COMPUTATIONAL INTELLIGENCE OPTIMIZATION (CIO)

(CT099-3-M-CIO)

INDIVIDUAL ASSIGNMENT PART 2

Title	**:**	A Hybrid Model of Genetic Algorithm and Ant Colony Optimization for Solving 0-1 Knapsack Problem
Intake Code	**:**	APUMF2006AI(PR)
Student ID	**:**	TP061241
Student Name	**:**	LAI KAI LOK
Module Lecturer	**:**	DR. IMRAN MEDI
Submission Date	**:**	24th January 2021

A Hybrid Model of Genetic Algorithm and Ant Colony Optimization for Solving 0-1 Knapsack Problem

Lai Kai Lok
School of Computing
Asia Pacific University
Kuala Lumpur, Malaysia
tp061241@mail.apu.edu.my

Imran Medi
School of Computing
Asia Pacific University
Kuala Lumpur, Malaysia
dr.imran.medi@apu.edu.my

***Abstract*—Genetic algorithm is efficient in solving simple optimization problems as it takes relatively less processing time compared to other algorithms and is easy to implement. However, it tends to converge prematurely when it comes to solving complex problems. Ant colony optimization is able to provide better solutions compared to genetic algorithm for most of the time but it is lacking diversity. It tends to focus on searching the existing good quality solution space without considering other possibilities, which can cause trapping in the local minimum. A hybrid model which combines the guided search capability from the ant colony optimization and the diversity characteristic of the mutation process from the genetic algorithm is proposed to solve the 0-1 knapsack problem. The 0-1 knapsack problem has various real-world applications, which are mainly related to finding the best way to allocate different resources to achieve maximum profit or value. Improving the performance of searching a better solution for the 0-1 knapsack problem directly translates to getting a more profitable solution in different domains. The proposed hybrid model is showing promising results in getting a better solution especially for the case of solving complex 0-1 knapsack problems.**

Keywords— ant colony optimization, hybrid model, genetic algorithm, 0-1 knapsack problem

I. INTRODUCTION

Optimization problem is formulated by using a set of parameters and constraints, so that techniques can be applied to search for the best solution among other possible solutions by varying the variables without violating the constraints [1]. Various optimization techniques are being applied in different domains such as the finance domain, manufacturing domain and engineering domain to search for the best solution according to specific objectives. As most of the real world optimization problems are NP-hard, where the polynomial-time solutions cannot be determined, it is not viable to use exact algorithms like the dynamic programming here due to time limitation. Thus, to search for solutions within a reasonable amount of time, metaheuristics algorithms such as the genetic algorithm and the ant colony optimization are widely used [2].

Genetic algorithm falls under the evolutionary algorithm category, inspired by Charles Darwin's theory of natural selection [3]. In the natural selection theory, better individuals will be able to preserve and survive, passing the good genes to the next generation. Genetic algorithm is relatively simple and consume less time compared to ant colony optimization. It can be adjusted easily to solve various complex problems and search the solution space in parallel by using different offspring. However, it tends to trap in the local minimum. Furthermore, genetic algorithm is also facing a redundancy issue, where it may search for the same solution space again and again, wasting resources [4]. The general flow of genetic algorithm is shown in Figure 1 below. Selection, crossover and mutation are the three main processes that affect the performance of the genetic algorithm. Selection is about selecting the right pair of parents for generating new offspring during the crossover process while mutation introduces diversity to the population, reducing the chance of premature convergence.

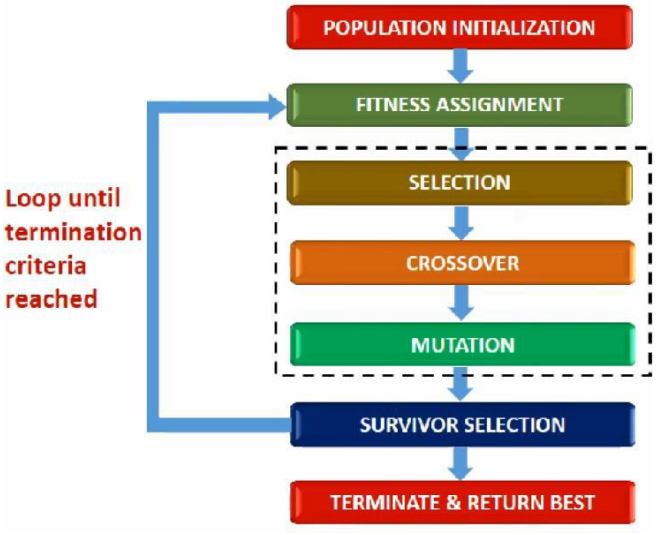

Fig. 1. The flow of genetic algorithm [5].

Ant colony optimization mimics the ants' foraging process, which is guided by the pheromone concentration. Pheromone is a scent chemical left by ants while moving and ants prefer to follow the path with higher pheromone concentration. Initially, there are multiple routes with the same amount of pheromone concentration to reach the food

source. As the ants which took the shortest path are able to reach the destination and return using a shorter time period, the pheromone concentration on the path will be higher compared to other routes. This further increases the likelihood of other ants to select this path and ultimately becoming the favoured path. The organised behaviour is mainly guided by the pheromone concentration without any master ant controlling the whole process [3].The foraging process of ants is shown in Figure 2.

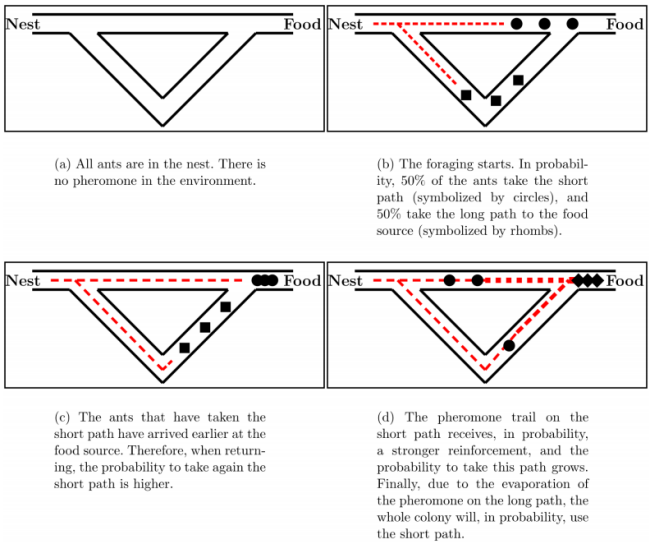

Fig. 2. Illustration of ants' foraging process [6].

The ant colony optimization is mainly guided by two functions, which are the attractiveness of each component and also the pheromone trail. Initially, the possible solutions are generated by using the attractiveness function and a constant pheromone trail. Then, the solutions formed are used to update the concentration of the pheromone trail. In the next iteration, the generation of new possible solutions are guided by the attractiveness function and the updated pheromone trail. This process is repeated until the solutions converge. The update of the pheromone trail is to focus on searching the space with high quality solutions [6].

