Assignment One: Problem Solving with an Evolutionary Algorithm

Introduction

For my design, I followed the steps from the requirements of CA1 to develop the Evolutionary Algorithm and got the result. Based on the design process and the result of the data, I will elaborate on and analyse it as follow.

Implementation of Evolutionary Algorithm

In this section, I will simply describe how I implement the evolutionary algorithm with some important function step by step in EA. (The completely commented code will attach in zip file.)

1. Generate an initial population of p

```
#Step1: Initial population and evaluate all fitness
   Population_P = init(Population_size)
   Fitness_P = eval_Fitness(Population_P)
   Population_P,Fitness_P,nsnum = selection_Filter(Population_P, Fitness_P)
```

Randomly generate a population p by init(), evaluate all fitness of the p by eval_Fitness(). If the chromosomes in population p are overweight, selection_Filter() function will pop out those solutions and re-generate solutions until the quantity meets the need of the population size.

2. Binary tournament selection to select two parents, a and b

```
#Step2: Do binary tournament to generate a & b
       champion =  [[0]*chromosome_Len for i in range(0,2)]
       k = 0
       while k <= 1: # Do tournament selection twice to select a & b
           random_index_T = random.sample(range(0,len(Population_P)),
tourn_size)
       a = champion[0]
       b = champion[1]
```

Randomly choose 2 solutions from population p to compare the values and the biggest value will be parent a. Repeat this tournament selection again to select parent b.

3. Single-Point crossover to generate 2 children, c and d

```
#Step3: Crossover and generate c and d
       child_cd = crossover(a,b)
       c = child_cd[0]
       d = child_cd[1]
```

Use crossover() to randomly select a crossover point and swap gene between a and b, then generate c and d.

4. Run mutation to give two new solutions, e and f

```
#Step4: Mutation c & d to generate e & f
        child_ef = mutation(child_cd, PM)
        while fitness_e[0] > weight_Limit or fitness_f[0] > weight_Limit:
            child_ef = mutation(child_cd,PM)
            fitness_ef = eval_Fitness(child_ef)
```

Use parameter PM to determine how many genes will mutate on a solution (per chromosome) and randomly select the position to do mutation by mutation(). If the weight of e or f is over 285kg, then re-mutate again until both e and f are less than the weight limit.

5. Weakest replacement

```
  #Step5: Weakest replacement
        Population_P = weakest_replacement(child_ef, fitness_e, fitness_f,
Fitness_P, Population_P)
        P = eval_Fitness(Population_P)
```

Use e and f to replace the weakest solution from population p by weakest_replacement() and develop a new generation population.

6. Termination criterion checkpoint

```
   if fitness_count > fitness_Limit:
        Final_solution = Best_solution
        Final_fitness = Fitness_P[Max_value_index]
        break
```

Set the checkpoint if the termination criterion reaches 10000 fitness evaluations, then stop the evolution process and record the previous max value as its best solution.

Result and Analysis of 4 Questions

1. Q1: Which combination of parameters produces the best result?

In my experiment, when the Population size is 2, the Tournament size is 2, and the Mutation rate is 1, it will have a higher probability to produce the best result of my design. Based on all my records, the total weight in the van is 284.0 kg, and the total value is 4353£ of the best result. The result is shown as follows:

```
================Final Solution==============
Generation count: 792
Best 0-1 solution is: [0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1,
 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1,
1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1
, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1]
Best bag combination: ['bag 2', 'bag 4', 'bag 5', 'bag 6', 'bag 8', 'bag 9', 'bag 15', 'bag 16',
'bag 19', 'bag 20', 'bag 21', 'bag 22', 'bag 24', 'bag 25', 'bag 28', 'bag 31', 'bag 32', 'bag 33
', 'bag 34', 'bag 36', 'bag 37', 'bag 38', 'bag 40', 'bag 42', 'bag 44', 'bag 45', 'bag 46', 'bag
 47', 'bag 48', 'bag 50', 'bag 51', 'bag 53', 'bag 54', 'bag 56', 'bag 57', 'bag 58', 'bag 59', '
bag 60', 'bag 61', 'bag 62', 'bag 63', 'bag 65', 'bag 66', 'bag 67', 'bag 69', 'bag 72', 'bag 73'
, 'bag 74', 'bag 76', 'bag 77', 'bag 78', 'bag 80', 'bag 83', 'bag 84', 'bag 86', 'bag 87', 'bag
89', 'bag 90', 'bag 91', 'bag 94', 'bag 95', 'bag 96', 'bag 99', 'bag 100']
Best Max value: 4353
Weight of total bags: 284.0
```

Image 1: Best result

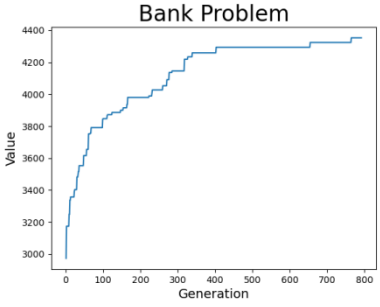

Figure 1: Best solution of each generation

Image_1 shows detailed information on the best result, like how many generations at the end, bag combination, max value, and total weight. Figure_1 shows the evolution process trend of the EA, and Figure_1 also proves that my design is correct to implement with the evolutionary algorithm.

