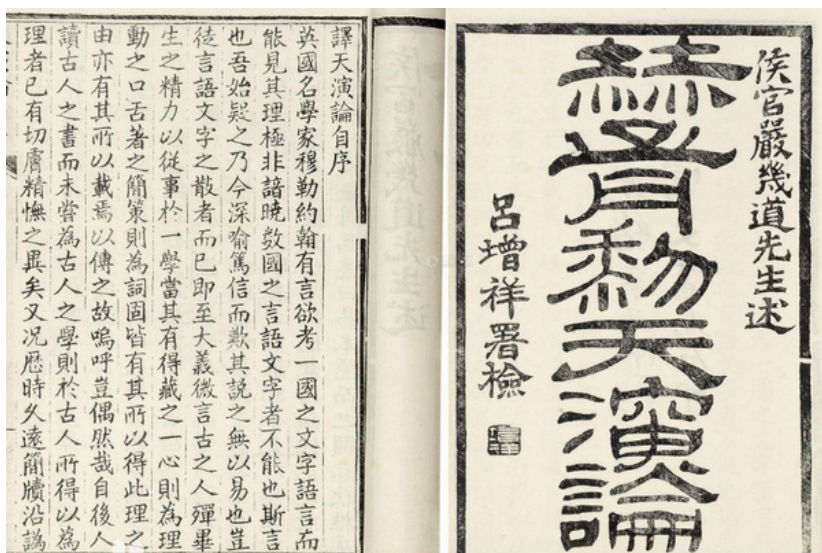


# EVOLUTION AND PROSPER



INFO6205\_111

Team Member:  
Jianchao Li (001054645)  
Jingwen Wang(001052378)

# Content

Info6205_111 .....	1
Content .....	2
1 Introduction.....	3
1.1 Genetic Algorithm .....	3
1.2 Project Goal .....	3
2 Question Description .....	4
2.1 Problem Statements .....	4
2.2 Concepts.....	4
2.2.1 Individuals.....	4
2.2.2 Genotype .....	4
2.2.3 Expression .....	4
2.2.4 Phenotype.....	5
2.2.5 Fitness.....	5
2.2.6 Extinction .....	5
3 Parameter Settings .....	6
3.1 Group size.....	6
3.2 Genotype in Original Population .....	6
3.3 Mutation probability .....	6
3.4 Single Point Crossover probability.....	6
3.5 Maximum Number of Evolution Generations.....	6
4 Program Structure.....	7
5 Algorithm Design.....	8
5.1 Original Population Data generator.....	8
5.2 Fitness Algorithm & Selection Process .....	8
5.3 Diversification Algorithm .....	8
5.4 Mutation Algorithm .....	9
5.5 Single Point Crossover Algorithm .....	10
5.6 Terminate Conditions .....	11
5.7 Algorithm Running Process .....	11
6 Data Analysis .....	12
7 Test Cases.....	15
8 Conclusions .....	16
9 Discussions.....	17
10 References .....	17

# **1 Introduction**

## **1.1 Genetic Algorithm**

In computer science and operations research, a genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection. The goal of the project is to look into a Paramecium(graphic) Genetic problem analogy and find the best solution to the problem.

## **1.2 Project Goal**

In this case, we are aiming at applying Genetic algorithm to find the best genotype. We should select the best non-mutation genotype from the original population at first. Then use the most “longevity” individual in the original population as the starting pattern of the evolution with gene mutation. Do the evolution process in the same nature environment (i.e the same evolution method and termination algorithm) can find the best genotype and do the same process recursively.

## **2 Question Description**

### **2.1 Problem Statements**

At the end of the Qing Dynasty, Chinese scholar Yan Fu translated the English biologist Biography Huxley's "Evolution and Ethics", and promoted the viewpoint of "natural selection, survival of the fittest." The "Compilation" was published, and the publication of the book had a huge social response in Chinese society that Yan Fu had never expected. The project is to validate that in most cases, gene and species diversity can increase the number of generations of population evolution.

### **2.2 Concepts**

#### **2.2.1 Individuals**

An individual means a candidate 2-Dimensional graphic and its location. Each shape is depicted on a 2-dimensional rectangular coordinate system, determined by its chromosome. The number of initial individuals is 50.

#### **2.2.2 Genotype**

The format of the genotype will be presented as decimal numbers. The length of genotype should be dividable by 2.

#### **2.2.3 Expression**

The expression is a two-to-one correspondence between genotypes and phenotypes. To be specific, two bits in genotype determine the coordinate of one point in phenotype, which used to to construct the current generation's shape.