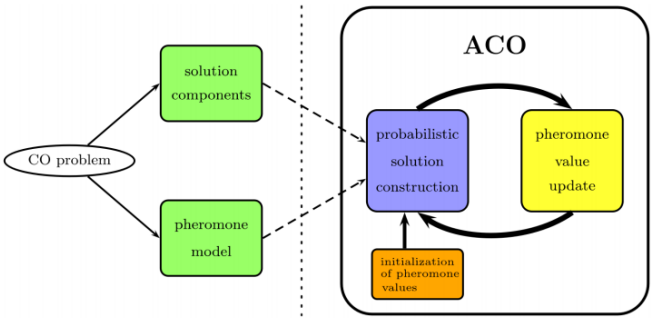

Fig. 3. The working of ant colony optimization [6].

The main purpose of this study is to improve the performance of solving the 0-1 Knapsack problem in terms of the ability to find a better solution, reducing the likelihood of premature convergence. This is done by developing a hybrid model of genetic algorithm and ant colony optimization so that the advantages of both models can be enhanced. Improving the performance of solving 0-1 knapsack problems translates to a better solution in real world applications with higher gain using the same or lower cost. This is because many real life applications like the investment portfolio and shipping goods management can be represented by using the 0-1 knapsack problem [7].

The remainder of this paper is arranged as follows. In Section II, the basic concepts and problem model of the 0-1 Knapsack problem are presented. Section III describes the implementation of the genetic algorithm, ant colony optimization and the hybrid model for solving 0-1 Knapsack problem. The results and performance comparison of the 3 algorithms are shown in Section IV. The conclusion and future research are discussed in the final Section.

II. 0-1 KNAPSACK PROBLEM

When a mountain climber is planning for a trip, he can only select a limited number of items to be included in his knapsack. This situation is where the name, "Knapsack Problem" originated from. In the classic knapsack problem, there are several items with unique weight and profit value. The knapsack has a fixed capacity. The main purpose is to include a combination of items which maximize the total profit while the total weight cannot exceed the capacity of the knapsack [7].

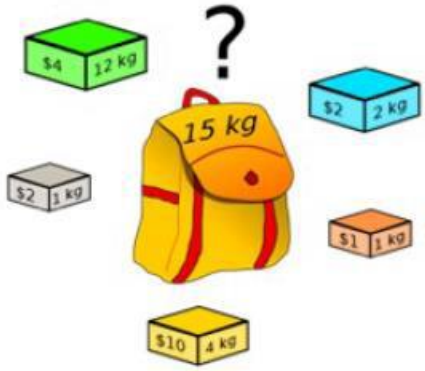

Fig. 4. Illustration of Knapsack Problem [8].

A. *Problem Model of 0-1 Knapsack Problem*

The 0-1 knapsack problem lists the solution in binary form, where 1 represents the item is included in the knapsack while 0 indicates it is excluded. For n total number of available items, the length of the solution is equal to n as well, where each component in the solution is represented by its status x_i, either 1 or 0. Each item i among the total n number of available items is unique and associated with a weight w_i and a profit value p_i. The capacity, C of the knapsack is the constraint of the problem where the total weight of selected items cannot exceed this value. For all the items with its status $x_i = 1$, which means it is included in the knapsack, the total profit and the total weight will be computed. The main purpose is to search for the best solution where the combination of items maximizes the total profit and does not violate the capacity constraint. The problem model of the 0-1 knapsack problem is represented in equation (1) – (3) [9].

$$Max \sum_i^n p_i x_i \quad (1)$$

$$Constraint \sum_i^n w_i x_i \leq C \quad (2)$$

$$x_i \in \{0,1\} \quad (3)$$

B. 0-1 Knapsack Problem dataset

Four standard datasets are obtained from a knapsack problem website [10] with the items length range from 7 to 24 and one dataset with items length of 50 are randomly generated. These 5 datasets are used to evaluate the performance of the genetic algorithm, ant colony optimization and the hybrid model. The summary of the dataset used in this study is shown in Table I.

III. IMPLEMENTATION

In this section, the major part of the code implementation for the three algorithms are discussed. For each of the possible solutions, the fitness is computed by summing up the values of the selected items. If the total weight of the selected items exceeds the knapsack capacity, a penalty is applied and the fitness value will be zero, else, the computed fitness value will be returned. Figure 5 shows the function coded to compute the fitness value.

```
def cal_fitness(sol, weights, values, capacity):
    total_weight = 0
    total_value = 0
    for i in range(len(sol)):
        if sol[i] == 1:
            total_weight += weights[i]
            total_value += values[i]
    if total_weight > capacity:
        return 0
    else:
        return total_value
```

Fig. 5. Function to calculate the fitness

A. Genetic Algorithm

After selecting the problem set and defining all the necessary constants, 5 solutions are generated as the initial population. This initial population is generated randomly without violating the weight constraint of the knapsack as shown in Figure 6.

```
def gen_initial_pop(self):
    population = []
    while len(population) < self.pop_size:
        indi_weight = self.prob_set.capacity + 1
        while indi_weight > self.prob_set.capacity:
            indi_sol = [random.randint(0, 1) for x in range(0, len(self.prob_set.weights))]
            indi_weight = 0
            for i in range(len(indi_sol)):
                if indi_sol[i] == 1:
                    indi_weight += self.prob_set.weights[i]
            if indi_weight <= self.prob_set.capacity:
                population.append(indi_sol)
    return population
```

Fig. 6. Function to generate initial population

TABLE I. SUMMARY OF KNAPSACK PROBLEM DATASET

Dataset Description	
Problem Set 01:	
Item length	7
Knapsack capacity	170
Item weights	[41, 50, 49, 59, 55, 57, 60]
Item values	[442, 525, 511, 593, 546, 564, 617]
Optimum solution	[0, 1, 0, 1, 0, 0, 1]
Problem Set 02:	
Item length	10
Knapsack capacity	165
Item weights	[23, 31, 29, 44, 53, 38, 63, 85, 89, 82]
Item values	[92, 57, 49, 68, 60, 43, 67, 84, 87, 72]
Optimum solution	[1, 1, 1, 1, 0, 1, 0, 0, 0, 0]
Problem Set 03:	
Item length	15
Knapsack capacity	750
Item weights	[70, 73, 77, 80, 82, 87, 90, 94, 98, 106, 110, 113, 115, 118, 120]
Item values	[135, 139, 149, 150, 156, 163, 173, 184, 192, 201, 210, 214, 221, 229, 240]
Optimum solution	[1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1]
Problem Set 04:	
Item length	24
Knapsack capacity	6404180
Item weights	[382745, 799601, 909247, 729069, 467902, 44328, 34610, 698150, 823460, 903959, 853665, 551830, 610856, 670702, 488960, 951111, 323046, 446298, 931161, 31385, 496951, 264724, 224916, 169684]
Item values	[825594, 1677009, 1676628, 1523970, 943972, 97426, 69666, 1296457, 1679693, 1902996, 1844992, 1049289, 1252836, 1319836, 953277, 2067538, 675367, 853655, 1826027, 65731, 901489, 577243, 466257, 369261]
Optimum solution	[1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1]
Problem Set 05:	
Item length	50
Knapsack capacity	Randomly generated by dividing the total weighs by 3.
Item weights	Randomly generated range from 10 to 500
Item values	Randomly generated range from 10 to 500
Optimum solution	Unknown

