Genetic Algorithm: Function optimization problems

Zhaojie Liu && Hanbai Zhao

Final Draft: April 11, 2018

**Problem:**

I created a genetic algorithm to solve the optimal solution problem of complex functions (within a certain range). The problem poses the situation: Given a complex function, find his optimal solution and return the minimum value of the function. The minimum value of a function may not be an integer, but it will be infinitely close to a certain value, and I will turn it into a table to prove.

**Implementation Design:**

1.Genetic code: Each base is one corresponding to a binary number 0 or 1. Each gene is a string of binary symbols. I'm using binary encoding because I think binary encoding conforms to the principle of minimum character set encoding and is easy to analyze using pattern theorem. What’s more, Binary can be very intuitive to show crossover and mutation (The mutual change of 0 and 1).

2.Gene expression: x1 and x2 are two variables, and the change interval is [0-6]. A chromosome consists of 46 bases. The first 23 bits of the binary number are the binary strings of x, and the last 23 bits are the binary strings of y.

3. Fitness function: Since I am evaluating for the minimum of a function, the fitness function is the original function. Calculate the fitness value of each individual on the population. The minimum value in the population is recorded by the comparison method, which is the optimal solution.

4. sort function: Since I am evaluating for the minimum of a function, the fitness function is the original function. Thus, I should disqualify individuals with large fitness. I am using TreeMap method, in descending order of fitness values, Finds the minimum value in the array using firstEntry ().

5. Crossing Over: Single-point crossover: The mutation probability is set to 0.6, and I set a breakpoint for chromosome crossover.

Example:

Before: x1=<0101|0000>  
 After: x1=<0101|1111>

6. Mutation: The principle of my mutation is to select the mutation site by the mutation probability and change the binary bit.

Example:

Before: a=< 1011001000111111000111110111100000010101001111>  
 After: b=<10110010001111110**1**0111110111100000010101001111>

7. Logging:

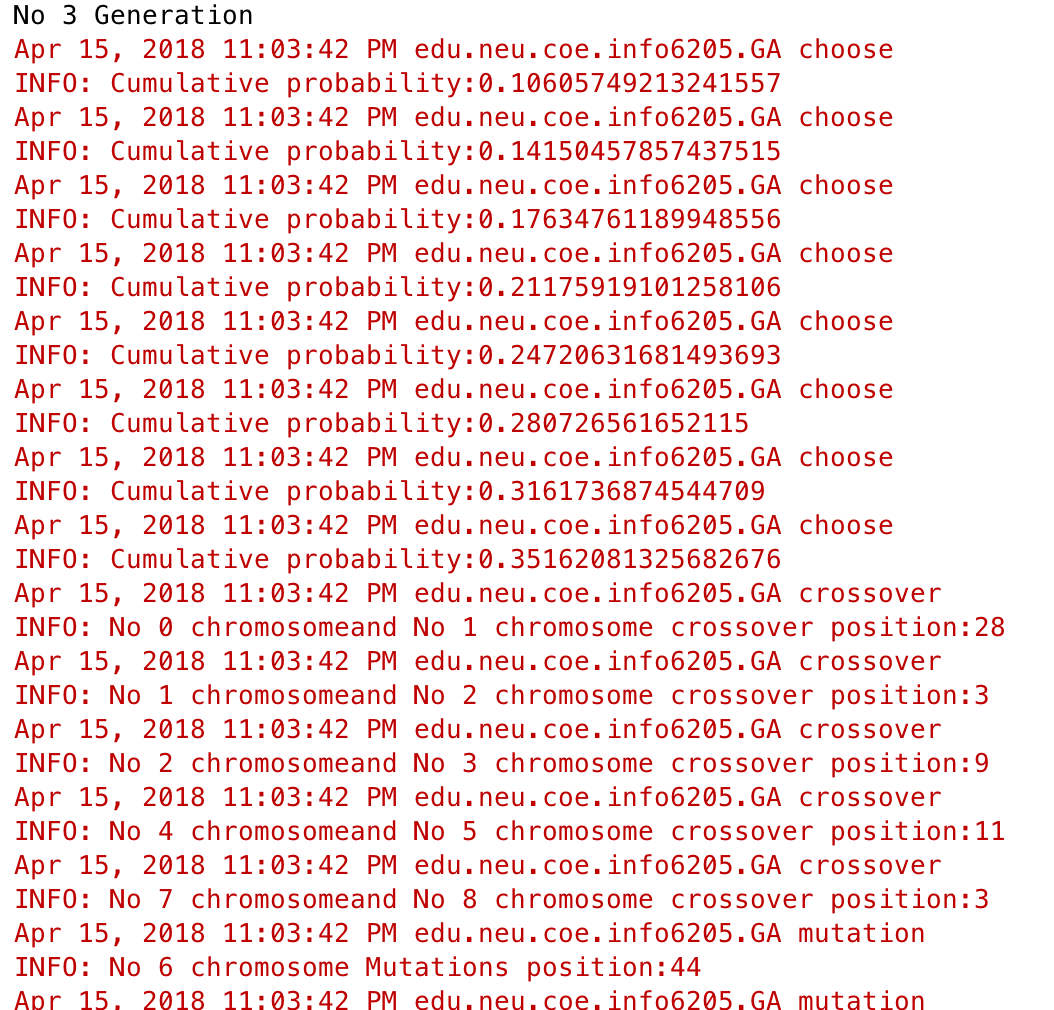
This is logging function to keep track of the progress of the evolution. However, it is the original log4j method of Java.

