Research Inventy: International Journal of Engineering And Science
Vol.14, Issue 7 (July 2024), PP 67-73
Issn (e): 2278-4721, Issn (p):2319-6483, www.researchinventy.com

Optimization Knapsack Problem Using a Genetic Algorithm-Artificial Immune System (GA-AIS)

Erianto Ongko, Sukiman, Hendry
Department of Digital Business, Institut Modern Arsitektur & Teknologi, Medan, Indonesia
Corresponding Author: eriantoongko@gmail.com

Abstract: *The Knapsack Problem is a well-known combinatorial optimization problem that aims to maximize the total utility of selected items within a given weight constraint. This study presents a comparative analysis of three algorithms to solve the Knapsack Problem: Genetic Algorithm (GA), Artificial Immune System (AIS), and a hybrid GA-AIS algorithm. We generated test data comprising 50 items with specific weights and values, and a knapsack capacity of 1000. The experimental results demonstrated that all three algorithms were effective in maximizing the total value while adhering to the weight constraint. The AIS and GA-AIS algorithms achieved the highest total value of 3123, outperforming the GA's total value of 3121. In terms of computational efficiency, the GA-AIS hybrid required the fewest iterations (69), followed by GA (71), and AIS (85). The GA-AIS hybrid algorithm stands out for its combination of high total value and computational efficiency, making it a highly effective approach for solving the Knapsack Problem.*

Keywords: *Knapsack Problem, Genetic Algorithm (GA), Artificial Immune System (AIS), Optimization, GA-AIS*

Date of Submission: 08-07-2024 Date of acceptance: 23-07-2024

I. INTRODUCTION

The Knapsack Problem is a renowned combinatorial optimization problem. It involves selecting a subset of items from a given set, each with a weight and a utility value. The objective is to maximize the total utility while adhering to a budget constraint on the total weight of the selected items. An issue frequently arises in the allocation of resources, wherein the individual responsible for making decisions must select from a collection of indivisible projects or tasks within a predetermined budget(Baldo et al., 2023). The knapsack problem has been recognized for more than a century. Contrary to popular belief, the name "knapsack" was not suggested by Tobias Dantzig, the father of George Dantzig (1884-1956), as the term does not appear in his book. The initial studies on algorithms were published in the 1950s by Dantzig and Bellman. However, it was in the 1960s that a significant surge in research activity began. The subsequent decades have demonstrated the success of this topic, as evidenced by its ranking as the 18th most popular algorithmic problem and the second most popular among the NP-hard problems(Cacchiani et al., 2022).

Since Dantzig's groundbreaking work, knapsack problems have received extensive research attention and have had significant applications in various industries and financial management. In addition, several integer programming problems can be converted into knapsack problems by relaxing certain constraints. Consequently, solving knapsack problems has become the benchmark for computational efficiency in integer problems. Numerous novel contributions have been documented in the subsequent literature to address this issue. A genetic algorithm-based metaheuristic has been developed to solve the multidimensional knapsack problem(Wu et al., 2017). Numerous meta-heuristic algorithms with a broad range of applications to various real-world problems are available in the literature. The paper primarily focuses on the SA (Simulated Annealing) and GA (Genetic Algorithm) as they serve as foundational algorithms for other recently developed meta-heuristic algorithms(Ezugwu et al., 2019).

Metaheuristic methods utilize various operators, including particles in particle swarm optimization (PSO)(Dahmani et al., 2020), fireflies in the firefly algorithm (FA)(Bhattacharjee & Sarmah, 2015), ants in ant colony algorithms (ACO)(BenMansour, 2023), chromosomes in genetic algorithms (GA)(Saraswat & Tripathi, 2021), and antibodies in the Artificial Immune System. The objective of these algorithms is to identify a global optimal or nearly global optimal through an iterative search. Populations will consistently migrate towards the global optimum and avoid getting stuck in local optima by employing various strategies, including information transmission, evolutionary tactics, and social behavior(Y. Zhou et al., 2023).

Genetic Algorithms (GA) have demonstrated their efficacy in achieving near-optimal solutions for complex problems. However, to enhance the efficiency of solutions and avoid early convergence, the operators used in the genetic algorithm relied heavily on large population sizes, resulting in slower computational speed compared to more recent algorithms. Hence, in this study, we present a method to achieve a substantial enhancement in GA for superior solution quality and reduced computational expenses while maintaining simplicity of implementation. Therefore, we examined the clonal selection algorithm from Artificial Immune Systems (AIS) to draw inspiration for genetic processes in Genetic Algorithms (GA). Next, we conducted a performance comparison between the proposed algorithm, known as GA-AIS, and its ability to solve the knapsack problem(Farhan & Zukhri, 2020).

