Smyrna Journal of Natural and Data Sciences
1(1), 34-40 (2025) - Research Article

An Adaptive Genetic Algorithm for the 0-1 Knapsack Problem

Kazım Erdoğdu*

Abstract
Solving the 0-1 knapsack problem is a combinatorial optimization problem. Although genetic algorithms (GAs) provide strong global search capabilities, early convergence and static parameter sets frequently impair their performance. In this study, an Adaptive Genetic Algorithm is proposed that adaptively selects crossover types during the search. The suggested AGA was tested on a set of small and large benchmark instance sets and compared with the three crossovers' performances.

Keywords: *0-1 knapsack problem, genetic algorithm, adaptive metaheuristic*

* *Corresponding author*

1. Introduction

The knapsack problem (KP) is a combinatorial optimization problem that finds extensive use in computer science, logistics, financial decision-making, and resource allocation. Fundamentally, in the conventional binary version of the KP (i.e., KP01), there are a certain number of items having both value and weight, and a knapsack with a capacity to add the selected items. The aim of the problem is to maximize the profit, that is, the total value of the items packed in the knapsack without exceeding its capacity. Despite its simple definition, KP is an NP-Complete problem (Karp, 1972). For this reason, various heuristic and metaheuristic methods, besides deterministic methods, have been proposed to solve KPs in the literature.

The mathematical model of the 0-1 KP is defined as follows. Given a set of items of $\{1, 2, ...n\}$ of whose elements have both positive profit (p_i) and weight (w_i) values, the aim of the problem is to maximize the total profit without exceeding the knapsack capacity (C)

$$\text{maximize} \sum_{i=1}^{n} p_i x_i \tag{1.1}$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq C \tag{1.2}$$

$$x \in \{0, 1\}, \qquad i = 1, 2, ..., n$$
$$x_i = \begin{cases} 0, & \text{if item } i \text{ is selected} \\ 1, & \text{otherwise} \end{cases} \tag{1.3}$$

The expression in (1.1) indicates the minimization of the objective function of the problem. The expression of (1.2) is the capacity constraint in KP. The expressions in (1.3) define the decision variable of the problem.

Received : *01.05.2025,* ***Accepted :*** *07.05.2025,* ***Available online :*** *20.06.2025*

1.1 Literature Review

Although the concept of the KP was known for a while, Dantzig (1957) was the first one who name and define the problem in the literature. He also proposed two approximation solutions to KP based on linear programming techniques and one exact solution based on the functional equation method. Since then, the different types and variations of KPs and solution methods for KPs have been proposed over time.

The earlier studies approached the solution of the KP with exact methods. Bellman (1957) proposed a dynamic programming approach to KP. Later, Greenberg (1969) and Yormark (1975) improved the Bellman's dynamic approach and applied their improved methods to KPs.

The later studies focused on more heuristic and metaheuristic algorithms in solving KPs. Here is a short list of them.

- Genetic Algorithm (GA): Khuri, Bäck, and Heitkötter (1994), Berberler, Güler, and Nuriyev (2016)
- Ant Colony Optimization (ACO): Changdar, Mahapatra, and Pal (2013), (Schiff, 2013)
- Artificial Bee Colony (ABC): Changdar, Mahapatra, and Pal (2017), Ekmekci (2020)
- Particle Swarm Optimization (PSO): Shen, Wang, Zheng, and Li (2006), Sun et al. (2021)
- Honey Badger Algorithm (HBA): Orucova and Haklı (2023)
- Bat Algorithm (BA): Zhou, Li, and Ma (2015), Chen (2016)
- Cuckoo Search Algorithm (CSA): Gherboudj, Layeb, and Drias (2013), Feng, Jia, and He (2014)
- Simulated Annealing (SA): Drexl (1988), Moradi, Kayvanfar, and Rafiee (2022)

There are also various studies focused on different versions on 0-1 KP in the literature. There are also studies on multi-objective versions of 0-1 KP. A comprehensive survey study can be found in the survey article of Cacchiani, Iori, Locatelli, and Martello (2022).

In this study, an adaptive GA was proposed and applied to the 0-1 KP on two sets of small and large KP benchmark instances. In Section 2, the proposed method is explained. Section 3 provides the experimental results and discussion. Finally, Section 4 wraps up the whole study in this paper.

2. Methods: Adaptive Genetic Algorithm (AGA)

Genetic Algorithms (GAs) are metaheuristic search algorithms that draw inspiration from natural selection. They operate on a population of solutions by using operations including crossover (recombination), mutation, and selection to evolve the next generation of potential solutions. A defined fitness function is used to determine how the population evolves toward better solutions throughout successive generations. GA was first formalized by Holland (1975).