2. Q2: Why do you think this case?

In my observation, when the population size and mutation rate go down, the result of the solution will tend to be better. And the test results are shown as follows:

(1) Fixed mutation rate and tournament size (Population size: 2 ~ 500)

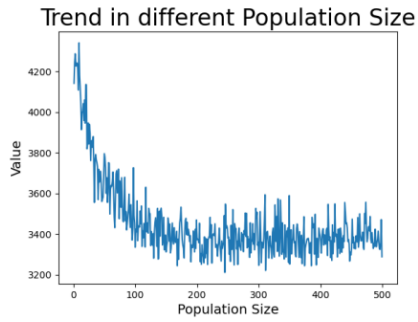

Figure 2: Mutation rate = 1, Tournament size = 2

Figure_2 shows the trend of the value going down when the population size is increasing. In my opinion, the smaller population size can increase the probability of becoming a and b of the next generation from the previous replacement. Thus, the smaller population size can have a faster evolutionary process to get the larger value in 10000 times fitness evaluation.

(2) Fixed population size and tournament size (Mutation rate: 1 ~ 50)

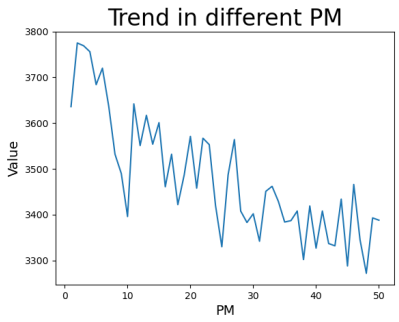

Figure 3: Population size=50, Tournament size=2

Figure_3 shows the trend of the value going down when the mutation rate is increasing. The situation I think is because as evolution continues, all possible solutions in the population become larger, and the big mutation rate could make these values of bigger solutions change to smaller values, and then lower the evolution speed to find the best result.

(3) Fixed population size and mutation rate (Tournament size: 2 ~ 50)

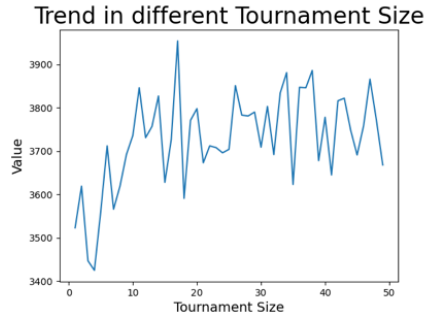

Figure 4: Population size=100, Mutation rate=1

Figure_4 shows the trend of the value going up slowly when the tournament size is increasing. However, if the tournament size increases, we must also let the population size increase. Because Figure_2 shows that increasing population size will make the result go down rapidly, thus, after careful consideration, I select a small tournament size to match a large population to find the best result.

3. Q3: What was the effect when you removed mutation? What about crossover?

(1) Removed mutation (Tournament size=2)

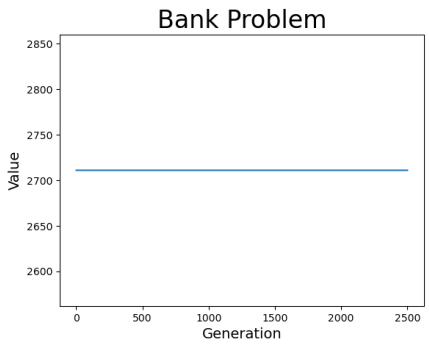

Figure 5: Population size=2

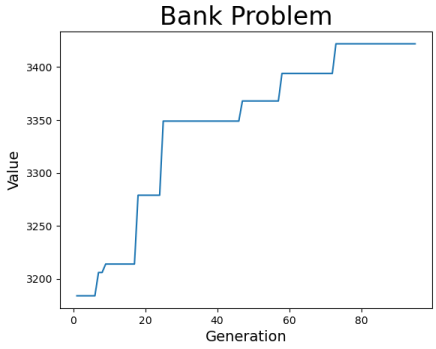

Figure 6: Population size=100

After removing the mutation, when the population is in small size, the population will not evolve like Figure_5; even if the population size increase like Figure_6, the evolutionary process will not grow smoothly and stop evolving at some stage.

(2) Removed crossover (Mutation rate=1, Tournament size=2)

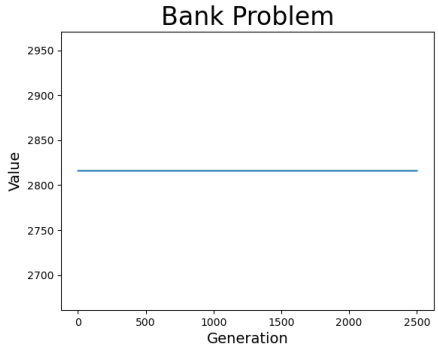

Figure 7: Population size=2

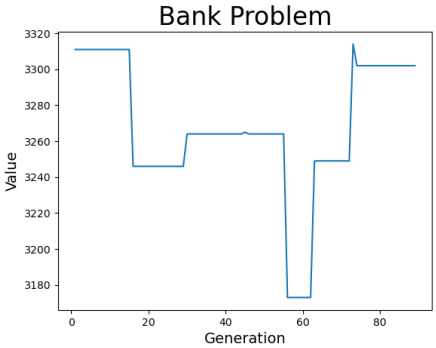

Figure 8: Population size=100

The figures shown above explain that no matter population size is small or large, if we remove the crossover step, it will make the EA can't normally work to find the best result, and the growth trend is not like Q1 curve of best result grows up gradually as well.

4. If you were to extend your EA to work with a multi-objective version of this problem, which functions in your program would you change and why?

According to the Pareto principle, since too many facts will have conflict, and roughly 80% of consequences come from 20% of causes, thus, I will use Pareto Domination Tournament selection to replace the binary tournament function. The non-dominated solutions are filtered using corresponding criteria. The retained non-dominated solution is the best solution group obtained by the algorithm.