II. EXPERIMENTAL PROCEDURE

2.1. Mathematical Model of Knapsack Problem

The classic problem of Knapsack Probblem is 0/1 Knapsack Problem. Here's a mathematical model for the 0/1 Knapsack Problem.

Notation

- n: Number of items.
- w_i: Weight of item i
- v_j: value of item i.
- W: maximum capacity of Knapsack.
- D_k: Total demand for demand k.
- x_i: Binary variable, 1 if item i is selected, 0 otherwise.

Objective

Maximize the total value without exciding Knapsack capacity:

$$Maximize\ Z = \sum_{i=1}^{n} v_i x_i \tag{1}$$

Constraints

1. Capacity constraints for each knapsack:

$$\sum_{i=1}^{n} w_i x_i \leq W \tag{2}$$

2. Binary Constraints for item selection:

$$x_i \in \{0,1\}\ \forall i = 1,2,\dots,n \tag{3}$$

Model Summary

$$Maximize\ Z = \sum_{i=1}^{n} v_i x_i$$

Subject to:

$$\sum_{i=1}^{n} w_i x_i \leq W$$

$$x_i \in \{0,1\}\ \forall i = 1,2,\dots,n$$

The 0/1 Knapsack Problem is a classic combinatorial optimization problem where the objective is to maximize the total value of items placed in a knapsack without exceeding its capacity. Each item has a specific weight and value, and the decision to include an item is binary (either included or not). The mathematical model consists of maximizing the sum of the values of the selected items while ensuring that the total weight does not surpass the knapsack's capacity.

2.2. Genetic Algorithm

The pseudocode for the Genetic Algorithm (GA) is as follows:

```
Input: Population size N, Crossover rate β, Mutation rate ρ, Number of generations G
Output: Best solution found
1. P ← initialize_population(N)
2. evaluate_fitness(P)
3. For g = 1 to G do
   a. Parents ← select_parents(P)
   b. Offspring ← crossover(Parents, β)
   c. Mutated_Offspring ← mutate(Offspring, ρ
   d. evaluate_fitness(Mutated_Offspring)
   e. P ← replace_population(P, Mutated_Offspring, N)
4. best_solution ← find_best_solution(P)
5. Return best_solution
End
```

The pseudocode above shows the evolution process using a Genetic Algorithm (GA) which aims to find the best solution in an optimization problem. The algorithm starts by initializing an initial population P with size N. Each individual in the population is evaluated for its fitness value. Next, in each generation G, several steps are carried out: first, the best individuals are selected as parents. Second, the crossover is carried out on the parental pair to produce new offspring with a crossover level of β. Third, the offspring are mutated with a mutation rate ρ using the swap_genes, reverse_subsequence, and insert_gene operations. After that, the fitness value of the mutated offspring is evaluated. The old population is then replaced by a new population consisting of the best individuals from the mutated breeds. This process is repeated until the specified number of G generations is reached. Finally, the best solution found in the current population is returned as the final result. This algorithm combines selection, crossover, and mutation to explore the solution space and find the optimal solution effectively.

2.3. Artificial Immune System

The pseudocode for the Artificial Immune System (AIS) is as follows:

```
Input: Population size N, Number of clones β, Muation rate ρ, Number of generations G
Output: Best solution found
1. P ← initialize_population(N)
2. evaluate_fitness(P)
3. For g = 1 to G do
   a. Best_Solutions ← select_high_affinity(P)
   b. Clones ← clone_solutions(Best_Solutions, β)
   c. Mutated_Clones ← mutate_clones(Clones, ρ)
      i. Operations: swap_genes, reverse_subsequence, insert_gene
   d. evaluate_fitness(Mutated_Clones)
   e. P ← replace_population(P, Mutated_Clones, N)
4. best_solution ← find_best_solution(P)
5. Return best_solution
End
```

This pseudocode describes the Artificial Immune System (AIS) algorithm that initializes a random population of solutions and evaluates the fitness of each solution. In each generation, the best solution is selected and cloned; then, the clones are mutated using operations such as gene swapping, subsequence reversal, and gene insertion. The fitness of the mutated clones is reevaluated, and the mutated clones replace the original population. This process is repeated for several generations until the algorithm finds and returns the best solution found throughout the iterations.