In this study, an adaptive version of GA was proposed. Its pseudocode is given in Algorithm 1. The genotype of the solutions was defined as binary vectors whose elements represent the items. The value 1 means the related item was added to the knapsack, while 0 means it was not added.

The proposed AGA begins with the initialization of the first population of solutions. In the initial population, the first solution is constructed greedily. That is, all the items are sorted in decreasing order according to their profit/weight ratios. Then, starting with the first item, the knapsack is filled with the consecutive items in this sorted list. If the current item in the list cannot fit in the knapsack, then the process continues with the next one. When either the knapsack is full or all the list elements are checked, the greedy filling is completed, and this solution is added to the population. The rest of the solutions are generated at random and added to the initial population. Once the population is constructed with the determined number, the solutions in it are evaluated. The solution with the highest fitness value (i.e., highest profit) is assigned as the current best solution.

After initializing the crossover counters with 1, the generational iteration begins. At each generation, new solutions (i.e., offspring) are produced by crossover and mutation operators. In this study, three crossovers: single point crossover (SPX), two point crossover (TPX), uniform crossover, and one mutation operator: bit-flip were used. The type of crossover is determined according to their success in the previous iterations (line 12 in

Algorithm 1). This feature makes the GA adaptive by giving higher chances to better crossover operators and arranging their counters accordingly. If the selected crossover operator updates the best solution, its counter is incremented by 1. This will increase the probability of being selected in the next iterations. The probability of each crossover type is calculated directly proportional to its counter value. The selected crossover and mutation operators are applied with some probability.

The generation of the offspring is completed for the related generation when the size of the offspring population is equal to the parent population. Afterwards, the parent and offspring populations are combined. This new population is reduced to the original population size by a survival selection mechanism. First, the whole population is sorted in descending order according to the fitness values of the solutions. Then, the top 10% of the population is moved to a new population. The rest of the solutions are selected by binary tournament until the new population size reaches the original population size. Once the total number of iterations is completed, the algorithm returns its best solution. The parameters and their values are given in Table 1

Algorithm 1 Adaptive Genetic Algorithm (AGA)

Initialize population $P[n]$
evaluate(P)
$bestSolution \leftarrow$ best(P)
initializeCounters()
for $k \leftarrow 1$ **to** $maxIterations$ **do**
  $C \leftarrow \emptyset$
  for $i \leftarrow 1$ **to** $\lfloor \frac{n}{2} \rfloor$ **do**
    Select parents $p_1, p_2 \in P$
    $c_1, c_2 \leftarrow p_1, p_2$
    $applied \leftarrow$ **false**
    if $random < crossoverRate$ **then**
      $crossoverType \leftarrow$ selectCrossoverType()
      $c_1, c_2 \leftarrow$ crossover($crossoverType, p_1, p_2$)
      $applied \leftarrow$ **true**
    end if
    if $random < mutationRate$ **then**
      $c_1, c_2 \leftarrow$ mutate(c_1, c_2)
      $applied \leftarrow$ **true**
    end if
    evaluate(c_1, c_2)
    $improved \leftarrow$ update($bestSolution, c_1, c_2$)
    if $improved ==$ **true then**
      incrementCounter($crossoverType$)
    end if
    if $applied ==$ **true then**
      $C \leftarrow C \cup \{c_1, c_2\}$
    end if
  end for
  $P \leftarrow P \cup C$
  $P \leftarrow$ survivalSelection(P)
end for
return $best\_solution$

Table 1. Parameters and Operators in AGA

Parameter / Operator	Value
Number of generations	100
Population size (n)	1000
Crossover types	SPX, TPX, Uniform
Mutation type	Bit-flip
Crossover probability	100%
Mutation probability	10%
Initial crossover counters	1

3. Experimental Results and Discussion

The proposed AGA was tested on 10 small-sized Zhou, Bao, Luo, and Zhang (2017) instances and 12 large-sized Pisinger (2005) instances. They can be found at `http://artemisa.unicauca.edu.co/~johnyortega/instances_01_KP/`. The number of items in small-sized instances varies from 4 to 23, while it varies from 100 to 1000 in the large-sized instances.

Three different crossover versions of the traditional GA (with SPX, TPX, and Uniform) and the AGA proposed in the paper were tested by running 10 times each on these small and large size problems. Tables 2 and 4 show the comparison of the performances of the best results of 10 runs for each version. Their columns from left to right indicate these, respectively: name of the instance, number of items in the problem (**m**), the capacity of the knapsack in the problem (**C**), optimum value in literature, the best result of SPX in 10 runs (**SPX**), the best result of TPX in 10 runs (**TPX**), the best result of Uniform in 10 runs (**Uniform**), and the best result of AGA in 10 runs (**AGA**).

