Application of Ant Colony Optimisation (ACO) to Security Van Problem

Abstract

The Security Van Problem is an NP-hard combinatorial optimization challenge. It involves selecting a subset of items to maximize total value while adhering to the weight constraints. This research provides fundamental knowledge to design a complete ACO algorithm to solve the problem and how to implement the algorithm step by step. Additionally, evaluations of experiments indicate that by modifying the combination of parameters, the ACO algorithm can achieve high-quality solutions in a reasonable time. Insights from the experiment suggest the next step is to find the correlation between the parameters to optimize fitness evaluation.

1 INTRODUCTION

The security van (SV) problem is a variant of the Knapsack Problem, classified as an impressive NP-hard problem in combinatorial optimization. The problem assumes a situation where we have to choose a set of bags to maximize the total value while adhering to a provided weight limit. This situation is often seen in many real-world cases such as telecommunication, manufacturing, and budgeting problems...

Over many years, a large number of studies have been conducted to improve methods for effectively solving this problem, in which many nature-inspired algorithms have been applied and have yielded valuable results. In this paper, I use the Ant Colony Optimization approach introduced by Dorigo, Maniezzo, and Colorni [1] to solve the security van problem.

2 LITERATURE REVIEW

This section will give you a glance at several nature-inspired approaches that could also have been used to solve the security van problem

2.1 Genetic Algorithm (GA)

The genetic algorithm is first suggested by John Holland in the 1960s [3]. GA is a subset of evolutionary algorithms that simulate the natural selection process and have been used effectively in a large number of optimization and search problems. Key elements of the genetic algorithm are constructed upon fundamental components inspired by biological evolution, which are:

- Populations of chromosomes: represent a group of candidate solutions to the problem.

- Selection according to fitness: picks chromosomes in the population for reproduction. If a chromosome is fitter, it is more likely to be selected for reproduction.

- Crossover to produce new offspring: randomly selects a locus and exchanges the subsequences.

- Random mutation of new offspring: randomly flips some of the bits in a chromosome, which might occur at each bit position in a string.

GA is a flexible algorithm and has the ability to adapt to domain knowledge by adjusting its operators. However, when applied to the SV problem, there are several limitations. For example, GA does not converge to the optimal solution without additional methods, which leads to more time needed to explore solutions. In addition, GA depends on parameter tuning, such as mutation rates or population size [4].

2.2 Particle Swarm Optimization (PSO)

PSO is known as a population-based, stochastic optimization algorithm inspired by the social behavior of animals such as insects, bird flocks, and fish schools [5]. The algorithm focuses on their collective behavior, such as the ability to cooperate and share information to optimize their search for food resources. This algorithm was proposed by Eberhart and Kennedy (1995) [2]. Below are the core components:

2.3 Ant Colony Optimization (ACO)

This algorithm is based on the foraging behavior of ants and the way each ant communicates with others. ACO uses artificial ants to simulate laying pheromone and following pheromone behaviors, leading to the ability to explore combinatorial optimization problems.

Like many nature-inspired metaheuristics, ACO has been used to solve various problems related to optimization and search problems. The core principles of ACO are:

Artificial pheromones: Artificial ants deposit pheromones on the edges of the constructing graph, impacting future path selections.

Probabilistic path selection: Ants choose paths probabilistically based on pheromone intensity and heuristic information.

Pheromone evaporation: By mitigating the influence of early paths, evaporation rate prevents premature convergence.

Positive update: By updating pheromones, artificial ants reinforce optimal solutions as more ants follow and strengthen promising paths.

Compared to Genetic Algorithms, ACO tends to converge to good results.

3 METHODOLOGY

This section mention the algorithm design for the ACO algorithm and then describe how this algorithm works for Security Van Problem. For simplicity, two subsections are added, namely, Creating a construction graph and Pseudocode for the proposed ACO algorithm respectively.