### **2.2.4 Phenotype**

The format of phenotype is a 2-Dimensional graphic that constructed by points, each point consists of two numbers represents x-axis and y-axis coordinate. The other attribute of certain phenotype is its fitness.

### **2.2.5 Fitness**

The fitness argument in our system is how many generations that an individual graphic can evolve. The larger fitness means the corresponding initial graphic fits the environment better, or in other words, more “live long and prosper”.

### **2.2.6 Extinction**

In the evolution process, the number of points of the graph becomes 0.

## **3 Parameter Settings**

### **3.1 Group size**

We generate the group size as 50 in original population and in each following recursion.

### **3.2 Genotype in Original Population**

We generate genotype length in origin population randomly and the scope is [2,20), additionally, the length should be even since every 2 bits determine 1 point.

### **3.3 Mutation probability**

The mutation using Jenetics API will occur during reproduction except the original population's evolution, the starting position of the intersection is entirely random. The probability will be 0.6.

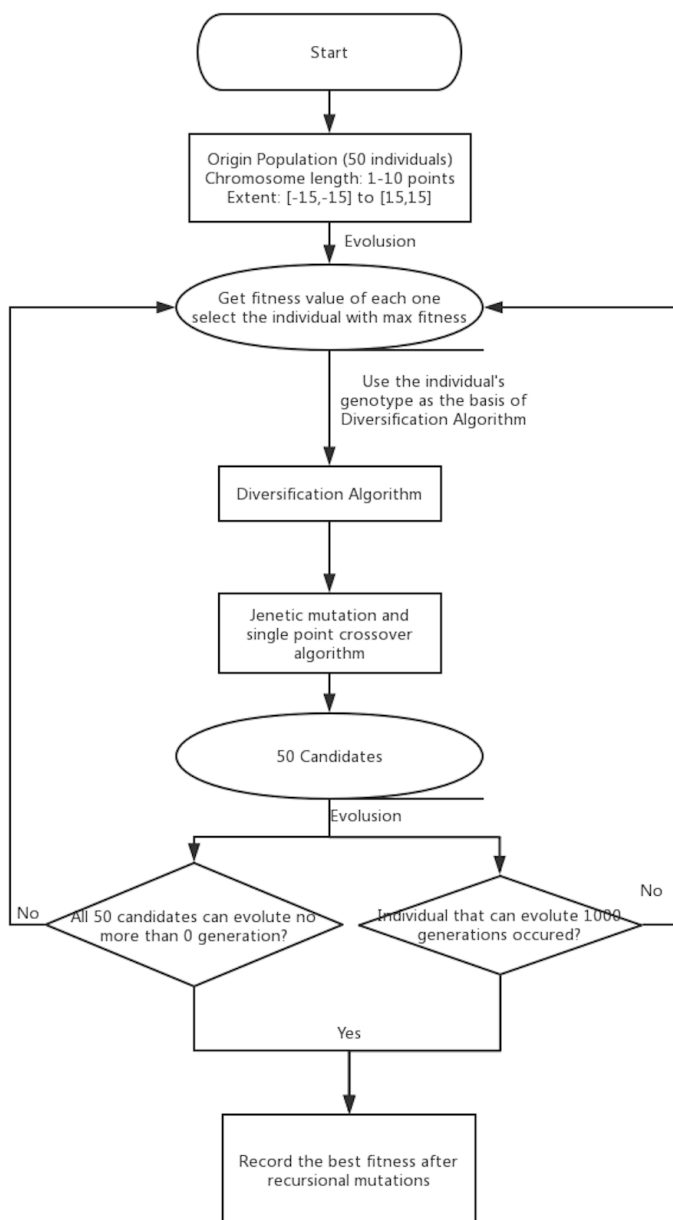
### **3.4 Single Point Crossover probability**

The crossing using Jenetics API will occur during reproduction except the original population's evolution, the starting position is entirely random. The probability will be 0.3.

### **3.5 Maximum Number of Evolution Generations**

In Original population and following recursions, the maximum number of generations would be 1000.

## 4 Program Structure



## **5 Algorithm Design**

### **5.1 Original Population Data generator**

The data generator is used to generate the genotypes of individuals in origin population.

Firstly, we generate 50 graphics randomly. Each graphic is consists of random number of points (from 1 to 10). Each point contains two decimal bit randomly generated from  $[-15,15)$  correspond to its x-axis and y-axis coordinate.