Tables 3 and 5, on the other hand, show how far the related algorithm's best value is from the optimum value (the columns begin with Δ), and the total time that the algorithm spent in the run that it found its best result (the columns begin with "Time(sn)"). Here $\Delta = (Algorithm_{bestb} - Optimum)/Optimum$.

It is clearly seen that all the versions of GA and AGA performed well on the small-sized instances by finding their optimum values (Table 2) in less than 2.5 seconds (Table 3). They, however, could not perform that well on the large-sized instances (Tables 4 and 5). GA with SPX found the optimum values in 4 out of 10, GA with TPX found the optimum values in 5 out of 10, GA with Uniform found the optimum values in 4 out of 10, and AGA found the optimum values in 5 out of 10 instances. A possible reason for this is related to the number of items and the capacities of the knapsacks. As their values increase, the feasible solution space grows, and in turn, the algorithms are challenged more. This challenge can be overcome either by adding a local search mechanism to the GAs or increasing the number of iterations.

Table 2. Best Values of 10 Runs for SPX, TPX, Uniform and AGA on the Small-Sized Instances

Instance	m	C	Optimum	SPX	TPX	Uniform	AGA
f1_ld_kp_10_269	10	269	**295**	**295**	**295**	**295**	**295**
f2_ld_kp_20_878	20	878	**1024**	**1024**	**1024**	**1024**	**1024**
f3_ld_kp_4_20	4	20	**35**	**35**	**35**	**35**	**35**
f4_ld_kp_4_11	4	11	**23**	**23**	**23**	**23**	**23**
f5_ld_kp_15_375	15	375	**481.07**	**481.07**	**481.07**	**481.07**	**481.07**
f6_ld_kp_10_60	10	60	**52**	**52**	**52**	**52**	**52**
f7_ld_kp_7_50	7	50	**50**	**50**	**50**	**50**	**50**
f8_ld_kp_23_10000	23	10000	**9767**	**9767**	**9767**	**9767**	**9767**
f9_ld_kp_5_80	5	80	**130**	**130**	**130**	**130**	**130**
f10_ld_kp_20_879	20	879	**1025**	**1025**	**1025**	**1025**	**1025**

Table 3. Distance of the Best of Crossover Types and AGA to the Optimum and Their Run Times on the Small-Sized Instances

Instance	Δ SPX	Δ TPX	Δ Uniform	Δ AGA	Time(sn) (SPX)	Time(sn) (TPX)	Time(sn) (Uniform)	Time(sn) (AGA)
f1_ld_10_269	0%	0%	0%	0%	1.59	1.61	1.72	1.73
f2_ld_20_878	0%	0%	0%	0%	2.04	2.11	2.37	2.22
f3_ld_1_4_201	0%	0%	0%	0%	1.32	1.17	1.27	1.41
f4_ld_1_4_375	0%	0%	0%	0%	1.31	1.36	1.32	1.44
f5_ld_1_10_11	0%	0%	0%	0%	1.59	1.65	1.75	1.75
f6_ld_1_10_375	0%	0%	0%	0%	1.22	1.41	1.33	1.43
f7_ld_1_20_600	0%	0%	0%	0%	1.29	1.15	2.42	1.75
f8_ld_1_20_3000	0%	0%	0%	0%	2.37	2.11	2.09	2.14
f9_ld_2_2_10000	0%	0%	0%	0%	1.17	1.41	1.41	1.47
f10_ld_1p_20_879	0%	0%	0%	0%	2.07	2.09	2.36	2.28

Table 4. Best Values of 10 Runs for SPX, TPX, Uniform and AGA on the Large-Sized Instances

Instance	**m**	**C**	**Optimum**	**SPX**	**TPX**	**Uniform**	**AGA**
knapPI_1_100_1000_1	100	995	**9147**	**9147**	**9147**	**9147**	**9147**
knapPI_1_200_1000_1	200	1008	**11238**	**11238**	**11238**	**11238**	**11238**
knapPI_1_500_1000_1	500	2543	**28857**	26943	27182	25983	27061
knapPI_1_1000_1000_1	1000	5002	**9052**	7534	7431	7026	7342
knapPI_1_100_1000_1	100	995	**1514**	**1514**	**1514**	**1514**	**1514**
knapPI_1_200_1000_1	200	1008	**1634**	**1634**	**1634**	1626	1626
knapPI_1_500_1000_1	500	2543	**4566**	4119	4207	3944	4126
knapPI_1_1000_1000_1	1000	5002	**14390**	11475	11384	10026	11269
knapPI_1_100_1000_1	100	995	**2397**	2297	2396	2396	**2397**
knapPI_1_200_1000_1	200	1008	**2697**	2693	**2697**	**2697**	**2697**
knapPI_1_500_1000_1	500	2543	**7117**	6551	6565	5679	6416
knapPI_1_1000_1000_1	1000	5002	**54503**	43968	45164	41100	45964