2.4. Knapsack Problem

The pseudocode for solving the Knapsack Problem is as follows:

```
Input: Weights W, Values V, Number of items n, Knapsack capacity C
Output: Maximum value that can be carried in the knapsack
1. function knapsack(W, V, n, C)
2.     Create a 2D array K of size (n+1) x (C+1)
3.     for i from 0 to n do
4.         for w from 0 to C do
5.             if i == 0 or w == 0 then
6.                 K[i][w] ← 0
7.             else if W[i-1] <= w then
8.                 K[i][w] ← max(V[i-1] + K[i-1][w-W[i-1]], K[i-1][w])
9.             else
10.                K[i][w] ← K[i-1][w]
11.    return K[n][C]
End
```

The pseudocode above solves the knapsack problem using a dynamic programming approach. This algorithm creates a 2D array of K with size (n+1)×(C+1)(n+1) to store the maximum value that can be obtained for each combination of a number of items and knapsack capacity. Table initialization is done with a value of zero when the number of items or capacity is zero. The algorithm then populates the table by checking whether each item whether to be placed in the knapsack or not based on whether the item's weight is less than or equal to the current capacity. If an item is included, the maximum value is calculated as the sum of the item value and the optimal value of the remaining capacity; otherwise, the previous optimal value is maintained. This process continues until the table is complete, and the maximum obtainable value is found at K[n][C].

2.5. Proposed Method

This study suggests using a combination of Genetic Algorithms (GA) and Artificial Immune Systems (AIS) to address the Knapsack Problem.

```
1. P ← initialize_population(N, W, V, C)
2. evaluate_fitness(P, W, V, C)
3. For g = 1 to G do
   a. Parents ← select_parents(P)
   b. Offspring ← crossover(Parents, β)
   c. Mutated_Offspring ← mutate(Offspring, ρ)
   d. evaluate_fitness(Mutated_Offspring, W, V, C)
   e. Combined_Population ← P + Mutated_Offspring
   f. Best_Solutions ← select_high_affinity(Combined_Population)
   g. Clones ← clone_solutions(Best_Solutions, γ)
   h. Mutated_Clones ← mutate(Clones, ρ)
   i. evaluate_fitness(Mutated_Clones, W, V, C)
   j. Combined_Population ← Combined_Population + Mutated_Clones
   k. P ← select_best_individuals(Combined_Population, N)
4. best_solution ← find_best_solution(P)
5. Return best_solution
End

function initialize_population(N, W, V, C)
   P ← []
   for i from 1 to N do
      individual ← random_binary_vector(length(W))
      P.append(individual)
   return P

function evaluate_fitness(P, W, V, C)
   for individual in P do
      weight, value ← 0, 0
      for i from 1 to length(individual) do
         if individual[i] == 1 then
            weight += W[i-1]
            value += V[i-1]
      if weight > C then
         individual.fitness ← 0
      else
         individual.fitness ← value

function select_parents(P)
   return tournament_selection(P)

function crossover(Parents, β)
   Offspring ← []
   for i from 1 to length(Parents) by 2 do
      if random(0, 1) < β then
         Offspring.append(perform_crossover(Parents[i], Parents[i+1]))
      else
         Offspring.append(Parents[i], Parents[i+1])
   return Offspring

function mutate(Individuals, ρ)
   for individual in Individuals do
      if random(0, 1) < ρ then
         perform_mutation(individual)
   return Individuals

function select_high_affinity(P)
   P.sort_by_fitness()
   return P[1:floor(length(P)/2)]
```

```
function clone_solutions(Best_Solutions, γ)
   Clones ← []
   for solution in Best_Solutions do
      for i from 1 to γ do
         Clones.append(solution)
   return Clones

function select_best_individuals(P, N)
   P.sort_by_fitness()
   return P[1:N]

function find_best_solution(P)
   P.sort_by_fitness()
   return P[1]

function perform_crossover(parent1, parent2)
   point ← random(1, length(parent1)-1)
   return parent1[1:point] + parent2[point+1:end], parent2[1:point] + parent1[point+1:end]

function perform_mutation(individual)
   point ← random(1, length(individual))
   individual[point] ← 1 - individual[point]
```

III. RESULTS AND DISCUSSIONS

3.1. Results

We generated the testing data with 50 objects. The results of the testing as follows
Size={ 91, 72, 63, 81, 71, 82, 67, 51, 52, 35, 60, 41, 38, 41, 49, 41, 19, 40, 50, 41, 39, 43, 31, 33, 27, 31, 40, 39, 40, 29, 39, 41, 59, 49, 30, 59, 19, 31, 40, 20, 30, 15, 25, 20, 30, 40, 4, 5, 2, 2}, Value={190, 218, 207, 202, 220, 171, 156, 161, 157, 156, 151, 121, 131, 131, 111, 117, 141, 120, 111, 121, 108, 99, 101, 102, 85, 79, 81, 86, 90, 73, 74, 69, 71, 68, 70, 65, 56, 65, 59, 61, 46, 40, 41, 31, 21, 18, 9, 6, 4, 2}, and W = 1000.