The selection process is done by using the tournament method. Two solutions are randomly chosen from the population and the one with better fitness will be selected to proceed to the crossover process. Figure 7 shows the code for the tournament selection process. For the crossover process, the ordered crossover method [11] is applied.

```
def tournament(self, pop):
    p1 = list(random.choice(pop)).copy()
    p2 = list(random.choice(pop)).copy()
    p1fit = ks.cal_fitness(p1, self.prob_set.weights, self.prob_set.values, self.prob_set.capacity)
    p2fit = ks.cal_fitness(p2, self.prob_set.weights, self.prob_set.values, self.prob_set.capacity)
    if p1fit >= p2fit:
        return p1
    return p2
```

Fig. 7. Function for tournament selection process

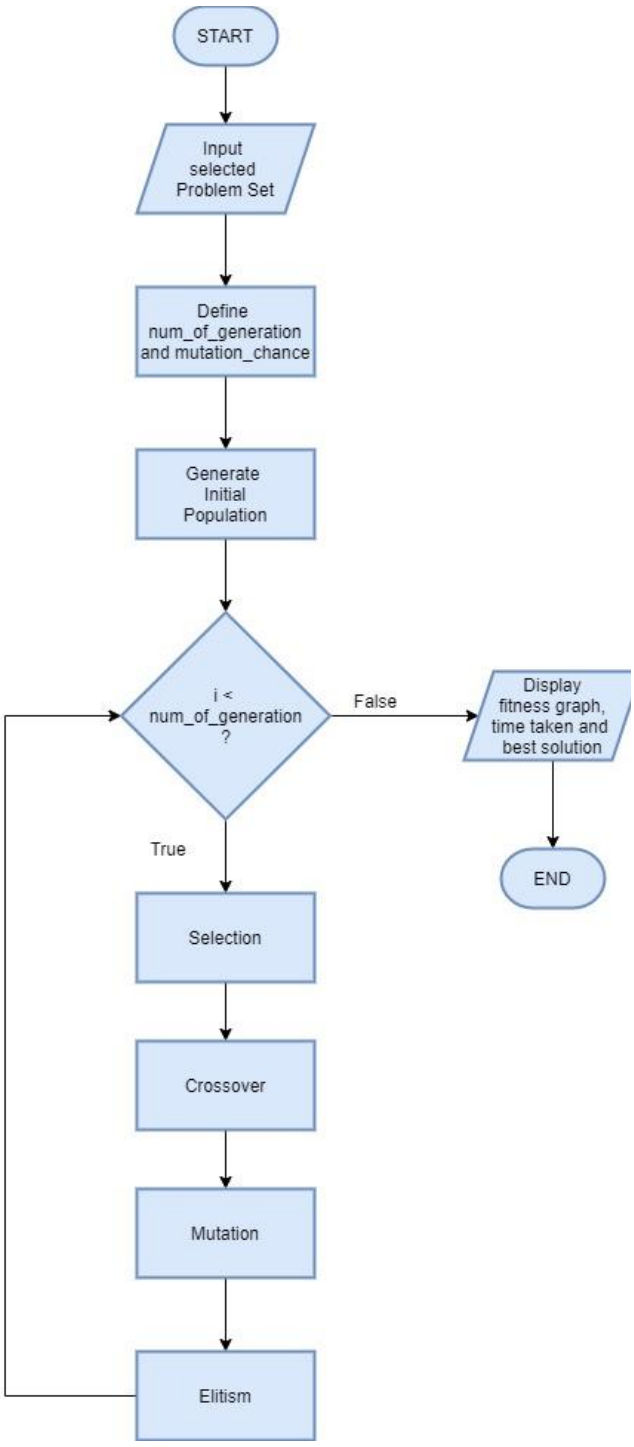

Fig. 8. The flowchart of genetic algorithm.

The mutation process is conducted by randomly selecting 2 components in the solution and flip its value as shown in Figure 9. All the solutions are then combined and sorted by using its fitness value. Elitism method is then applied to only keep the top 5 solutions and move to the next generation as shown in Figure 10. The selection, crossover, mutation and elitism processes are repeated until the generation reaches 100. The overall flow of the genetic algorithm is shown in Figure 8.

```
def mutation(self, chromo):
    num_mutation_chromo = 2
    for i in range(num_mutation_chromo):
        r = random.randint(0, len(chromo) - 1)
        if chromo[r] == 1:
            chromo[r] = 0
        else:
            chromo[r] = 1
    return chromo
```

Fig. 9. Function for mutation process

```
pwf = self.pop_with_fit(all_pop)
sort_pwf = sorted(pwf, key=lambda x: x[1], reverse=True)

fit_progress.append(sort_pwf[0][1])

# elitism: only selct the top 5 as new pop
initial_ga_pop.clear()
for k in range(5):
    initial_ga_pop.append(list(sort_pwf[k][0]))
```

Fig. 10. Elitsm process

B. *Ant Colony Optimization*

The attractiveness function, μ_i for item i and the pheromone update function $\Delta\tau$ are defined initially by using equation (4) and (5) below. The probability of selecting an item from a list of possible items is computed by using equation (6) [12].

$$\mu_i = \frac{z_i}{w_i^2} \tag{4}$$

$$\Delta\tau = \frac{1}{1+\frac{z_{best}-z_{current}}{z_{best}}} \tag{5}$$

$$P_i = \frac{\tau_i^\alpha \mu_i^\beta}{\sum_i^n \tau_i^\alpha \mu_i^\beta} \ , \ i = 1, 2, \ldots, n \tag{6}$$

where z_i is the profit or value of item i and w_i is its weight. z_{best} is the total profit of the best solution white $z_{current}$ is the total profit of the current solution. τ is the pheromone concentration of the item. Constant α and β indicate the importance of τ and μ respectively.