4. Conclusion

In this study, a version of GA, called AGA, was proposed and applied to the 0-1 Knapsack Problem. AGA, adaptively selects among three crossovers based on their performances in the previous iterations during the search. Each time a crossover contributes to the search by improving the current best solution, its counter is incremented. The counters of each crossover type determine the probability of being selected for the crossover. To the best of our knowledge, this is the first study to apply this mechanism to a GA in the 0-1 KP problem. This is the main contribution of this study.

Although the proposed AGA performed well on the small-sized benchmark instances, it found the optimum values in half of the large-sized benchmark instances.

As a future work, AGA can be hybridized with some local search heuristics or improved in its crossover selection mechanism. Once the optimum values are obtained, it can be tested on other optimization problems, as well.

Article Information

Ethical Approval and Participant Consent: It is declared that during the preparation process of this study, scientific and ethical principles were followed and all the studies benefited from are stated in the bibliography.

Conflict of Interest Disclosure: No potential conflict of interest was declared by the author.

Table 5. Distance of the Best of Crossover Types and AGA to the Optimum and Their Run Times on the Large-Sized Instances

Instance	Δ SPX	Δ TPX	Δ Uniform	Δ AGA	Time(sn) (SPX)	Time(sn) (TPX)	Time(sn) (Uniform)	Time(sn) (AGA)
knapPI_1_100_1000_1	0.00%	0.00%	0.00%	0.00%	4.70	4.73	6.18	13.97
knapPI_1_200_1000_1	0.00%	0.00%	0.00%	0.00%	8.04	8.18	11.11	26.02
knapPI_1_500_1000_1	6.63%	5.80%	9.96%	6.22%	18.27	18.38	25.97	53.61
knapPI_2_1000_1000_1	16.77%	17.91%	22.38%	18.89%	36.60	36.64	51.78	107.02
knapPI_2_100_1000_1	0.00%	0.00%	0.00%	0.00%	4.74	4.72	6.18	5.51
knapPI_2_200_1000_1	0.00%	0.00%	0.49%	0.49%	8.05	14.70	29.38	8.50
knapPI_2_500_1000_1	9.79%	7.86%	13.62%	9.64%	19.67	19.51	46.30	19.71
knapPI_3_1000_1000_1	20.26%	20.89%	30.33%	21.69%	41.49	41.99	57.00	40.00
knapPI_3_100_1000_1	4.17%	0.04%	0.04%	0.00%	5.42	5.18	7.21	5.38
knapPI_3_200_1000_1	0.15%	0.00%	0.00%	0.00%	9.57	9.38	11.55	8.77
knapPI_3_500_1000_1	7.95%	7.76%	20.21%	9.85%	19.60	20.81	29.08	20.05
knapPI_1_1000_1000_1	19.33%	17.13%	24.59%	15.67%	35.84	35.45	50.31	43.11

Availability of data and materials: Not applicable.

Plagiarism Statement: This article was scanned by the plagiarism program. No plagiarism detected.

References

Bellman, R. (1957, October). Letter to the Editor—Comment on Dantzig's Paper on Discrete Variable Extremum Problems. *Operations Research*, *5*(5), 723-724. doi: 10.1287/opre.5.5.723

Berberler, M. E., Güler, A., & Nuriyev, U. (2016). A new genetic algorithm for the 0-1 knapsack problem. *Academic Platform - Journal of Engineering and Science*, *4*(3). doi: 10.21541/apjes.14020

Cacchiani, V., Iori, M., Locatelli, A., & Martello, S. (2022). Knapsack problems — an overview of recent advances. part ii: Multiple, multidimensional, and quadratic knapsack problems. *Computers & Operations Research*, *143*, 105693. doi: doi.org/10.1016/j.cor.2021.105693

Changdar, C., Mahapatra, G., & Pal, R. K. (2013). Solving 0–1 knapsack problem by continuous aco algorithm. *International Journal of Computational Intelligence Studies*, *2*(3/4), 333–349. doi: 10.1504/IJCISTUDIES.2013.057638

Changdar, C., Mahapatra, G. S., & Pal, R. K. (2017). A modified artificial bee colony approach for the 0–1 knapsack problem. *Applied Intelligence*, *47*(4), 1011–1028. doi: 10.1007/s10489-017-1025-x