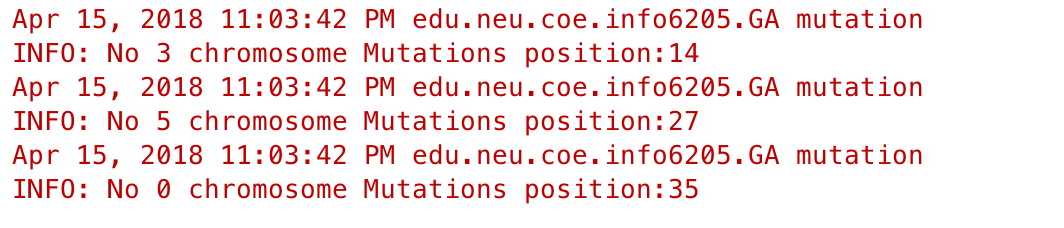
private static Logger log = Logger.getLogger(GA.class.getName());

log.info(“ ”);

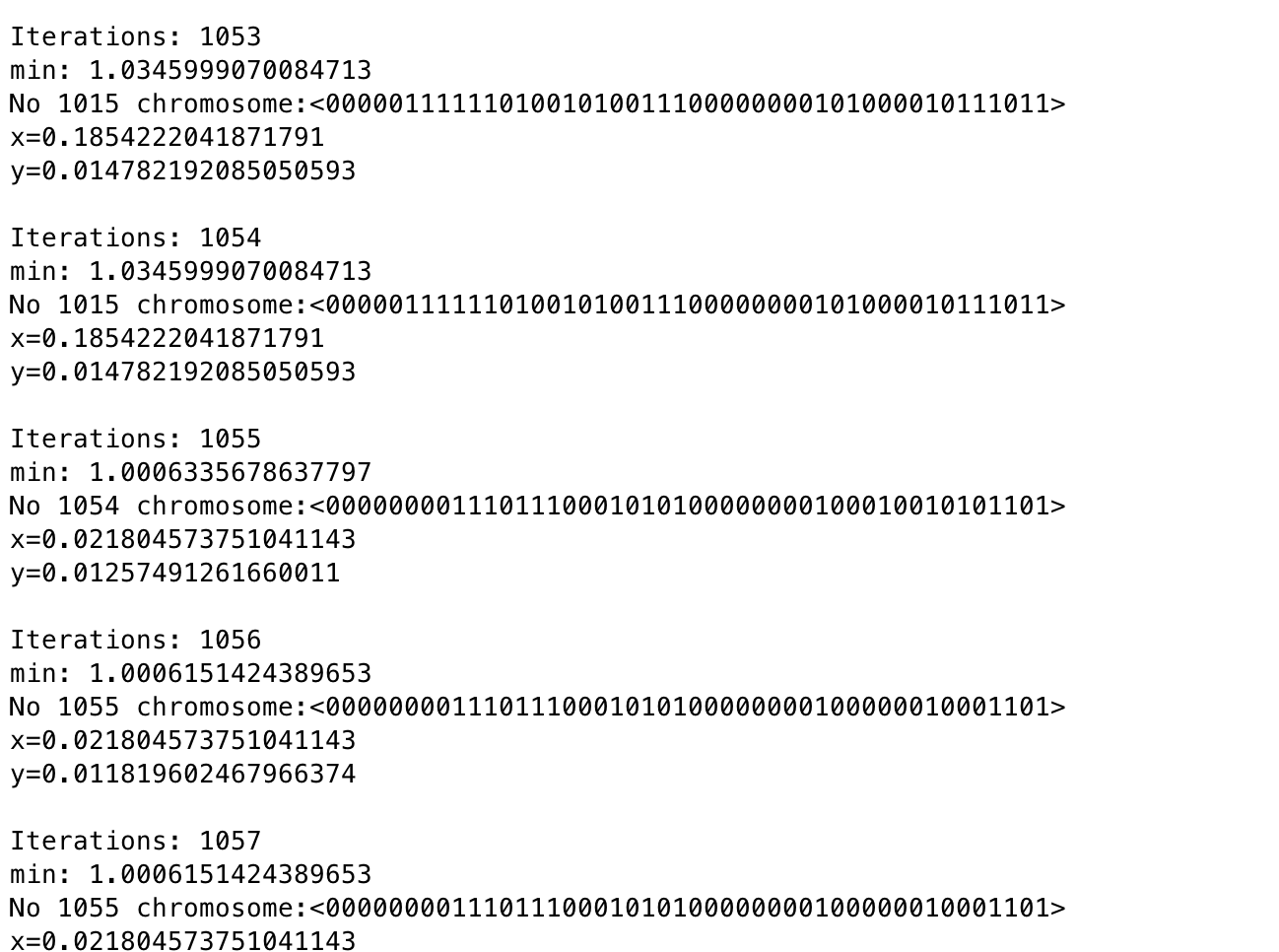
I recorded the evolution, mutation and crossover process. Because I use the roulette wheel selection, I record is the cumulative probability of each individual about evolution.