The initial values of the pheromone trail are all set as 1 while the score value of each item is calculated by using equation (4) as shown in Figure 11.

```
for i in range(len(self.prob_set.weights)):
    self.phe.append(1)
    score = self.prob_set.values[i]/self.prob_set.weights[i]**2
    self.itemScore.append(score)
```

Fig. 11. Initialize the score values and the pheromone values

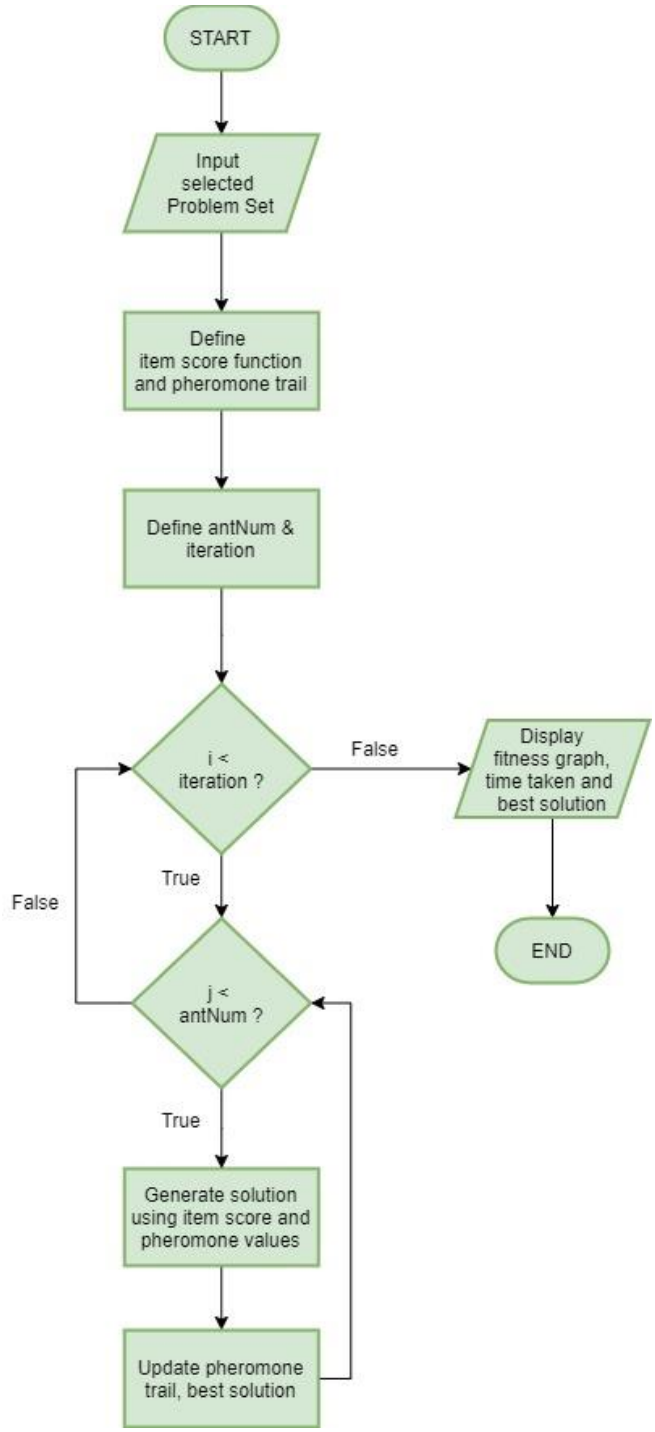

Fig. 12. The flowchart of ant coloy optimization.

The probability of each item to be selected is encoded by using equation (6) as shown in Figure 13.

```
def cal_probability(self, possible_weight_list):
    prob_value = [self.phe[i]**self.beta*self.itemScore[i]**self.alpha
                  for i in range(len(self.phe)) if self.prob_set.weights[i] in possible_weight_list]
    prob_list = [x/sum(prob_value) for x in prob_value]
    return prob_list
```

Fig. 13. Function to calculate the probability of each possible item

When generating a solution, each time an item is selected based on the probability, the capacity left and the list of possible items will be updated. This process is repeated until there is no capacity left or there is no possible item left. The generate_solution function is shown in Figure 14.

```
def generate_solution(self):
    solution = []
    selected_weight = []
    possible_weight_list = list(self.prob_set.weights).copy()
    capacity_left = int(self.prob_set.capacity)
    while (capacity_left > 0 and len(possible_weight_list) != 0):
        prob_l = self.cal_probability(possible_weight_list)
        weight_pick = self.selectObject(prob_l, possible_weight_list)
        selected_weight.append(weight_pick)
        capacity_left -= weight_pick
        possible_weight_list.remove(weight_pick)
        possible_weight_list = [x for x in possible_weight_list if x <= capacity_left]

    for i in self.prob_set.weights:
        if i in selected_weight:
            solution.append(1)
        else:
            solution.append(0)

    profit = ks.cal_fitness(solution, self.prob_set.weights, self.prob_set.values, self.prob_set.capacity)

    return solution, profit
```

Fig. 14. Function to generate a solution

After each solution is generated, the pheromone trail will be updated by using equation (5) and an evaporation rate of 0.95 as shown in Figure 15. This updated pheromone trail will be used to compute the probability of each item to be selected in the next iteration. Figure 12 showed the overall flow of the ant colony optimization.

```
def update_phenomone(self, solution, profit, best_profit):
    delta_phe = 1 / (1 + (best_profit - profit) / best_profit)
    evaporate_C = 0.95
    for i in range(len(solution)):
        self.phe[i] *= evaporate_C
        if solution[i] == 1:
            self.phe[i] += delta_phe
```

Fig. 15. Function to update the pheromone trail

C. Hybrid Model

In this hybrid model, the structure and flow of the algorithm is mainly using the structure of the genetic algorithm. The attractiveness function and the pheromone trail function from the ant colony optimization are incorporated into the population initialization process, the crossover process and the mutation process.

While generating the initial population, rather than randomly selecting an item, it is guided by the attractiveness function from equation (4). After computing the score value of each item, these values are then used to calculate the

probability of each item to be selected. The initial pheromone values are all set as 1.

```
    for i in range(len(self.prob_set.weights)):
        self.phe.append(1)
        score = self.prob_set.values[i] / self.prob_set.weights[i] ** 2
        self.itemScore.append(score)

def cal_IS_probability(self, possible_weight_list):
    prob_value = [self.itemScore[i] ** self.alpha for i in range(len(self.phe)) if
                  self.prob_set.weights[i] in possible_weight_list]
    prob_list = [x / sum(prob_value) for x in prob_value]
    return prob_list
```

Fig. 16. Define the initial pheromone values and item score values. The item score values are used to compute the probability for generating initial population.

The structure of generating a solution in the ant colony optimization is applied here to generate the initial population. However, in this hybrid model, only the attractiveness function is used to compute the probability of each item to be selected. The function to generate the initial population is shown in Figure 17 below. The purpose of using attractiveness function to generate the initial population is to create good quality solutions right from the beginning of the process.

```
def gen_initial_pop(self):
    population = []
    while len(population) < self.pop_size:
        solution = []
        selected_weight = []
        possible_weight_list = list(self.prob_set.weights).copy()
        capacity_left = int(self.prob_set.capacity)
        while (capacity_left > 0 and len(possible_weight_list) != 0):
            prob_l = self.cal_IS_probability(possible_weight_list)
            weight_pick = self.selectObject(prob_l, possible_weight_list)
            selected_weight.append(weight_pick)
            capacity_left -= weight_pick
            possible_weight_list.remove(weight_pick)
            possible_weight_list = [x for x in possible_weight_list if x <= capacity_left]

        for i in self.prob_set.weights:
            if i in selected_weight:
                solution.append(1)
            else:
                solution.append(0)

        population.append(solution)
    return population
```

Fig. 17. Generating initial population in the hybrid model.