Chen, Y. (2016). A novel bat algorithm of solving 0-1 knapsack problem. In *Proceedings of the 2016 4th international conference on machinery, materials and computing technology* (p. 1597-1600). Atlantis Press. doi: 10.2991/icmmct-16.2016.318

Dantzig, G. B. (1957). Discrete-variable extremum problems. *Operations Research*, *5*(2), 266–277. doi: 10.1287/opre.5.2.266

Drexl, A. (1988). A simulated annealing approach to the multiconstraint zero-one knapsack problem. *Computing*, *40*(3), 211–221. doi: 10.1007/BF02242185

Ekmekci, D. (2020). 0-1 Çok boyutlu sırt Çantası probleminin feromonal yapay arı koloni (fyak) algoritması ile Çözümü. *Academic Platform - Journal of Engineering and Science*, *8*(2), 355–364. doi: 10.21541/apjes.640252

Feng, Y., Jia, K., & He, Y. (2014, 01). An improved hybrid encoding cuckoo search algorithm for 0-1 knapsack problems. *Computational intelligence and neuroscience*, *2014*, 970456. doi: 10.1155/2014/970456

Gherboudj, M., Layeb, A., & Drias, H. (2013). Solving 0-1 knapsack problems by a discrete binary version of cuckoo search algorithm. *International Journal of Computer Science Issues (IJCSI)*, *10*(3), 87–93.

Greenberg, H. (1969). An algorithm for the computation of knapsack functions. *Journal of Mathematical Analysis and Applications*, *26*(1), 159-162. doi: 10.1016/0022-247X(69)90185-1

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press.

Karp, R. M. (1972). Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, & J. D. Bohlinger (Eds.), *"complexity of computer computations: Proceedings of a symposium on the complexity of computer computations, held march 20–22, 1972, at the ibm thomas j. watson research center, yorktown heights, new york* (pp. 85–103). Boston, MA: Springer US. doi: 10.1007/978-1-4684-2001-2_9

Khuri, S., Bäck, T., & Heitkötter, J. (1994). The zero/one multiple knapsack problem and genetic algorithms. In *Proceedings of the 1994 acm symposium on applied computing* (p. 188–193). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/326619.326694

Moradi, N., Kayvanfar, V., & Rafiee, M. (2022). An efficient population-based simulated annealing algorithm for 0–1 knapsack problem. *Engineering with Computers*, *38*(3), 2771–2790. doi: 10.1007/s00366-020-01240-3

Orucova, G. B., & Haklı, H. (2023). Binary honey badger algorithm for 0-1 knapsack problem. *Journal of Intelligent Systems: Theory and Applications*, *6*(2), 108–118. doi: 10.38016/jista.1200225

Pisinger, D. (2005). Where are the hard knapsack problems? *Computers & Operations Research*, *32*(9), 2271–2284. doi: 10.1016/j.cor.2004.03.002

Schiff, K. (2013). Ant colony optimization algorithm for the 0–1 knapsack problem. *Technical Transactions*, *110*(3-AC), 39–52.

Shen, X., Wang, W., Zheng, B., & Li, Y. (2006). Based on improved optimizing particle swarm optimization to solve 0-1 knapsack problem. *Computer Engineering*, *32*(18), 23–24.

Sun, W.-Z., Zhang, M., Wang, J.-S., Guo, S.-S., Wang, M., & Hao, W.-K. (2021). Binary particle swarm optimization algorithm based on z-shaped probability transfer function to solve 0–1 knapsack problem. *IAENG International Journal of Computer Science*, *48*(2), 294–303.

Yormark, J. S. (1975). Accelerating greenberg's method for the computation of knapsack functions. *Journal of Mathematical Analysis and Applications*, *49*(3), 695–702. doi: 10.1016/0022-247X(75)90202-4

Zhou, Y., Bao, Z., Luo, Q., & Zhang, S. (2017). A complex-valued encoding wind driven optimization for the 0-1 knapsack problem. *Applied Intelligence*, *46*(3), 684–702. doi: 10.1007/s10489-016-0855-2

Zhou, Y., Li, L., & Ma, M. (2015). A complex-valued encoding bat algorithm for solving 0–1 knapsack problem. *Neural Processing Letters*, *44*(3), 407–430. doi: 10.1007/s11063-015-9465-y

Affiliation

Kazim Erdoğdu
Address: Department of Software, Yaşar University, Üniversite Caddesi No:37-39, Bornova, İzmir, Türkiye
E-mail: kazim.erdogdu@yasar.edu.tr
Orcid: 0000-0001-6256-3114