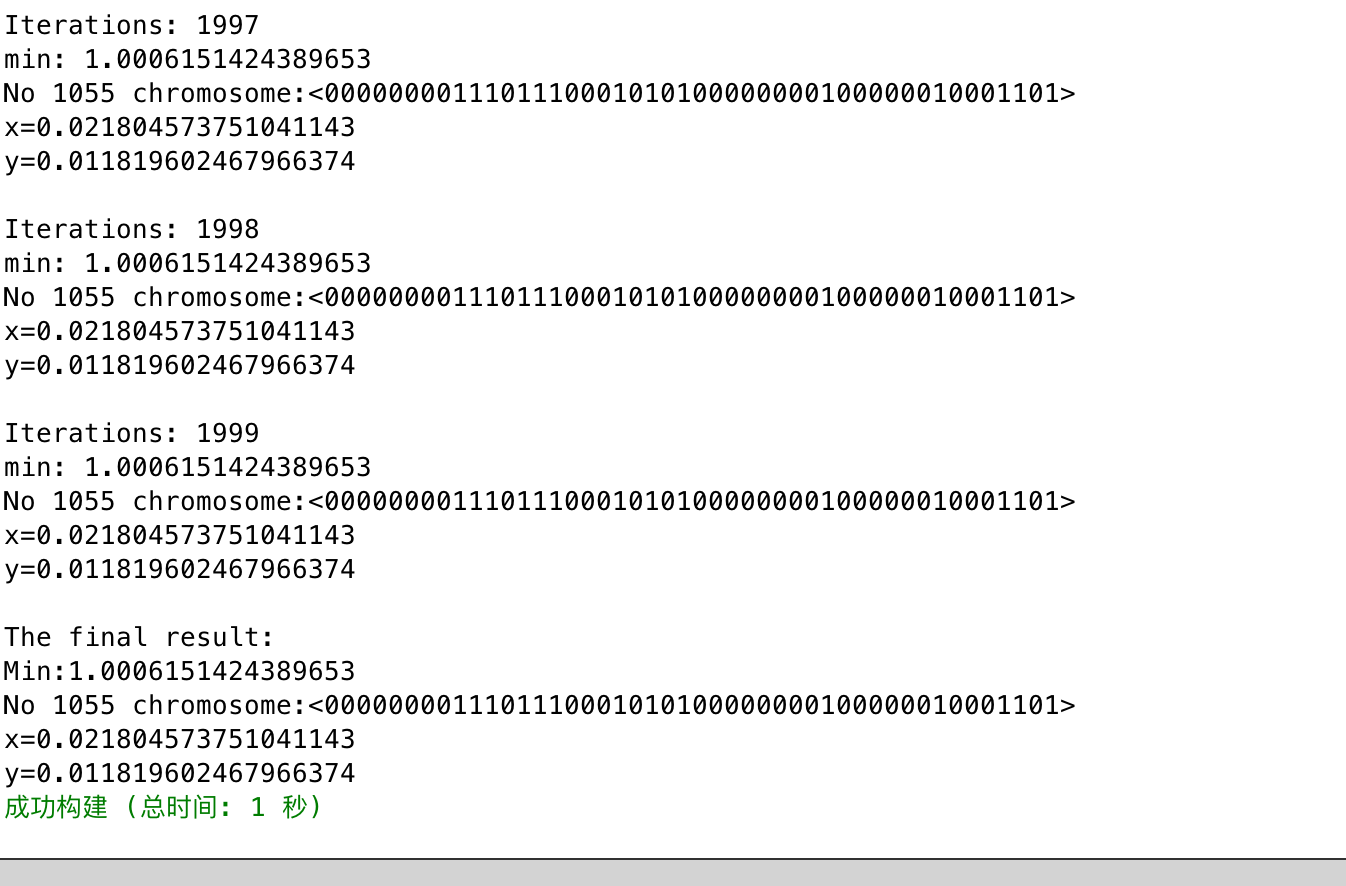
This picture shows the 3th generation of selection, variation and crossover process.