### **5.2 Fitness Algorithm & Selection Process**

In this project, the fitness would be number of evolution generations, which used to access whether an individual can fit the environment well in given environment and mutation methods. In each recursion, the program will select the individual with best fitness. The subsequent recursion is base on the mutations of the progeny of the fittest individual in previous recursion.

### **5.3 Diversification Algorithm**

The subsequence recursion is breed from fittest individuals in the last recursion. We define the fittest individual in one recursion as “winner” of this recursion. The genotype of the winner is used as the “library” of the Diversification Algorithm. In this Algorithm, the “winner” have probability to occur several kinds of alters, the alter methods and corresponding probability are as follows:

Step 1. For each point of winner, 3 possible situations will happen:



Situation	Remains the Same	Shift Towards North Eastern	Eliminate
Probability	95%	4%	1%
Example 1	(1,1)->(1,1)	(1,1)->(2,2)	(1,1)->Null
Example2	(10,10)->(10,10)	(10,10)->(11,11)	(10,10)->Null

Step 2. For each point of winner, there is a probability of 50% to become the accordion of the generating process of the additional point on this chromosome. Assume the winner has  $n$  points and the coordinate of certain point of winner is  $(x, y)$ . The additional point's coordinate would be randomly generated in the range of  $[-2x, 2x]$  if  $x > y$ , and in  $[-2y, 2y]$  otherwise.

Additionally, the total length of the chromosome after alternation will no longer than  $1.5n$ . That is to say, the number of additional points will in the range of 0 to  $0.5n$ . If  $n < 10$ , the number of additional point will still probably be 0 or 1.

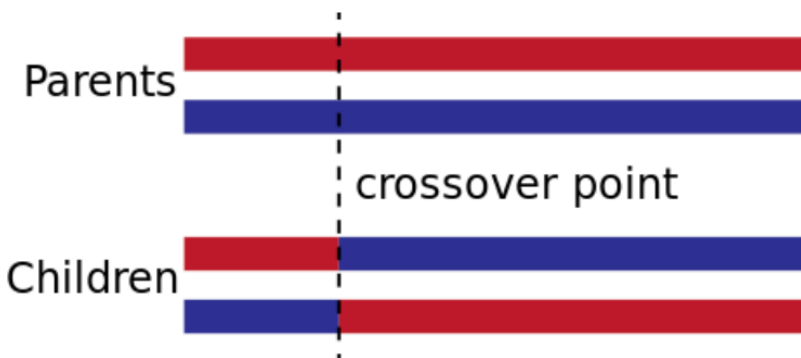
**Example:** Assume that current operating point is (10,20)

->additional point will be generated in the extent of (-40,-40) to (40,40)

## 5.4 Mutation Algorithm

Mutation is a genetic level variation, not for individuals level.

Throughout the whole group, each's genetic sequence is likely to mutate over time. However, this mutation is not the entire gene se-



quence replaced by a new gene sequence, but some sites on the gene sequence been randomly replaced.

The mutation probability is the parameter that must be optimized.

The optimal value of the mutation rate depends on the role mutation plays. If mutation is the only source of exploration (if there is no crossover) then the mutation rate should be set so that a reasonable neighborhood of solutions is explored. [1]

We used the class *Mutator*<>() in *Jenetics API*. In our model, the mutation probability is 0.6.

## 5.5 Single Point Crossover Algorithm

If we create a child and its complement we preserving the total number of genes in the population, preventing any genetic drift.

Single-point crossover is the classic form of crossover.[2]

A point on both parents' chromosomes is picked randomly, and designated a 'crossover point'. Bits to the right of that point are swapped between the two parent chromosomes. This results in two offspring, each carrying some genetic information from both parents.

## 5.6 Terminate Conditions

There are 3 conditions that make an individual stop evolution:

1. It has evaluated for 1000 generations.
2. An evolution pattern that same with its ancestors appears.
3. The number of points in certain generation becomes 0 (Extinction).

There are 2 conditions that make the recursion process stop:

1. The offsprings of the winner in current recursion can evolve no more than 0 generation.
2. Individual that can evolve 1000 generations appears.

## 5.7 Algorithm Running Process

The algorithm is divided into two parts. In the first part, the origin population is generated randomly, which means each individual in the population (represented by graphic consist of points). Calculate each individual's fitness (i.e generation of evolution) in the graph group. The most longevity individual are selected as the gene library of the second part of the experiment.

