained by taking 25%, 50% and 75% of total weight of the items. To find the optimal solutions of the problems, a program written by dynamic programming technique has been used; running times are shown in Table 1, Table 2 and Table 3. If the optimal solution of the problem couldn't be found by DP, GAMS IDE has been used.

Table 1. Optimal values of the problem for 25%

			time (sec)	
n	**c**	**Optimum**	**DP**	**GA**
100	1162	2982	0,000	0,000
200	2599	5518	0,000	0,015
300	4054	8035	0,015	0,031
400	5100	11428	0,015	0,046
500	5902	14135	0,031	0,078
600	7729	16778	0,046	0,140
700	8727	20573	0,062	0,171
800	10041	23190	0,078	0,234
900	11651	25773	0,109	0,296
1000	12333	29241	0,125	0,375
2000	24952	56522	0,546	1,484
3000	38777	86411	1,234	3,328
4000	50874	114753	2,171	5,906
5000	62744	146156	3,359	9,343
6000	75487	173990	4,812	13,515
7000	88562	200902	-	18,281
8000	101489	234443	-	24,156
9000	114276	258826	-	30,235
10000	125586	288929	-	37,484

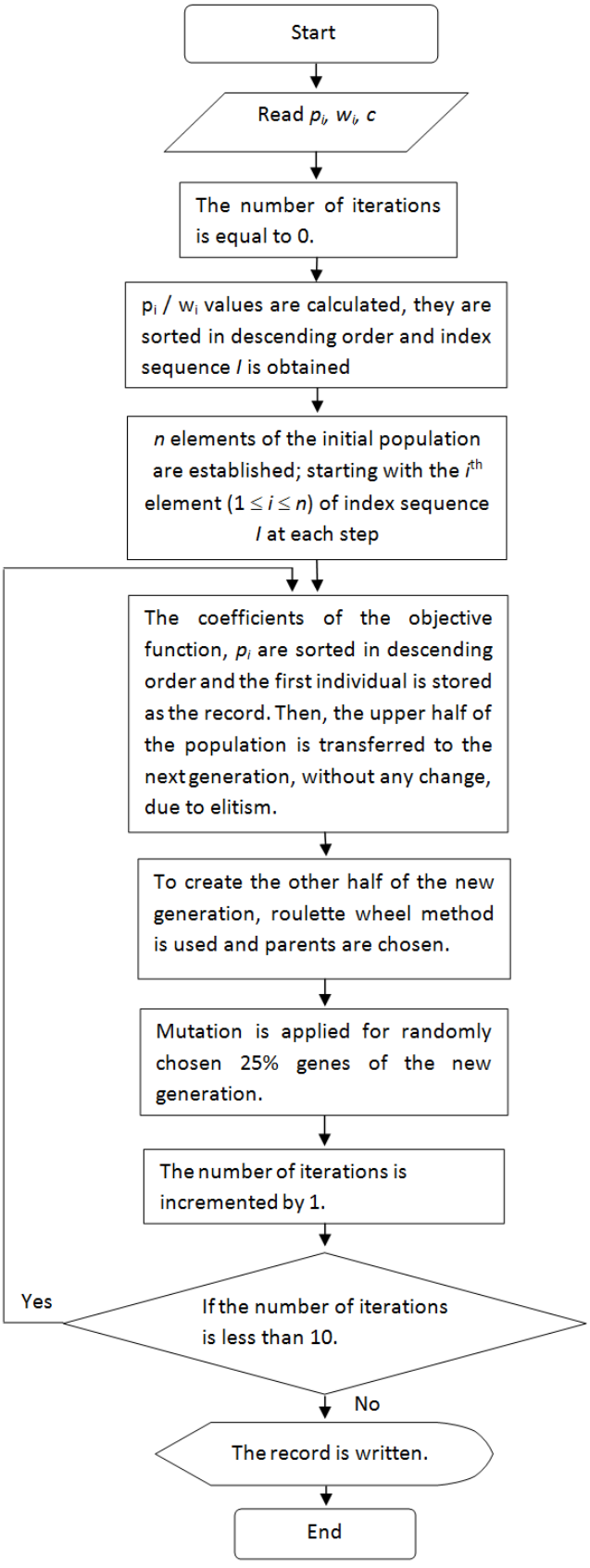

Chart 1. The flow chart of the algorithm knapG

Table 2. Optimal values of the problem for 50 %

			time (sec)	
n	**c**	**Optimum**	**DP**	**GA**
100	2445	4100	0,000	0,000
200	5140	7927	0,000	0,015
300	7637	12570	0,015	0,031
400	10400	16483	0,046	0,062
500	12429	20662	0,078	0,078
600	15301	25026	0,109	0,125
700	17286	29281	0,141	0,187
800	19537	32588	0,171	0,234
900	22585	36155	0,218	0,312
1000	25110	40979	0,281	0,390
2000	50488	82305	1,109	1,562
3000	78109	121976	2,562	3,484
4000	100551	164034	4,406	6,218
