Genetic Algorithm: The Traveling Salesman Problem

Haobo Zhao 001279524 Yitong Liu 001822175

Final Draft: April 14, 2018

Problem

Our team use the genetic algorithm to generate our own solution, solving the Traveling Salesman Problem. The situation for this problem is, there are 48 cities, we need to find a shortest path to visit each of them, and then return this shortest route to the first city. We set a graph model with these 48 nodes, they are connected by undirected edges with different weight. And in the problem solving, we will visit each node and edge once. The distance of this shortest path decides whether we have done a good job or not. Until now, the best result we have generated is less than 16,000.

Implementation Design

Genetic code: Every city has an index to stand itself, so [0] … [47] is the expression of the genetic code for these 48 cities. Each sequence stands for the route of all the 48 cities. And it is very important that each city only can be visited once.

Phenotype: includes the path expression and the distance of the path.

Gene expression: we use binary code to represent these 48 cities, and translate them to string type to represent every gene code, which is called phenotypic code. For example, the first three cities are expressed by binary code like, "000000, 0000001, 0000010", after the translation, they become string type code like, ""000000", "0000001", "0000010",".

Fitness of an individual: The fitness for each individual equals to the reciprocal of the travel distance of the path. But in our team, we don't calculate values as this way, because we think it will cost too much time to do a log calculation (-log N). So we calculate the exact value of each individual's fitness according to the distance, also we did a comparison, then selected the less value.

Fitness function: We set "populationSize" in value of 100 and we name this number "100" as "seed", which means based on these 100 original possibilities, more and more possibilities will be produced, and we will select better one from the produced results. So this Fitness function in class "Solution" is used to compare 100 results, first selecting a better one, then returning the value of "distance".

Mutation: we perform swap mutations by randomly selecting a value, and then send the result to the "Fitness" function. Also, we should notice that the probability of the mutation should not be too large. For example:

Index:	0	1	2	3	4	5	6
Parent:	a	b	c	d	e	f	g

swap indexes 0 and 1

Child:

b	a	c	d	e	f	g

Crossing Over: On this function, we randomly select a pair of value, parent 1 and parent 2. We select value randomly from an even index of parent 1, then put the value into the even index of the child; also we select value randomly from the first index of parent 2, then put the value into the odd index of the child. For example:

Index:

0	1	2	3	4	5	6

Parent 1:

1	3	2	4	5	7	6

we select parent1[2] = 2, put value =2 in the Child[2], then child[2] = 2;
Parent 2:

4	7	1	3	6	5	2

Then we select parent2[1] = 7, put value =7 in the Child[1], then child[1] =7;
Child1:

1	2	3	4	5	6	7

After swap
Child 1:

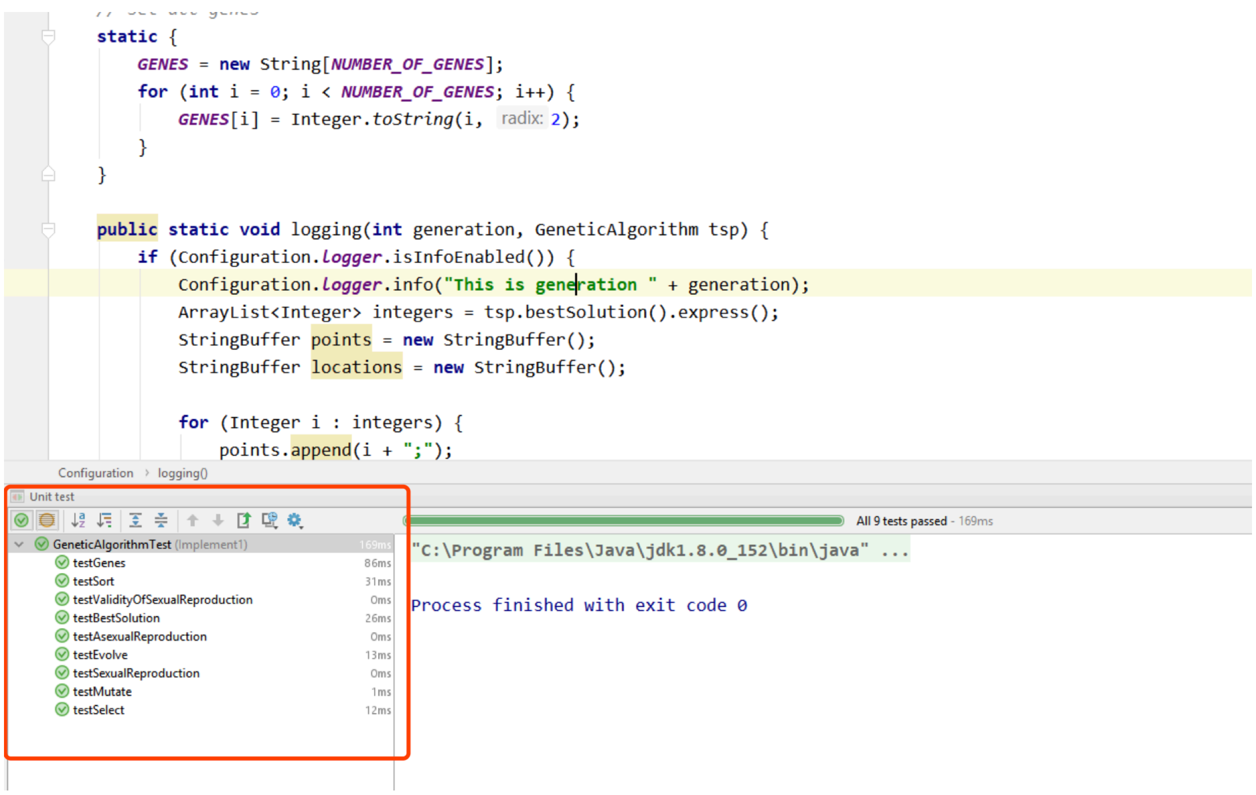
7	4	2	3	5	7	6

As we can see, there are some same values in the "Child1" after the swap. So then we will do while loop to eliminate these duplications.

Evolution: There are three ways in out nature world for an individual, asexual organisms mutate to get genetic diversity, and for sexual organisms, crossing over is the major source for them to get more genetic diversity.

Configuration: As we have three classes, "geneticAlgorithm", "solution", "logging", we put some parameter values in the class "Configuration", so that it is very concise for us to change the values and have more experiments to get some good comparisons.

Results
Here is the result of our unit tests:



```
    // set all genes
    static {
        GENES = new String[NUMBER_OF_GENES];
        for (int i = 0; i < NUMBER_OF_GENES; i++) {
            GENES[i] = Integer.toString(i,  radix: 2);
        }
    }

    public static void logging(int generation, GeneticAlgorithm tsp) {
        if (Configuration.logger.isInfoEnabled()) {
            Configuration.logger.info("This is generation " + generation);
            ArrayList<Integer> integers = tsp.bestSolution().express();
            StringBuffer points = new StringBuffer();
            StringBuffer locations = new StringBuffer();

            for (Integer i : integers) {
                points.append(i + ";");
```

Configuration › logging()

Unit test

```
GeneticAlgorithmTest (Implement1)     169ms
    testGenes                          86ms
    testSort                           31ms
    testValidityOfSexualReproduction    0ms
    testBestSolution                   26ms
    testAsexualReproduction             0ms
    testEvolve                         13ms
    testSexualReproduction              0ms
    testMutate                          1ms
    testSelect                         12ms
```

All 9 tests passed - 169ms

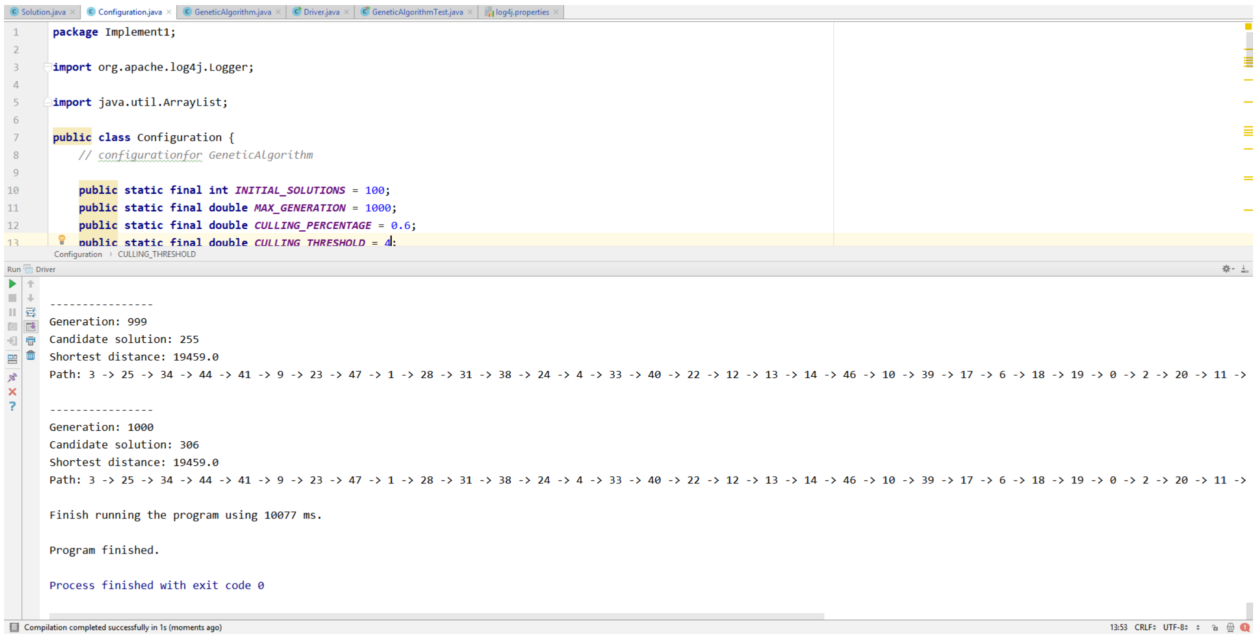
```
"C:\Program Files\Java\jdk1.8.0_152\bin\java" ...

Process finished with exit code 0
```