3.1 Creating a construction graph

The SVP is represented by a construction graph below:

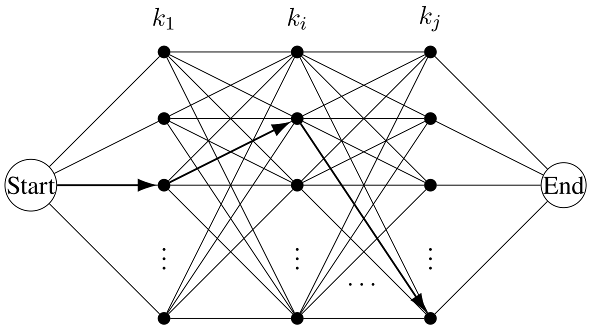

Figure 1: Ant colony optimization graph.

Each node in the graph represents a step (bag) that an ant can choose to move to. For example, the second node in column 2 means that an ant chose bag number 2 as its step in this round. The ant traverses the graph to construct a solution, and the solution is represented by the edges in the graph. In this problem, the ant does not need to go through all the columns as long as the total weight does not exceed the van's capacity.

3.2 Pseudocode for the ACO algorithm

The pseudocode is written to understand the algorithm quickly. This is the step by step explanation of the proposed algorithm. The pseudocode is as follows:

Algorithm 1 The ACO Metaheuristic algorithm

```
procedure ACO algorithm
    function Initialization
        Setting parameters
        Initialize the heuristic matrix ($n \times n$)
        Initialize the pheromone matrix ($n \times n$)
    end function

    while termination condition is not met do
        for each ant in ants do
            Put the ant on a randomly chosen first bag
            for each edge in a construction graph do
                Applying a stochastic transition rule to choose the next bag to move
            Calculate fitness function
            end for
        end for
        Update pheromone trail based on fitness
    end while
end procedure
```

Step 1: randomly initialize a potential solution for the parameters. I run an experiment with three parameters: number of ants (p), pheromone deposit rate (m), and evaporation rate (e) to evaluate the impact of each parameter on the algorithm and find the best result.

Initialize the heuristic matrix ($n \times n$): heuristic for each edge (η_{ij}) is represented by a formula:

$$\eta_{ij} = \frac{v_j}{w_j} \qquad (1)$$

where v_j is the bag's value (pounds) and w_j is the weight's bag (kilogam).

Initialize the pheromone matrix ($n \times n$): the initial pheromone amount (τ_{ij}) for each edge on the construction graph is set to the default value of 1.

Parameter settings	Value
Number of fitness evaluations	10000
Number of runs for all implementations	10
Number of ants (p)	{10, 30, 50, 70, 90}
Pheromone Deposit Rate (Q or m)	{0.1, 0.3, 0.5, 0.7, 0.9}
Evaporation Rate (e)	{0.5, 0.6, 0.7, 0.8, 0.9}
α	1
β	1

Table 1: Parameter settings

Step 2: construct ant solutions.

Each ant has a different starting point in the construction graph. While the ant selects items, those items are sequentially removed from the list of next nodes. The next node is selected with a probability:

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{h \in J_i^k} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta}, & \text{if } j \text{ has been visited,} \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Calculate fitness function (L_k): total value of bags (v_{ij}^k) found by the ant after constructing a solution (P_k).

$$L_k = \sum_{j \in N_i^k} v_{ij}^k \tag{3}$$

Update pheromone trail: for each ant path pheromone value is updated as describe below

$$\tau_{ij}(t+1) \leftarrow (1-\rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}^k(t) \tag{4}$$

where

$$\Delta\tau_{ij}^k(t) = \begin{cases} Q/L_k, & \text{if ant } k \text{ used edge } ij, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

4 DESCRIPTION OF RESULTS

To solve the SVP, I run the ACO algorithm, experimenting with three dynamic parameters, using five testing values for each parameter to create 125 combinations of parameters (5 $\times$ 5 $\times$ 5). Each combination runs through 10,000 fitness evaluations and best fitness ($L_{avg.best}$) is averaged over 10 runs for each implementation.

Generations	p	m	e	$L_{avg.best}$
1	10	0.1	0.6	4224.0
2	10	0.1	0.6	4224.0
3	10	0.1	0.6	4224.0
...	...	...	...	...
1	90	0.9	0.9	4216.0
2	90	0.9	0.9	4239.0
3	90	0.9	0.9	4255.0
...	...	...	...	...

Table 2: Parameter settings

5 DISCUSSION AND FURTHER WORK

Question 1: Which combination of parameters produces the best results?

