Week 8 Report

Some modification to the GA

37237256 薛丁銘

This week’s assignment was to complete the generation algorithm taught in the class. I did some modifications and experiments. Now I would like to introduce the code.

Firstly, the gene pool was generated as in Figure 1. The population size was also settled to 10. The gene pool was generated by ASCII code.

```python
import random

target_sentence = "I love machine learning"

## Gene_pool establishment
gene_pool = " "
for i in range(65,122):
    gene_pool += chr(i)
print(gene_pool)

population_size = 10
```

Figure 1. Gene pool establishment

Then, the functions of generating chromosomes and calculation of fitness were created as shown in Figure 2. I also made a function for calculating the fitness of the whole population.

```python
#Generate Initial Population
def generate_chromosome(length):
    genes = []
    while len(genes) < length:
        genes.append(gene_pool[random.randrange(0,len(gene_pool))])
    return ''.join(genes)

#Random function
def randomfunction(end):
    return random.randrange(0,end)

#Calculate Fitness
def calculate_fitness(chromosome):
    fitness =0
    for i in range(len(chromosome)):
        if chromosome[i] == target_sentence[i]:
            fitness +=1
    return fitness

def calculate_population_fitness(population):
    population_fitness = []
    for chromosome in population:
        population_fitness.append(calculate_fitness(chromosome))
    return population_fitness
```

Figure 2. chromosome generation and fitness calculation

Then I added crossover functions. The idea was to select two healthiest parents for crossover. I also made a function of selecting two unhealthiest chromosomes for cleaning the population. The code is as shown in Figure 3.

```python
#Crossover function
def crossover(chromosome1, chromosome2):
    crossover_loc = random.randrange(0,len(chromosome1)-1)
    chromosome1_first = chromosome1[0:crossover_loc]
    chromosome1_second = chromosome1[crossover_loc:]
    chromosome2_first = chromosome2[0:crossover_loc]
    chromosome2_second = chromosome2[crossover_loc:]
    chromosome1_final = chromosome1_first + chromosome2_second
    chromosome2_final = chromosome2_first + chromosome1_second
    return chromosome1_final, chromosome2_final

def find_two_healthest_parents(population):
    population_modify = population.copy()
    population_modify_fitness = calculate_population_fitness(population_modify)
    healthest1th = max(population_modify_fitness)
    population_modify_fitness.remove(healthest1th)
    healthest2nd = max(population_modify_fitness)
    population_modify_fitness = calculate_population_fitness(population_modify)
    chromosome1 = population_modify[population_modify_fitness.index(healthest1th)]
    chromosome2 = population_modify[population_modify_fitness.index(healthest2nd)]
    chromosome1_index = population.index(chromosome1)
    chromosome2_index = population.index(chromosome2)
    return chromosome1_index, chromosome2_index, chromosome1, chromosome2

def find_two_unhealthest_chromoome(population):
    population_modify = population.copy()
    population_modify_fitness = calculate_population_fitness(population_modify)
    healthest1th = min(population_modify_fitness)
    population_modify_fitness.remove(healthest1th)
    healthest2nd = min(population_modify_fitness)
    population_modify_fitness = calculate_population_fitness(population_modify)
    chromosome1 = population_modify[population_modify_fitness.index(healthest1th)]
    chromosome2 = population_modify[population_modify_fitness.index(healthest2nd)]
    chromosome1_index = population.index(chromosome1)
    chromosome2_index = population.index(chromosome2)
    return chromosome1_index, chromosome2_index, chromosome1, chromosome2

def crossover_population(population):
    chromosome1_index, chromosome2_index, chromosome1, chromosome2 = find_two_healthest_parents(population)
    chromosome1_new, chromosome2_new = crossover(chromosome1, chromosome2)
    population[chromosome1_index] = chromosome1_new
    population[chromosome2_index] = chromosome2_new
    return population
```