The pheromone concentration values are used to calculate the probability of an item to be selected as part of the solution during the crossover process and the mutation process. Figure 18 shows the code for calculating the probability by using pheromone values.

```
def cal_phe_probability(self, possible_weight_list):
    prob_value = [self.phe[i] ** self.beta for i in range(len(self.phe)) if
                  self.prob_set.weights[i] in possible_weight_list]
    prob_list = [x / sum(prob_value) for x in prob_value]
    return prob_list
```

Fig. 18. Function to calculate the probability to be selected for each possible item using pheromone values.

Two solutions will be randomly selected from the population for the crossover process. At the end of the first generation, all solutions are ranked and only the top 5 will proceed to the next generation. Thus, the elitism method is mainly applied in the selection process. During the crossover process, each item will be selected from each parent consecutively until there is no capacity left or there is no possible item left from each parent to fill in the capacity. The items to be selected from each parent to form a new offspring is done by using the roulette wheel selection method, based on the probability obtained from pheromone concentration. If the fitness of the offspring is better than the parent, the offspring solution will be used to update the pheromone trail the same way as in ant colony optimization. The code to generate new offspring in the crossover process is shown in Figure 19.

```
def phe_guide_new_gen(self, pop):
    n_ge = []
    p1, p2 = self.random_select_parent(pop)
    p1Weight = [self.prob_set.weights[i] for i in range(len(p1)) if p1[i] == 1]
    p2Weight = [self.prob_set.weights[i] for i in range(len(p2)) if p2[i] == 1]

    selected_weight = []
    possible_p1weight_list = p1Weight.copy()
    possible_p2weight_list = p2Weight.copy()
    capacity_left = int(self.prob_set.capacity)
    while (capacity_left > 0):
        # select one from p1
        if len(possible_p1weight_list) != 0:
            prob_l = self.cal_phe_probability(possible_p1weight_list)
            weight_pick = self.selectObject(prob_l, possible_p1weight_list)
            selected_weight.append(weight_pick)
            capacity_left -= weight_pick
            possible_p1weight_list.remove(weight_pick)
            possible_p1weight_list = [x for x in possible_p1weight_list if x <= capacity_left]
            possible_p2weight_list = [x for x in possible_p2weight_list
                                      if (x <= capacity_left and x not in selected_weight)]


        # select one from p2
        if len(possible_p2weight_list) != 0:
            prob_l = self.cal_phe_probability(possible_p2weight_list)
            weight_pick = self.selectObject(prob_l, possible_p2weight_list)
            selected_weight.append(weight_pick)
            capacity_left -= weight_pick
            possible_p2weight_list.remove(weight_pick)
            possible_p2weight_list = [x for x in possible_p2weight_list if x <= capacity_left]
            possible_p1weight_list = [x for x in possible_p1weight_list
                                      if (x <= capacity_left and x not in selected_weight)]

        if len(possible_p1weight_list) == 0 and len(possible_p2weight_list) == 0:
            capacity_left = 0

    for i in self.prob_set.weights:
        if i in selected_weight:
            n_ge.append(1)
        else:
            n_ge.append(0)

p1fit = ks.cal_fitness(p1, self.prob_set.weights, self.prob_set.values, self.prob_set.capacity)
p2fit = ks.cal_fitness(p2, self.prob_set.weights, self.prob_set.values, self.prob_set.capacity)
n_gefit = ks.cal_fitness(n_ge, self.prob_set.weights, self.prob_set.values, self.prob_set.capacity)

# update phetomone if child is better than parent
if n_gefit > p1fit and n_gefit > p2fit:
    self.update_phenomone(n_ge)

return n_ge
```

Fig. 19. Crossover process in the hybrid model.

For the mutation process, those items not included in the existing solution are all placed under a possible solution list. Then, the item with the heaviest weight in the solution is removed and the possible items list is updated with only those items that are less than or equal to the capacity. Items will be selected from the possible items list until there is no capacity left or there is no possible item left, mainly guided by the pheromone concentration values. The code for the mutation process is shown in Figure 20. In the selection process, if both selected parents are exactly the same, the mutation process will be applied to the second parent to

introduce diversity into the offspring. Figure 21 shows the overall flow of the proposed hybrid model.

```
def mutation(self, chromo):
    # guided mutation
    capacity_left = int(self.prob_set.capacity)
    selected_weights = [self.prob_set.weights[i] for i in range(len(chromo)) if chromo[i] == 1]
    possible_weight_list = [x for x in self.prob_set.weights if x not in selected_weights]
    largest_num = max(selected_weights)
    selected_weights.remove(largest_num)
    capacity_left -= sum(selected_weights)
    possible_weight_list = [x for x in possible_weight_list if x <= capacity_left]
    while len(possible_weight_list) != 0:
        prob_l = self.cal_phe_probability(possible_weight_list)
        weight_pick = self.selectObject(prob_l, possible_weight_list)
        selected_weights.append(weight_pick)
        capacity_left -= weight_pick
        possible_weight_list = [x for x in possible_weight_list if x <= capacity_left]

    mutated_chromo =[]
    for i in self.prob_set.weights:
        if i in selected_weights:
            mutated_chromo.append(1)
        else:
            mutated_chromo.append(0)

    return mutated_chromo
```

Fig. 20. Mutation process in the hybrid model.