5000	125911	203760	-	9,750
6000	150837	243720	-	14,031
7000	176637	283828	-	19,171
8000	202291	330285	-	25,078
9000	226887	367242	-	31,812
10000	251020	412150	-	39,250

Table 3. Optimal values of the problem for 75%

			time (sec)	
n	**c**	**Optimum**	**DP**	**GA**
100	3877	4795	0,000	0,000
200	7896	10134	0,015	0,015
300	11274	14843	0,046	0,031
400	14786	19446	0,062	0,062
500	19394	23389	0,109	0,093
600	21383	28490	0,140	0,125
700	26796	33434	0,203	0,187
800	30532	38319	0,281	0,250
900	33624	42881	0,328	0,328
1000	37393	48126	0,421	0,406
2000	75646	95647	1,671	1,578
3000	116490	143361	3,844	3,546
4000	148290	190228	-	6,328
5000	189026	243225	-	9,984
6000	227118	283215	-	14,343
7000	266520	336159	-	19,671

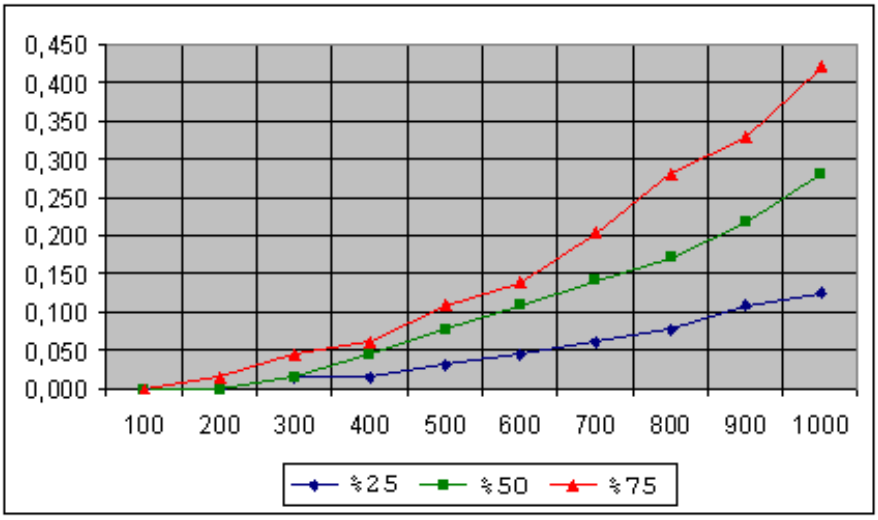

Figure 1. Running times for DP

Fig. 1 and Fig. 2 show the running times for DP and GA respectively. As is seen in Fig. 1, running times obtained by DP change in proportion to the size of parameter *c*, although *n* is taken the same. Moreover, the dependence of computation time of DP technique on the size of the problem is seen in the memory usage as well; the problems in which the value of n*c exceeds a particular number cannot be solved. However, as is seen in Fig. 2, the computation time of the GA is just dependent on parameter *n*, and the memory usage of this technique is much more efficient.

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 addresshttp://kisi.deu.edu.tr/murat.berberler/knap01GA/.