In order to facilitate comparison, we implemented the genetic algorithm, the Artificial Immune System, and GA-IAS. The results of the experiment are as follows:

1. The Genetic Algorithm: Total Value is 3121 Total weight is 1000, maximum 71 iteration
2. The Artificial Immune System Algorithm: Total Value is 3123 Total weight is 1000, maximum 85 iteration
3. The GA-AIS Algorithm: Total Value is 3123 Total weight is 1000, maximum 69 iteration

3.2. Discussion

All three algorithms were able to fill the knapsack to its maximum capacity (1000) while achieving high total values. The AIS and GA-AIS algorithms both achieved the highest total value of 3123, slightly outperforming the GA, which achieved a total value of 3121. This indicates that the AIS and GA-AIS algorithms were slightly more effective in selecting the optimal combination of items.

Maximum Iterations

The number of iterations required to reach the optimal or near-optimal solution varied among the algorithms:

- The GA-AIS hybrid algorithm required the fewest iterations (69), suggesting a more efficient search process.
- The GA required 71 iterations, indicating a slightly less efficient search compared to the GA-AIS.
- The AIS required the most iterations (85), suggesting that while it achieved the highest total value, it did so with greater computational effort.

Comparative Analysis

1. **Genetic Algorithm (GA)**: The GA demonstrated robust performance, achieving a high total value with a relatively low number of iterations. Its total value was only marginally lower than the AIS and GA-AIS algorithms, indicating it is a competitive approach for the Knapsack Problem.
2. **Artificial Immune System (AIS)**: The AIS achieved the highest total value but required the most iterations to do so. This indicates that while AIS is highly effective in finding optimal solutions, it may require more computational effort and time.

3. **GA-AIS Hybrid Algorithm**: The GA-AIS hybrid algorithm combined the strengths of both GA and AIS, achieving the highest total value with the fewest iterations. This suggests that the hybrid approach is highly efficient and effective, leveraging the exploration capabilities of GA and the optimization strengths of AIS.

Practical Implications

The choice of algorithm depends on the specific requirements and constraints of the problem at hand:

- **Efficiency**: If computational efficiency and speed are paramount, the GA-AIS hybrid algorithm is the preferred choice due to its lower iteration count.
- **Optimality**: If achieving the absolute highest value is the goal, both AIS and GA-AIS are suitable, with AIS being slightly more thorough but at a higher computational cost.
- **Balance**: The GA provides a good balance between efficiency and optimality, making it a versatile choice for various problem sizes and constraints.

IV. CONCLUSION

The comparative study highlights that while all three algorithms are effective for the Knapsack Problem, the GA-AIS hybrid algorithm stands out due to its combination of high total value and computational efficiency. The AIS algorithm, while achieving the highest total value, requires more iterations, making it less efficient in terms of computational effort. The GA offers a balanced approach, achieving competitive results with moderate computational effort. Future work could explore further optimization of these algorithms, potentially integrating additional heuristics or adaptive mechanisms to enhance performance. Additionally, testing on larger datasets and different problem instances would provide more insights into the scalability and robustness of these algorithms.

Conflict of interest

The authors certify that no personal relationships or known competing financial interests could have compromised the integrity of the research presented in this article.

ACKNOWLEDGEMENT

The authors wish to express their gratitude to the editor and reviewers for their insightful feedback and recommendations. The research was funded in part by Institut Modern Arsitektur & Teknologi.