Figure 3. Crossover code

Next comes the mutation. The mutation consists of two parts, the mutation of the chromosomes and the possibility of mutation. The code is as shown in Figure 4.

```python
def mutate(chromosome):
    index_to_mutate = randomfunction(len(chromosome))
    gene = list(chromosome)
    mutated_gene = gene_pool[randomfunction(len(gene_pool))]
    gene[index_to_mutate] = mutated_gene
    return ''.join(gene)

def mutate_population(population, mutate_probability):
    chromosome1_index, chromosome2_index, chromosome1, chromosome2 = find_two_healthest_parents(population)
    chromosome3_index, chromosome4_index, chromosome3, chromosome4 = find_two_unhealthest_chromoome(population)
    if mutate_decide(mutate_probability):
        chromosome1_new = mutate(chromosome1)
        population[chromosome3_index] = chromosome1_new
    if mutate_decide(mutate_probability):
        chromosome2_new = mutate(chromosome2)
        population[chromosome4_index] = chromosome2_new
    return population

def evolution_criteria(population):
    for fittness in calculate_population_fitness(population):
        if fittness == len(target_sentence):
            return True
    return False

## TEST
# chromosome1 = 'eeeeee'
# chromosome2 = 'wwwwww'
# population = [chromosome1, chromosome2]
# crossover_population(population)
# print(population)
```

Figure 4. The mutations

Finally, it comes the generation code. Because I have made the functions in detailed, the generation function seems to be very simple as shown in Figure 5.

```python
# Generation
def bug_discover(population):
    size = []
    for chromosome in population:
        size.append(len(chromosome))
    return size


def generation_algorithms(generation_times, mutate_probability):
    population = []
    for i in range(population_size):
        population.append(generate_chromosome(len(target_sentence)))
    population_fitness = calculate_population_fitness(population)

    for generation in range(generation_times):

        # Crossover
        population = crossover_population(population)

        # Mutate
        population = mutate_population(population, mutate_probability)

        if evolution_criteria(population):
            break

        # print(bug_discover(population))
        print(generation)

    print("Current Population: ", population)
    print("Current Fitness", calculate_population_fitness(population))


generation_algorithms(generation_times = 100000, mutate_probability = 0.7)
```

Figure.5 Generation code

I set the generation times to 100000 and usually the mutation stops as 3000-6000 times of generation. Figure 6 shows one of the generation results.

```
Current Population:  ['I love machine learning', 'Mdlove machine learning', '[ love machine learning', 'v
ing', 'k love machine learning', 'H love machine learning', 'e love machine learning']
Current Fitness [23, 21, 22, 22, 22, 22, 22, 22, 22, 22]
PS D:\code> 
```

Figure.6 Generation result

We can find some interesting results that usually when the code reaches the criteria, all populations are almost similar to the target sentence.

With the increase of the length of the target sentence, we can see that more generation times are required as shown in Figure 7.

```python
target_sentence = "I love machine learning and I really really love it"
```

```
Current Population:  ['I love machine learning and I really really love it', 'I love macUine learningHand I re
arning and I really really love it', 'I love macDine learning and I really really love it', 'I love macaine le
love macDine learning and I really really love it', 'I love macDine learning and I really really love it', 'I
Current Fitness [51, 49, 50, 50, 50, 50, 50, 50, 50, 50]
```

Figure.7 result with longer target sentence

I also adjusted the mutation possibility. The conclusion is that with lower mutation possibility, the generation time extends as shown in Figure 8.

```
generation_algorithms(generation_times = 100000, mutate_probability = 0.3)
```

```
8895
8896
8897
Current Population:  ['I love machine learning', 'I lofe machinejlearning', 'I love machineZlearning', 'I love machinejlea
ing', 'I love machineylearning', 'I love machinewlearning', 'I love machineilearning']
Current Fitness [23, 21, 22, 22, 22, 22, 22, 22, 22, 22]
PS D:\code>
```

Figure 8. result when mutation possibility is low