In the second part, the selected individual's genotype will be modified by a well designed Diversification Algorithm, then produce new population using this modified genotype using mutation and single point crossover algorithm through Jenetics API. Set recursion termination conditions for judgment to out put the pattern with best fitness.



Statistics of this case are as follows:

Time statistics
Selection: sum=0.056648366000 s; mean=0.000026165527 s
Altering: sum=0.212992558000 s; mean=0.000098379934 s
Fitness calculation: sum=4.997829342000 s; mean=0.002308466209 s
Overall execution: sum=5.340607253000 s; mean=0.002466793188 s
Evolution statistics
Generations: 2,165
Altered: sum=277,772; mean=128.301154734
Killed: sum=0; mean=0.000000000
Invalids: sum=50,490; mean=23.321016166
Population statistics
Age: max=14; mean=0.738051; var=1.262350
Fitness:
min = 0.000000000000
max = 1.000000000000
mean = 0.990762124711
var = 0.009152621500
std = 0.095669334165

Result:  
the bestWithoutMutation is : -13 7 ,7 -11 ,-5 9 ,-10 12  
the bestWithoutMutation fitness is : 1  
Evolution 109 generation  
Found Best Pattern : -10 -13 ,9 12  
Found Best Fitness : 1

This case shows that the mutation failed to enhance the genetic diversity of the population, which cause low performance of the patterns.

There may be one “tragic” case that all offsprings performance do not exceed their ancestor. So the best pattern would be the first pattern that generated randomly.

```
<terminated> Game (4) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/java (Dec 7, 2019, 3:37:38 PM)
[-7 9 , -7 -7 ] -> 1
[9 -7 , -7 -7 ] -> 1
[-7 -7 , 9 -7 ] -> 1
[-7 -7 , 9 -7 ] -> 1
[-7 9 , -7 -7 ] -> 1
[9 -7 , -7 -7 ] -> 1
[9 -7 , -7 -7 ] -> 1
[-7 9 , -7 -7 ] -> 1
[-7 -7 , -7 9 ] -> 1
[-7 9 , -7 -7 ] -> 1
[-7 -7 , 9 -7 ] -> 1
[9 -7 , -7 -7 ] -> 1
[9 -7 , -7 -7 ] -> 1
[9 -7 , -7 -7 ] -> 1
[-7 -7 , -7 9 ] -> 1
[-7 -7 , 7 9 ] -> 1
[-7 -7 , -7 9 ] -> 1
[-7 -7 , 9 -7 ] -> 1
[-7 -7 , 9 -7 ] -> 1
[-7 -7 , 9 -7 ] -> 1
[-7 9 , -7 -7 ] -> 1
[9 -7 , -7 -7 ] -> 1
[-7 9 , -7 -7 ] -> 1
[-7 -7 ] -> 0

+-----+
| Time statistics |
+-----+
| Selection: sum=0.027108901000 s; mean=0.000036387787 s |
| Altering: sum=0.104683630000 s; mean=0.000140514940 s |
| Fitness calculation: sum=2.653652032000 s; mean=0.003561949036 s |
| Overall execution: sum=2.815221054000 s; mean=0.003778820207 s |
+-----+
| Evolution statistics |
+-----+
| Generations: 745 |
| Altered: sum=73,120; mean=98.147651007 |
| Killed: sum=0; mean=0.000000000 |
| Invalids: sum=16,466; mean=22.102013423 |
+-----+
| Population statistics |
+-----+
| Age: max=11; mean=0.744215; var=1.302770 |
| Fitness: |
| min = 0.000000000000 |
| max = 1.000000000000 |
| mean = 0.973154362416 |
| var = 0.026125650686 |
| std = 0.161634311600 |
+-----+
Result:
the bestWithoutMutation is : 9 -6 ,10 -7 ,9 -7 ,9 -7
the bestWithoutMutation fitness is : 2
Didn't evaluate pattern better than first pattern
Found Best Pattern : 9 -6 ,10 -7 ,9 -7 ,9 -7
Found Best Pattern : 2
```

This case shows that the genotypes and phenotypes failed to fit the current environment with current mutation methods.

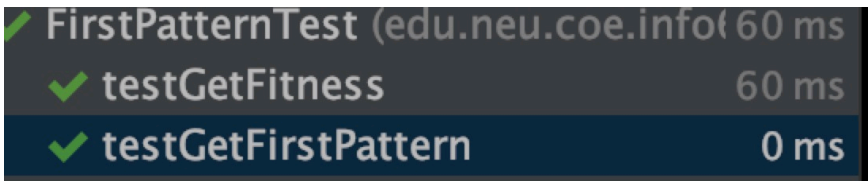
From all these cases, we found that when individual’s genotype becomes more diversified, the individual will perform higher fitness, in other words, more longevous.