REFERENCES

[1]. Antariksa, G., Muammar, R., & Lee, J. (2022). Performance evaluation of machine learning-based classification with rock-physics analysis of geological lithofacies in Tarakan Basin, Indonesia. Journal of Petroleum Science and Engineering, 208, 109250. https://doi.org/10.1016/j.petrol.2021.109250

[2]. Baldo, A., Boffa, M., Cascioli, L., Fadda, E., Lanza, C., & Ravera, A. (2023). The polynomial robust knapsack problem. European Journal of Operational Research, 305(3), 1424–1434. https://doi.org/10.1016/j.ejor.2022.06.029

[3]. BenMansour, I. (2023). An Effective Hybrid Ant Colony Optimization for the Knapsack Problem Using Multi-Directional Search. SN Comput. Sci., 4(2). https://doi.org/10.1007/s42979-022-01564-5

[4]. Bhattacharjee, K. K., & Sarmah, S. P. (2015). A binary firefly algorithm for knapsack problems. 2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), 73–77. https://doi.org/10.1109/IEEM.2015.7385611

[5]. Cacchiani, V., Iori, M., Locatelli, A., & Martello, S. (2022). Knapsack problems — An overview of recent advances. Part I: Single knapsack problems. Computers & Operations Research, 143, 105692. https://doi.org/10.1016/j.cor.2021.105692

[6]. Cappanera, P., & Trubian, M. (2005). A Local-Search-Based Heuristic for the Demand-Constrained Multidimensional Knapsack Problem. INFORMS Journal on Computing, 17(1), 82–98. https://doi.org/10.1287/ijoc.1030.0050

[7]. Dahmani, I., Hifi, M., Saadi, T., & Yousef, L. (2020). A swarm optimization-based search algorithm for the quadratic knapsack problem with conflict Graphs. Expert Systems with Applications, 148, 113224. https://doi.org/10.1016/j.eswa.2020.113224

[8]. Ezugwu, A. E., Pillay, V., Hirasen, D., Sivanarain, K., & Govender, M. (2019). A Comparative Study of Meta-Heuristic Optimization Algorithms for 0 – 1 Knapsack Problem: Some Initial Results. IEEE Access, 7, 43979–44001. IEEE Access. https://doi.org/10.1109/ACCESS.2019.2908489

[9]. Farhan, L. O. M., & Zukhri, Z. (2020). Hybridization of Genetic Algorithm and Artificial Immune System for Assignment Problem. IOP Conference Series: Materials Science and Engineering, 803(1), 012024. https://doi.org/10.1088/1757-899X/803/1/012024

[10]. Li, Y., Wang, Y., Li, T., Li, B., & Lan, X. (2021). SP-SMOTE: A novel space partitioning based synthetic minority oversampling technique. Knowledge-Based Systems, 228, 107269. https://doi.org/10.1016/j.knosys.2021.107269

[11]. Lu, Y., & Vasko, F. J. (2020). A comprehensive empirical demonstration of the impact of choice constraints on solving generalizations of the 0–1 knapsack problem using the integer programming option of CPLEX®. Engineering Optimization, 52(9), 1632–1644. https://doi.org/10.1080/0305215X.2019.1658748

[12]. Ruuska, S., Hämäläinen, W., Kajava, S., Mughal, M., Matilainen, P., & Mononen, J. (2018). Evaluation of the confusion matrix method in the validation of an automated system for measuring feeding behaviour of cattle. Behavioural Processes, 148, 56–62. https://doi.org/10.1016/j.beproc.2018.01.004

[13]. Saraswat, M., & Tripathi, R. C. (2021). Solving Knapsack Problem with Genetic Algorithm Approach. In Mathematical Modeling and Computation of Real-Time Problems. CRC Press.

[14]. Song, M. S., Emerick, B., Lu, Y., & Vasko, F. J. (2022). When to use Integer Programming Software to solve large multi-demand multidimensional knapsack problems: A guide for operations research practitioners. Engineering Optimization, 54(5), 894–906. https://doi.org/10.1080/0305215X.2021.1933965

[15]. Wu, Z., Hu, F., & Fu, B. (2017). Solving the large-scale knapsack feasibility problem using a distributed computation approach to integer programming. Applied Informatics, 4(1), 18. https://doi.org/10.1186/s40535-017-0047-0

[16]. Zhou, S., Gu, Y., Yu, H., Yang, X., & Gao, S. (2023). RUE: A robust personalized cost assignment strategy for class imbalance cost-sensitive learning. Journal of King Saud University - Computer and Information Sciences, 35(4), 36–49. https://doi.org/10.1016/j.jksuci.2023.03.001

[17]. Zhou, Y., Shi, Y., Wei, Y., Luo, Q., & Tang, Z. (2023). Nature-inspired algorithms for 0-1 knapsack problem: A survey. Neurocomputing, 554, 126630. https://doi.org/10.1016/j.neucom.2023.126630