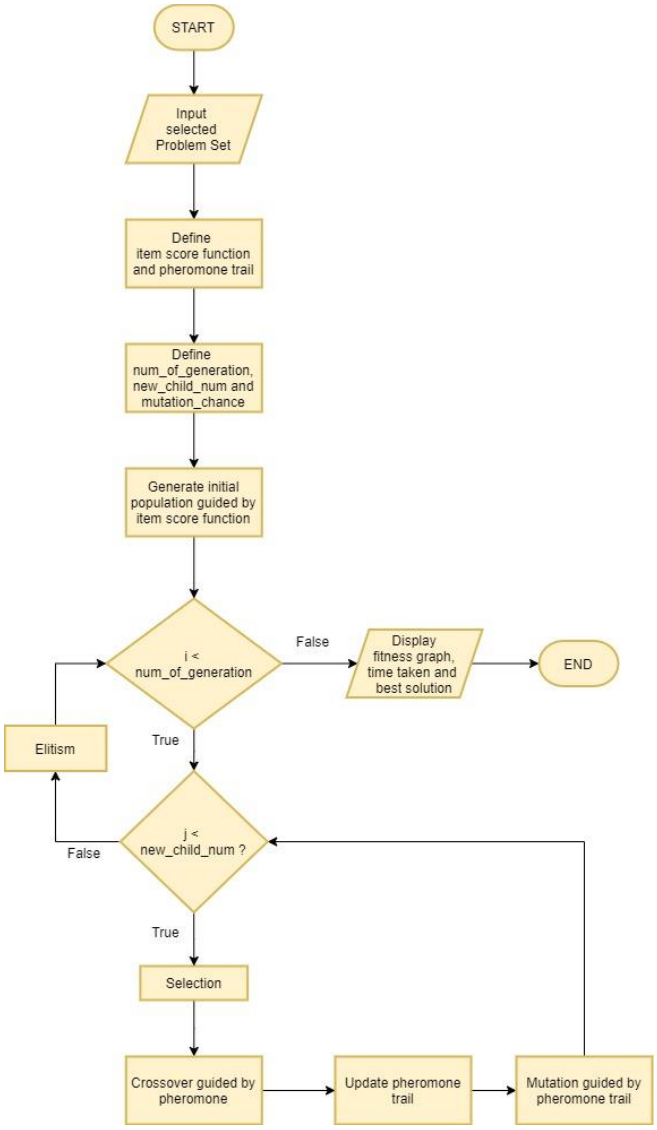

Fig. 21. The flowchart of hybrid model.

IV. PERFORMANCE EVALUATION

In this section, 5 different problem sets are being used to evaluate the performance of the genetic algorithm, ant colony optimization and the hybrid model in terms of time taken and also finding the best solution. Each problem set will be repeated for 10-fold and the average will be computed.

The parameters for the three algorithms are kept as constant throughout the 5 problem sets. The number of generation or iteration are all set as 100 while the initial population or initial ants number are all set as 5. For the genetic algorithm and the hybrid model, the mutation chance is set as 0.2. The α value is set as 0.7 while the β value is set as 0.8 in the ant colony optimization. For the hybrid model, both the α and β values are set as 1. This is because the attractiveness and the pheromone value are applied in different processes. The evaporation rate for both ant colony optimization and the hybrid model are set as 0.95.

A. *Problem Set 01*

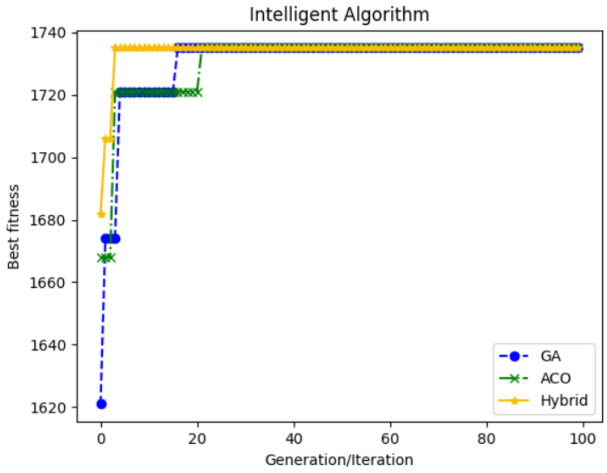

Fig. 22. Graph of best fitness against generation or iteration for one of the example in problem set 01.

TABLE II. SUMMARY OF PERFORMANCE OF THE 3 ALGORITHMS IN PROBLEM SET 01

Algorithm	No	Time Taken (s)	Best Fitness Value
Genetic Algorithm	1	0.03123	1735
	2	0.01562	1735
	3	0.01564	1735
	4	0.03125	1735
	5	0.03125	1735
	6	0.03125	1735
	7	0.01562	1735
	8	0.03125	1735
	9	0.03127	1735
	10	0.01563	1735
	Ave	**0.025001**	**1735**

Ant Colony Optimization	1	0.03125	1735
	2	0.03123	1735
	3	0.03123	1735
	4	0.03123	1735
	5	0.03125	1735
	6	0.01563	1735
	7	0.03125	1735
	8	0.03127	1735
	9	0.03125	1735
	10	0.0156	1735
	Ave	**0.028119**	**1735**
Hybrid Model	1	0.21877	1735
	2	0.25001	1735
	3	0.21877	1735
	4	0.21875	1735
	5	0.23438	1735
	6	0.20311	1735
	7	0.21875	1735
	8	0.20312	1735
	9	0.23439	1735
	10	0.21877	1735
	Ave	**0.221882**	**1735**

For the simple problem like in the case of problem set 01, all the three algorithms are able to obtain the best solution. Thus, for a simple optimization problem, genetic algorithm is the best choice among the three algorithms as it uses relatively less time.

B. Problem Set 02

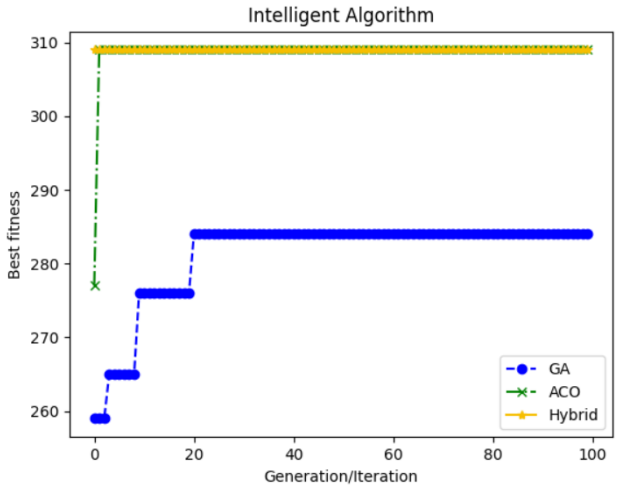

Fig. 23. Graph of best fitness against generation or iteration for one of the example in problem set 02.

TABLE III. SUMMARY OF PERFORMANCE OF THE 3 ALGORITHMS IN PROBLEM SET 02

Algorithm	No	Time Taken (s)	Best Fitness Value
Genetic Algorithm	1	0.03125	247
	2	0.03125	284
	3	0.03125	284
	4	0.03123	309
	5	0.03127	284
	6	0.01562	309
	7	0.03125	309
	8	0.03127	284
	9	0.03123	247
	10	0.03125	284
	Ave	**0.029687**	**284.1**
Ant Colony Optimization	1	0.0625	309
	2	0.07813	309
	3	0.04688	309
	4	0.04687	309
	5	0.0625	309
	6	0.04687	309
	7	0.04686	309
	8	0.04686	309
	9	0.03125	309
	10	0.04685	309
	Ave	**0.051557**	**309**
Hybrid Model	1	0.26563	309
	2	0.28126	309
	3	0.25	309
	4	0.23438	309
	5	0.25	309
	6	0.20313	309
	7	0.20313	309
	8	0.20315	309
	9	0.1875	309
	10	0.20313	309
	Ave	**0.228131**	**309**

For a slightly more complex problem like in the case of problem set 02, genetic algorithms have the tendency to get stuck in the local minimum while both the ant colony optimization and the hybrid model are able to search for the best solution at the end of the iteration. Both the ant colony optimization and hybrid model are able to create better quality solutions at the beginning of the stage due to a guided search algorithm. This can be observed in Figure 23.