Within the limitations of an experiment using 125 combinations of parameters, 4527 is the highest result recorded. The list of bags is shown below:

[(7.4, 94.0), (7.7, 59.0), (7.4, 83.0), (2.9, 82.0), (1.1, 91.0), (9.0, 84.0), (8.1, 85.0), (7.5, 94.0), (4.4, 31.0), (2.0, 31.0), (2.0, 42.0), (5.2, 55.0), (7.1, 97.0), (6.8, 79.0), (1.6, 10.0), (3.0, 100.0), (1.6, 98.0), (2.1, 19.0), (4.9, 77.0), (7.7, 60.0), (2.1, 22.0), (8.4, 84.0), (1.9, 89.0), (3.8, 46.0), (4.3, 85.0), (8.5, 94.0), (1.0, 65.0), (2.6, 34.0), (2.1, 27.0), (7.4, 91.0), (1.5, 17.0), (2.2, 56.0), (7.9, 89.0), (1.5, 18.0), (2.4, 91.0), (1.6, 79.0), (7.5, 99.0), (2.5, 45.0), (7.6, 73.0), (4.8, 81.0), (6.5, 96.0), (1.5, 51.0), (2.3, 96.0), (1.0, 63.0), (8.1, 93.0), (6.7, 87.0), (5.8, 71.0), (4.9, 74.0), (1.6, 15.0), (7.4, 57.0), (1.4, 70.0), (4.1, 62.0), (7.4, 71.0), (1.7, 57.0), (1.2, 97.0), (1.9, 33.0), (5.9, 59.0), (3.7, 91.0), (1.7, 81.0), (1.5, 60.0), (7.8, 90.0), (7.6, 87.0), (3.3, 76.0), (3.9, 76.0), (7.1, 59.0), (3.9, 40.0), (3.9, 59.0)].

There are many combinations of parameters reaching that result:

Question 2: What do you think is the reason for your findings in Question 1?

ACO is a meta-heuristic algorithm which is stochastic. Every time I run an implementation, each ant does not give the same selection. This means that the result is always different even using the same combination. As a result, a bad combination sometimes gives a good solution due to its stochastic nature.

In addition, ACO has a tendency to converge to

p	m	e	$L_{avg.best}$
10	0.1	0.5	4527.0
30	0.1	0.5	4527.0
30	0.1	0.7	4527.0
...	...	...	...
30	0.1	0.6	4527.0
90	0.9	0.8	4527.0

Table 3: Combination of parameters produces the best result

good results. This can be obviously seen in the logic of algorithm design in using heuristic information and updating pheromone trails. The ant colony always traverses in the good path with new generations. This leads to a situation where, after several tours, many combinations of parameters go to the best solution.

Question 3: How do each of the parameter settings influence the performance of the algorithm?

To understand how each parameter impacts the output, I tested several values of specific parameters while keeping others constant. The tested values were listed in the Methodology section.

With experiments of population size parameter, the results show that the increase in the number of ants leads to fewer generations to reach the best fitness. The graph indicates that with 10 ants, I need about 200 generations to get the same level as others. I also see that the average number of fitness evaluations needed to reach the optimal solution is approximately 2000. This observation gives me the idea of increasing the number of ants to save time spent on running the program. However, an optimal number to harmonize with other parameters still needs to be explored more. Alberto Colorni et al. (1992) previously indicated that an increase in the number of ants results in increased efficiency of the overall system.

In terms of the pheromone deposit rate parameter, it directly affects the learning ability and adaptation of the algorithm with the search space. If pheromone is deposited slowly on the path, the convergence of the algorithm would need more time. On the other hand, this would lead to more chances to explore the current search space because the probability of each solution is not too different.