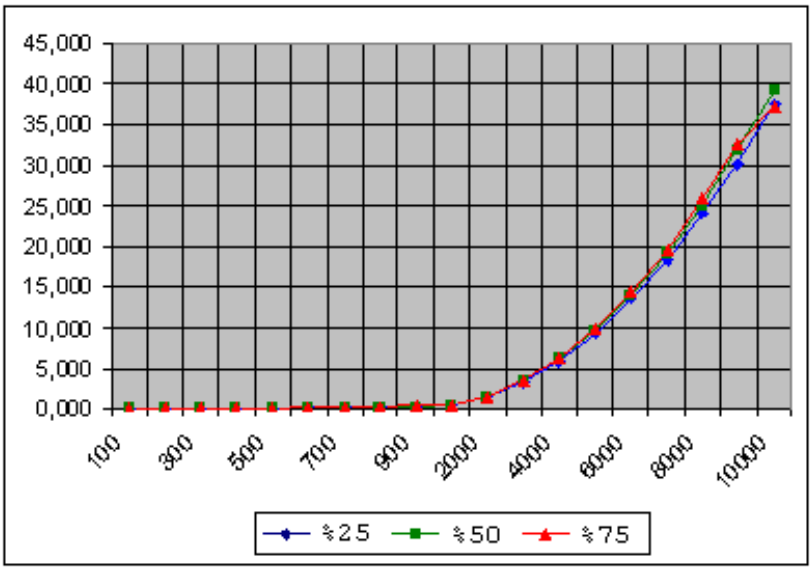

Figure 2. Running times for GA

5. Conclusion

In this paper, the 0-1 Knapsack Problem (KP) 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 KP 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. The algorithm is written in C programming language and is tested on randomly generated instances. It is seen that it yields optimal solutions for all instances. The computation time of the GA is dependent on parameter *n*, while running times obtained by DP change in proportion to the size of parameter *c*, although *n* is taken the same. Furthermore, the memory usage of this technique is much more efficient.

References

[1] Kellerer H., Pferschy U., Pisinger D., Knapsack Problems, (Springer, Berlin, 2004).
[2] Garey M. R. and Johnson D. S., Computers and Intractability: A Guide to the Theory of NP-Completeness, (Freeman, San Francisco, 1979).
[3] Sahni, S., Approximate algorithms for the 0/1 knapsack problems, Journal of ACM, 1, 115-124 (1975).
[4] Ibarra, O. H and Kim, C. E., Fast approximation algorithms for the knapsack and sum of subset problems, Journal of ACM, 22(4), 463-468 (1975).
[5] Martello, S. and Toth, P., Knapsack Problems, (John Wiley&Sons, England, 1990).
[6] Horowitz E., Sahni, S. and Rajasekaran, S., Computer algorithms, (New York: Computer Science Press, W.H. Freeman & Co., 1994).
[7] Berberler M.E., Nuriyev U.G., A new heuristic algorithm for the one-dimensional cutting stock problem, Applied and Computational Mathematics, Vol. 9, No. 1, 19-30, (2010).
[8] Goldberg, D.E., Genetic Algorithms in Search, Optimization and Machine Learning, (Addison–Wesley, 1989).
[9] Chu, P., A Genetic Algorithm Approach for Combinatorial Optimization Problems, Ph.D. thesis at the Management School, Imperial College of Science, London, (1997).
[10] Chu, P., and Beasley, J., A genetic algorithm for the multidimensional knapsack problem. Journal of Heuristics, 4, 63–86 (1998).
[11] Spillman, R., Solving large knapsack problems with a genetic algorithm. Proceedings of IEEE International Conference Systems, Man, and Cybernetics, Vancouver, Canada. Piscataway, NJ: IEEE, 632– 637 (1995).
[12] Falkenauer, E., A hybrid grouping genetic algorithm for bin packing. Journal of Heuristics, 2, 5–30 (1996).
[13] Hussain, S. A., and Sastry, V. U. K., Application of genetic algorithm for bin packing, International Journal of Computer Mathematics, 63, 203–214 (1997).
[14] Khuri, S., Back, T., Heitkotter, J.: The zero/one multiple knapsack problem and genetic algorithms. In: Deaton, E., et al. (eds.) Proc. of the 1994 ACM symposium of Applied Computation, ACM Press, New York 188–193 (1994).
[15] Battiti, R., Tecchiolli, G., Parallel biased search for combinatorial optimization: Genetic algorithms and tabu search, Microprocessors and Microsystems, 16, 351–367 (1992).

[16] Khan M. H. A., An Evolutionary Algorithm with Masked Mutation for 0/1 Knapsack Problem, Proceedings of Informatics, Electronics & Vision (ICIEV), 2013 International Conference on, IEEE, 1-6 (2013).

[17] Zhao J., Pang F., Huang T. and Liu Y., Genetic algorithm based on Greedy strategy in the 0-1 Knapsack Problem, Proceedings of Third International Conference on Genetic and Evolutionary Computing, IEEE, 105-107 (2009).

[18] Neoh S. C., Morad N., Lim C. P. and Aziz Z. A., A GA-PSO layered encoding evolutionary approach to 0/1 knapsack optimization, International journal of innovative computing, information and control 6 (8) 3489-3505 (2010).