C. Problem Set 03

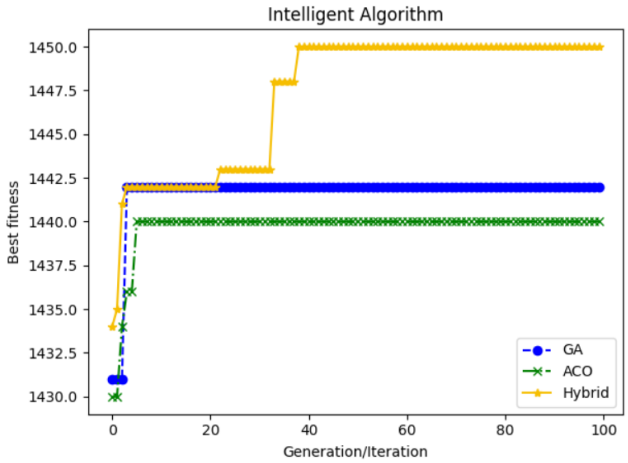

Fig. 24. Graph of best fitness against generation or iteration for one of the example in problem set 03.

TABLE IV. SUMMARY OF PERFORMANCE OF THE 3 ALGORITHMS IN PROBLEM SET 03

Algorithm	No	Time Taken (s)	Best Fitness Value
Genetic Algorithm	1	0.03125	1447
	2	0.03125	1441
	3	0.04687	1454
	4	0.03125	1447
	5	0.03127	1442
	6	0.03127	1448
	7	0.03125	1449
	8	0.04687	1444
	9	0.04688	1441
	10	0.04686	1445
	Ave	**0.037502**	**1445.8**
Ant Colony Optimization	1	0.125	1458
	2	0.12497	1447
	3	0.10938	1451
	4	0.10938	1449
	5	0.1094	1440
	6	0.1094	1458
	7	0.12498	1446
	8	0.10938	1442
	9	0.10938	1455
	10	0.10937	1458
	Ave	**0.114064**	**1450.4**
Hybrid Model	1	0.45315	1448
	2	0.4219	1446
	3	0.42188	1458
	4	0.42188	1451
	5	0.42188	1450
	6	0.42186	1450
	7	0.42188	1450
	8	0.4375	1458
	9	0.43751	1440
	10	0.4219	1446
	Ave	**0.428134**	**1449.7**

For problem set 03, ant colony optimization is able to give the overall best fitness value. Furthermore, it is using less time compared to the hybrid model. All the three algorithms are trapped in the local minimum for most of the test cases. However, the differences of the fitness value among the three algorithms are quite small. Therefore, time taken should be the main focus here and the genetic algorithm is still the best choice.

D. Problem Set 04

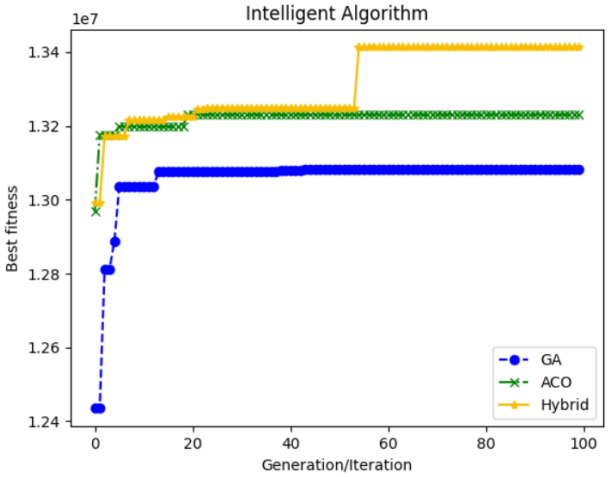

Fig. 25. Graph of best fitness against generation or iteration for one of the example in problem set 04.

TABLE V. SUMMARY OF PERFORMANCE OF THE 3 ALGORITHMS IN PROBLEM SET 04

Algorithm	No	Time Taken (s)	Best Fitness Value
Genetic Algorithm	1	0.04685	13083461
	2	0.0625	12995562
	3	0.06249	13122562
	4	0.04687	13104080
	5	0.06252	13272199
	6	0.0625	12968671
	7	0.06249	13219032
	8	0.0469	13015122
	9	0.0625	13259654
	10	0.06248	13270514
	Ave	**0.05781**	**13131085.7**
Ant Colony Optimization	1	0.32813	13231254
	2	0.31252	13332957
	3	0.35938	13377961
	4	0.3125	13473482

	5	0.34373	13311003
	6	0.34374	13271998
	7	0.32813	13341675
	8	0.32811	13264009
	9	0.31248	13368521
	10	0.3125	13410139
	Ave	**0.328122**	**13338299.9**
Hybrid Model	1	0.89066	13412823
	2	0.82814	13378270
	3	0.84378	13412823
	4	0.81253	13264596
	5	0.79691	13441033
	6	0.81253	13412823
	7	0.84378	13494420
	8	0.82816	13413375
	9	0.82816	13425442
	10	0.84376	13366848
	Ave	**0.832841**	**13402245.3**

In problem set 04, there are 24 items and the range of the profit and weight value are the largest among the 5 problem sets. The hybrid model performed the best in terms of finding the best fitness solution although it consumed the longest time. If this case scenario is translated into the finance domain, the hybrid model on average is able to search for a solution with around $60,000 more in profit than the ant colony optimization and around $270,000 more in profit compared to the genetic algorithm. This indicates that for a complex problem, both in terms of the number of items and the range of the values, the hybrid model is the best choice among the three algorithms.

E. Problem Set 05

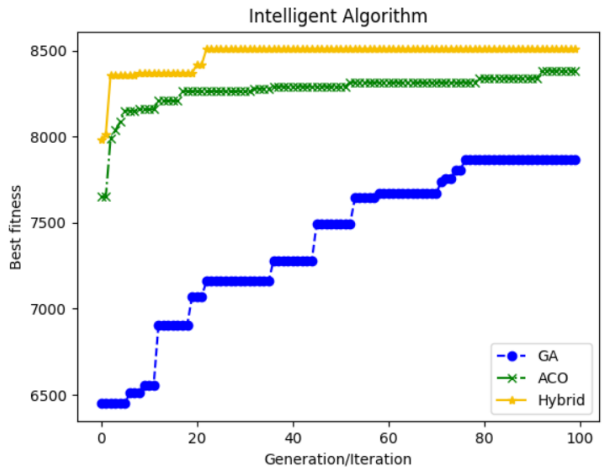

Fig. 26. Graph of best fitness against generation or iteration for one of the example in problem set 05.