The evolutionary process was terminated after the 1105 generation and the best fitness was found. It has not changed and became the optimal solution.





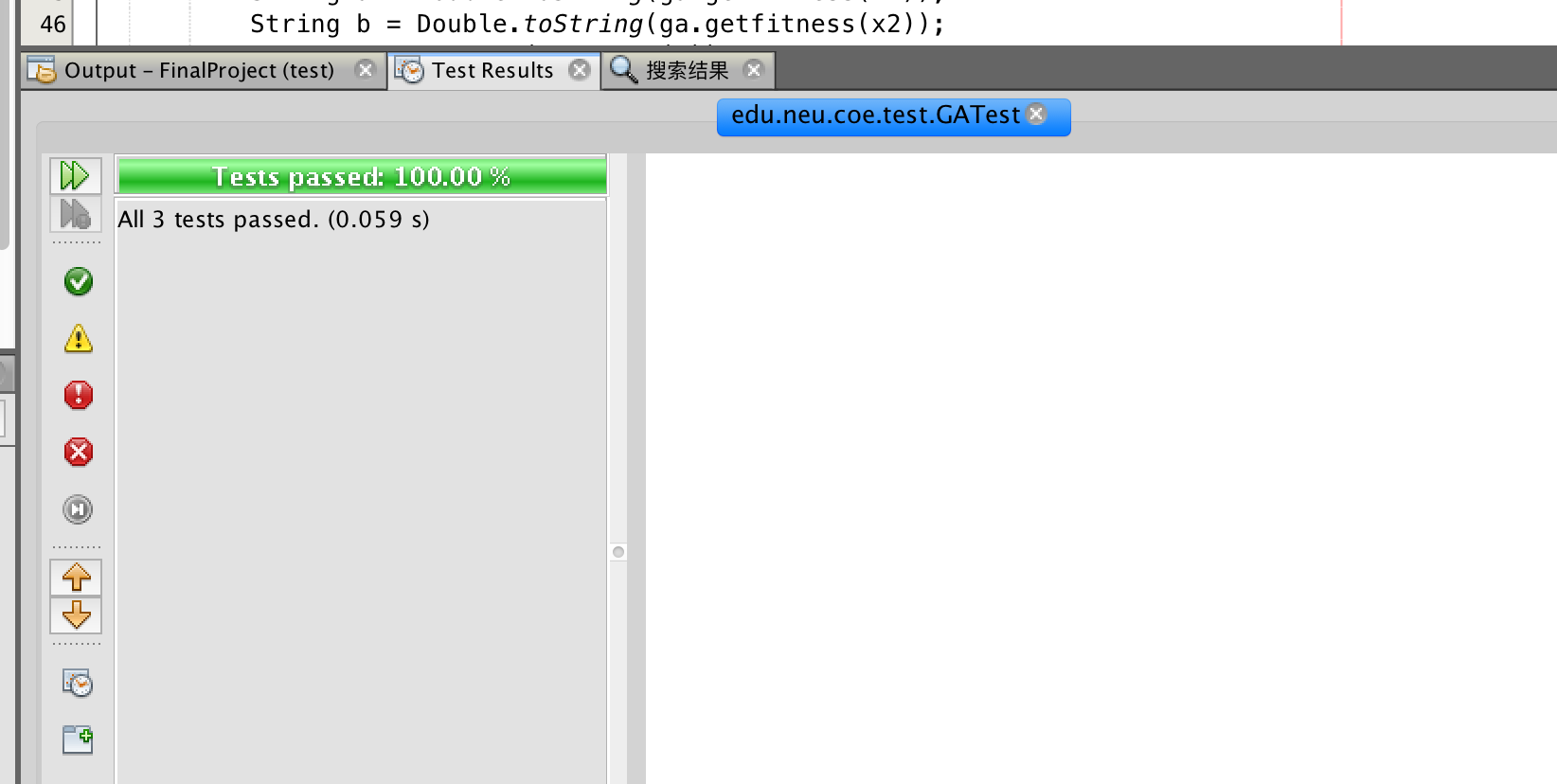
8. Evolution:

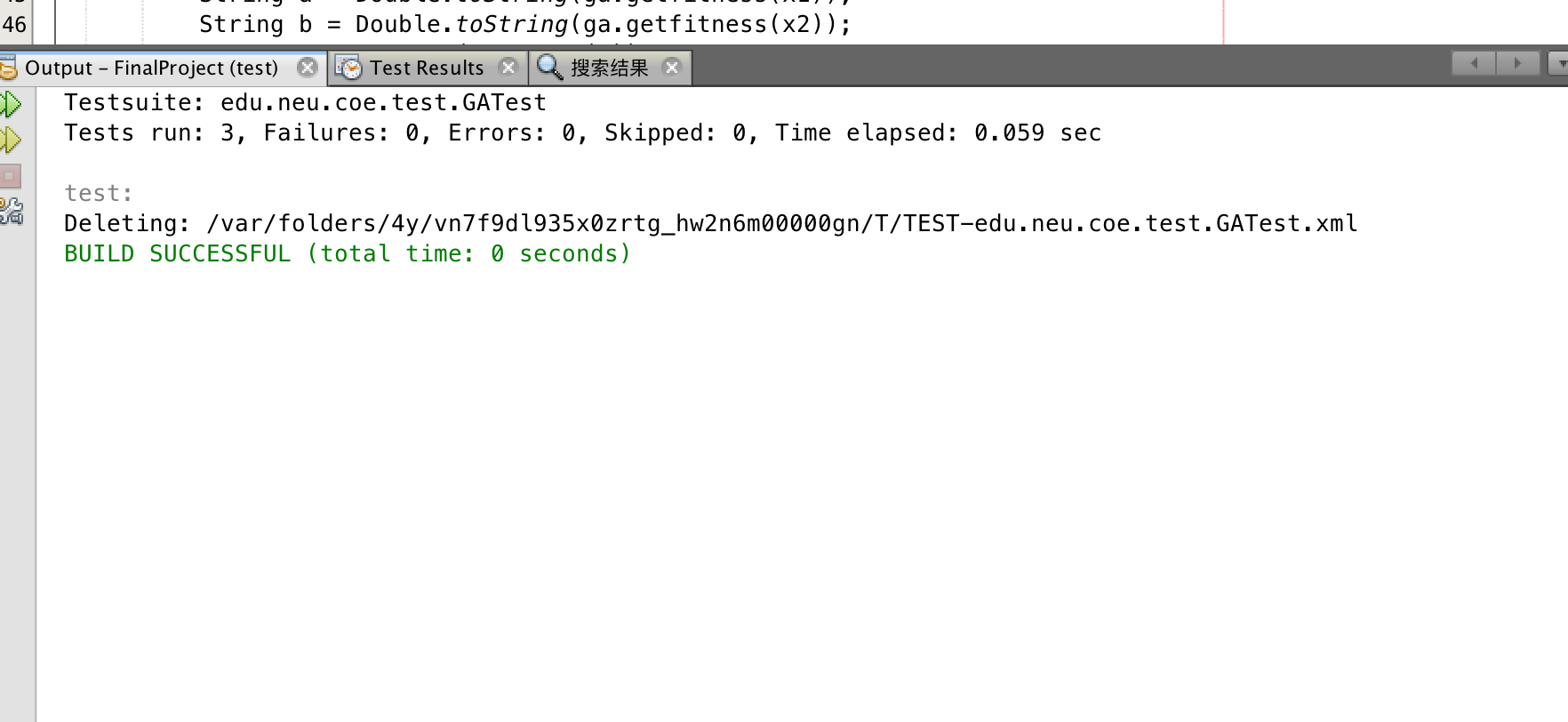
For each generation, all individuals sexually reproduce, doubling the population, and each child experiences the two mutation methods. I then calculate the fitness function for each individual and each person's cumulative probability. I use Roulette Wheel Selection to select people with high fitness who will be selected for the next generation. The remaining individuals then reproduce, doubling the population again.

(1). Generate initial generations, calculate their fitness, and sort. (2). then high fitness of the individuals was kept for mutation and crossover, there is the second generation. (3). The evolutionary process terminates after 2000 generations or when the best fitness score hasn't changed in 900 generations. (4). Output optimal solution.

**Result:**

My project builds and passes my unit test cases:





I found 5 functions to test. Record the relationship between the generation and the best fit. We can see from these five charts, with the increase of generation, fitness will be infinitely close to a certain value.