## 7 Test Cases

Our program includes following test case:

The *testGetFitness* and *testFitness* tested fitness function;

The *testGetfirstPattern* tested mutation

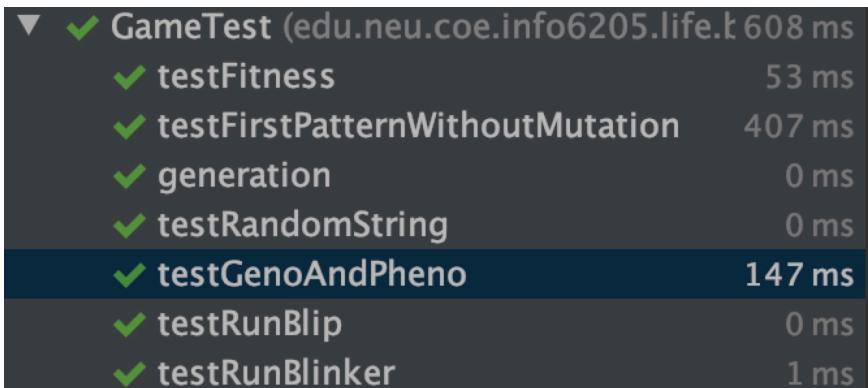


A screenshot of a test runner interface showing results for 'FirstPatternTest'. The interface has a dark background with green checkmarks and white text. The test name and its total duration are at the top. Below it, two sub-tests are listed with their individual durations.

✓ FirstPatternTest (edu.neu.coe.info6205.life.k	60 ms
✓ testGetFitness	60 ms
✓ testGetFirstPattern	0 ms

*testFirstPttternWithoutMutation* and *testRandomString* tested candidate selection function and expression function: the latter generate a string randomly as genotype and expressed as phenotype in the former test case. After selection function, it will return the best phenotype.

*testGenoAndPheno* validates that the expression function (Genotype!= Phenotype)



A screenshot of a test runner interface showing results for 'GameTest'. The interface has a dark background with green checkmarks and white text. The test name and its total duration are at the top. Below it, seven sub-tests are listed with their individual durations. The 'testGenoAndPheno' row is highlighted with a darker blue background.

▼ ✓ GameTest (edu.neu.coe.info6205.life.k	608 ms
✓ testFitness	53 ms
✓ testFirstPatternWithoutMutation	407 ms
✓ generation	0 ms
✓ testRandomString	0 ms
✓ testGenoAndPheno	147 ms
✓ testRunBlip	0 ms
✓ testRunBlinker	1 ms

## 8 Conclusions

The above modeling, algorithm design and output statistics shows a simplified nature evolution model. In most cases, the evolution process can be sum up in three phases: primitive period, development period and peak period.

In the primitive period, individuals usually with low performance, that is, their fitness is relatively low (under 10).

The society will radically turn into next phase — development period (fitness > 10). The genotype of the population will be diversified by gene mutations and single point crossover (there will be more conditions such as multi point crossover and uniform crossover in real world). There are also some extreme cases, such as genetic alters that failed to make the population become more competitive. But in most cases, those operations will highly increase the gene diversity and that will reflect to phenotype diversity. After recursively mutations, crossovers and selections. The probability of occurring individuals that can fit the environment better will increase.

After highly development of the population, there will be more likely to occur individuals with high fitness in current environment. Which is called peak period.



## 9 Discussions

The Genetic Algorithm model is inspired by nature rules. However, the program still have some aspects that can be improved:

1. In the real natural situation, when the number of individuals in the population approaching saturation, internal competition will intensify, which will affect the fitness of the individuals in the population to a certain extent.
2. Considering the system running time, the program will stop when certain individual's fitness reach 1000, but in real world, further evolution may be happened and the output can't be predicted in our algorithm.
3. The nature environment will change in real world, which cause the selection method change with time. Our algorithm assume that the selection rules remains the same in the simulation model.

## 10 References

[1] Jenetics. (Franz Wilhelmstötter). Retrieved November 25, 2019, from <http://jenetics.io/javadoc/jenetics/5.1/index.html>.

[2] Crossover (genetic algorithm). (2019, December 2). Retrieved December 5, 2019, from [https://en.wikipedia.org/wiki/Crossover\\_\(genetic\\_algorithm\)](https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)).