TABLE VI. SUMMARY OF PERFORMANCE OF THE 3 ALGORITHMS IN PROBLEM SET 05

Algorithm	No	Time Taken (s)	Best Fitness Value
Genetic Algorithm	1	0.1094	7867
	2	0.12498	7783
	3	0.10938	6698
	4	0.10936	8820
	5	0.10937	7979
	6	0.10938	7379
	7	0.09375	6997
	8	0.10937	8067
	9	0.10937	7243
	10	0.125	7677
	Ave	**0.110936**	**7651**
Ant Colony Optimization	1	1.43754	8384
	2	1.42187	8586
	3	1.43753	7591
	4	1.50002	9782
	5	1.53127	8448
	6	1.3594	8251
	7	1.4219	7877
	8	1.37502	8145
	9	1.29689	7508
	10	1.50002	8330
	Ave	**1.428146**	**8290.2**
Hybrid Model	1	2.03128	8508
	2	2.0313	8523
	3	2.03128	7519
	4	2.01563	9782
	5	2.2188	8452
	6	1.96878	8183
	7	1.98438	7905
	8	1.9844	8327
	9	1.92192	7497
	10	2.1563	8352
	Ave	**2.034407**	**8304.8**

For problem set 05, the hybrid model is only performed slightly better on average than the ant colony optimization for searching the best fitness solution. Genetic algorithm will need more generation to reach for a better solution if there are many possibilities. Both the ant colony optimization and the hybrid model are able to provide a way better quality solution at the beginning as shown in Figure 26.

Genetic algorithm is the best choice when it comes to solving a simple optimization problem as it is using relatively less time. However, it has the tendency to get stuck in the local minimum. Thus, to avoid the likelihood of premature convergence, the experiment needs to be

conducted multiple times and keep track of the best solution. Besides, the mutation rate can be increased as well.

Ant colony optimization ranked in between the genetic algorithm and the hybrid model in terms of time taken and also the ability to provide a better solution. It is able to outperform the hybrid model by providing a slightly better solution in some cases while using lesser time. As ant colony optimization is guided by the attractiveness function and also the pheromone values, it is mainly focusing on improving the quality of the current solution by searching the good quality space. This resulted in a lack of diversity and in some cases causing premature convergence.

To solve the problem of premature convergence, the hybrid model improves the mutation process with the guided search method. The hybrid model performed the best in terms of searching for the best fitness solution, especially for complex cases where there are many items and the range of the profit values and weight are large. However, it consumed way more time than the other 2 algorithms as the population initialization process, the crossover process and the mutation process are all guided by either attractiveness function or pheromone trail values. The hybrid model is the best choice for the complex cases where the range of the cost and profit values are large.

V. CONCLUSION AND FUTURE RESEARCH

The proposed hybrid model combined the guided search functions from the ant colony optimization with the main algorithm structure from the genetic algorithm. The initial population is generated and guided by using the attractiveness function from the ant colony optimization to create better quality solutions at the beginning. The crossover process is guided by the pheromone concentration function. The pheromone will only be updated if the fitness of the offspring is better than both of the parents. This further improves the quality of the existing solutions by exploring the quality solution space. Mutation process is done by removing the item with the largest weight from the solution and selecting other possible items guided by the pheromone concentration. This method introduces diversity into the population while maintaining the quality of the mutation offspring. The proposed hybrid model is showing promising performance in terms of obtaining the best fitness solution, especially for the case of solving complex problems with large numbers of items and large range of items' profit values and weights.

In the future, the evaluation of the algorithms can be done by checking the fitness value of the best solution after a fixed amount of time is passed. This method is more accurate as it evaluates the algorithms by fixing the amount of computational power allocated for searching the best solution. Besides, how each parameter in the hybrid model affects its performance can be further studied. Different combinations of the attractiveness function and the pheromone concentration function can be applied in the hybrid model to test on its effectiveness. Furthermore, the artificial immune system algorithm can also be incorporated into the proposed hybrid model to further enhance its performance.

ACKNOWLEDGMENT

Appreciation to Dr. Imran Medi for the guidance given in completing this study.

REFERENCES

[1] Z. C. Dagdia and M. Mirchev, "When Evolutionary Computing Meets Astro- and Geoinformatics," in *Knowledge Discovery in Big Data from Astronomy and Earth Observation*, Elsevier, 2020, pp. 283–306.

[2] T. Weise, M. Zapf, R. Chiong, and A. J. Nebro, "Why is optimization difficult?," in *Studies in Computational Intelligence*, vol. 193, Springer, Berlin, Heidelberg, 2009, pp. 1–50.

[3] X. S. Yang, *Nature-Inspired Optimization Algorithms*. Elsevier Inc., 2014.

[4] F. Zhao, Z. Yao, J. Luan, and X. Song, "A Novel Fused Optimization Algorithm of Genetic Algorithm and Ant Colony Optimization," *Math. Probl. Eng.*, vol. 2016, pp. 1–10, 2016, doi: 10.1155/2016/2167413.

[5] M. Okwu, O. B. Otanocha, H. O. Omoregbee, and B. A. Edward, "Appraisal of genetic algorithm and its application in 0-1 knapsack problem," *J. Mech. Energy Eng.*, vol. 4, no. 1, pp. 39–46, 2020, doi: 10.30464/jmee.2020.4.1.39.

[6] C. Blum, "Ant colony optimization: Introduction and recent trends," *Physics of Life Reviews*, vol. 2, no. 4. Elsevier, pp. 353–373, Dec. 01, 2005, doi: 10.1016/j.plrev.2005.10.001.

[7] M. Assi and R. A. Haraty, "A Survey of the Knapsack Problem," *ACIT 2018 - 19th Int. Arab Conf. Inf. Technol.*, pp. 1–6, 2019, doi: 10.1109/ACIT.2018.8672677.

[8] S. Xu, X. Chen, X. Pi, C. Joe-Wong, P. Zhang, and H. Y. Noh, "Incentivizing vehicular crowdsensing system for large scale smart city applications," in *Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems 2019*, Mar. 2019, vol. 10970, p. 51, doi: 10.1117/12.2514021.

[9] L. Soukaina, N. Mohamed, E. A. Hassan, and A. Boujemâa, "A hybrid genetic algorithm for solving 0/1 knapsack problem," *ACM Int. Conf. Proceeding Ser.*, no. May, pp. 1–6, 2018, doi: 10.1145/3230905.3230907.

[10] P. T. Donald Kreher, Douglas Simpson, Silvano Martello, "KNAPSACK_01 - Data for the 01 Knapsack Problem." https://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01/knapsack_01.html (accessed Jan. 23, 2021).

[11] "deap/crossover.py at master · DEAP/deap · GitHub," 2019. https://github.com/DEAP/deap/blob/master/deap/tools/crossover.py (accessed Jan. 23, 2021).

[12] K. Schiff, "Ant Colony Optimization Algorithm for the 0-1 Knapsack Problem Algorytm Mrówkowy Dla 0-1 Problemu Plecakowego."