Regarding the evaporation rate parameter, with a low value, pheromone would evaporate more slowly. This could help utilize the solutions ex-

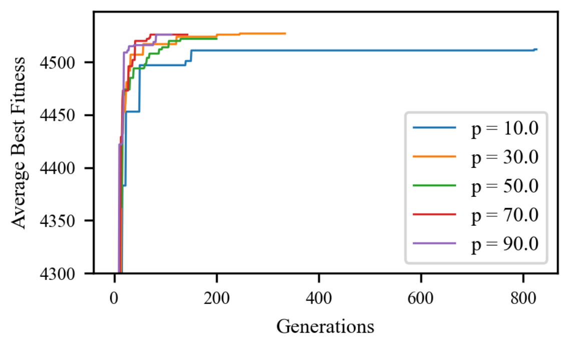

Figure 2: Experiment on the Population size Parameter (fixed m=0.5, e=0.6

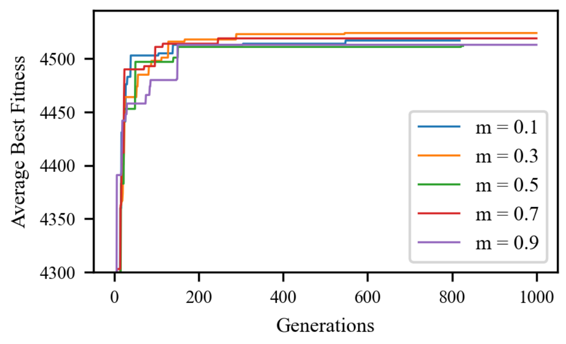

Figure 3: Experiment on the Pheramone deposit rate Parameter (fixed p=10, e=0.6)

plored before but easily risks getting stuck in local search. Conversely, a high value for this parameter means that pheromone would evaporate more quickly, hence the colony would promptly forget old paths and explore new solutions.

A third set of experiments was run to illustrate the effect of the evaporation rate parameter on the efficiency of the solving process. It is not easy to recognize a pattern like population size, making it difficult to know if an increase or decrease in this parameter will impact the system. The efficiency of the evaporation rate needs to be explored in correlation with other parameters instead of being evaluated independently.

Question 4: Do you think that one of the algorithms in your literature review might have provided better results? Explain your answer.

I personally believe that there is no single best algorithm to solve this problem. The reason is that each algorithm has its own advantages and disadvantages depending on the way it is implemented. Finding the right combination of parameters plays a major role in the success of the algorithm, and

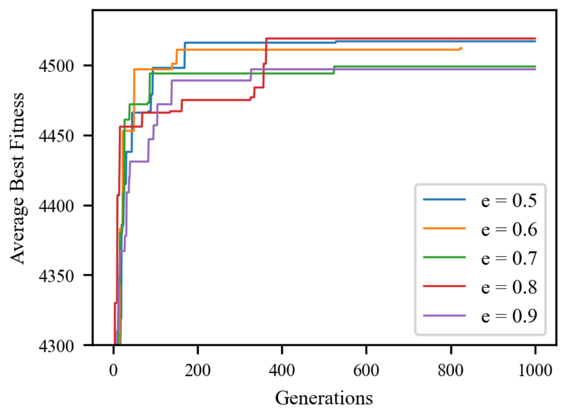

Figure 4: Experiment on the Evaporation rate Parameter (fixed p=10, m=0.5)

these parameters vary with different problems.

6 CONCLUSION

In conclusion, this research paper provides a glance through several nature-inspired algorithms to solve the Security Van Problem. Besides, it also provides a robust foundation for understanding the Ant Colony Optimization Algorithm and successfully applies this to tackle the problem. Implementing several combinations of parameters illustrated how each parameter setting influences the performance of the algorithm to create an efficient optimization technique.

Future work may focus on finding the correlation between each parameter to optimize the performance of the algorithm.

References

[1] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. "Ant system: optimization by a colony of cooperating agents". In: *IEEE transactions on systems, man, and cybernetics, part b (cybernetics)* 26.1 (1996), pp. 29–41.

[2] Russell Eberhart and James Kennedy. "A new optimizer using particle swarm theory". In: *MHS'95. Proceedings of the sixth international symposium on micro machine and human science*. Ieee. 1995, pp. 39–43.

[3] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[4] Richard Spillman. "Solving large knapsack problems with a genetic algorithm". In: *1995 IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century*. Vol. 1. IEEE. 1995, pp. 632–637.

[5] Dongshu Wang, Dapei Tan, and Lei Liu. "Particle swarm optimization algorithm: an overview". In: *Soft computing* 22.2 (2018), pp. 387–408.