Here are the results of nine tests, and they all ran successfully.

Changing the values of parameter

Here are four results, we changed the values of parameter “INITIAL_SOLUTIONS” and “MAX_GENERATION” to observe the influence to the solution.

Result 1

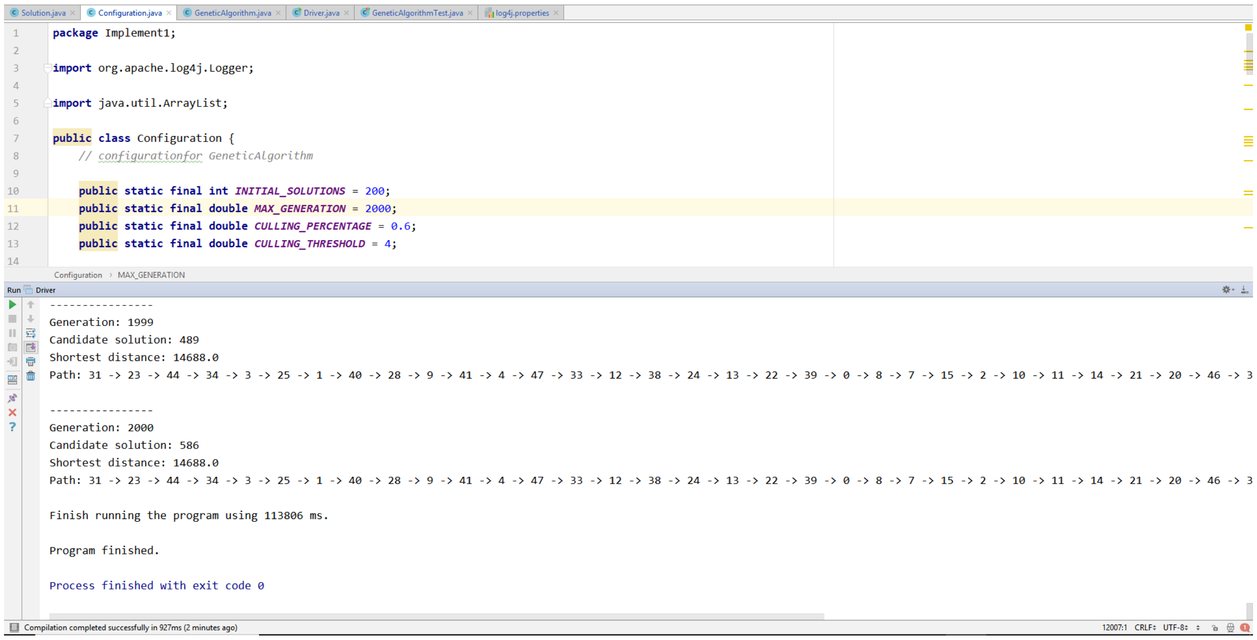


```
package Implement1;

import org.apache.log4j.Logger;

import java.util.ArrayList;

public class Configuration {
    // configurationfor GeneticAlgorithm

    public static final int INITIAL_SOLUTIONS = 100;
    public static final double MAX_GENERATION = 1000;
    public static final double CULLING_PERCENTAGE = 0.6;
    public static final double CULLING_THRESHOLD = 4;
```

```
----------------
Generation: 999
Candidate solution: 255
Shortest distance: 19459.0
Path: 3 -> 25 -> 34 -> 44 -> 41 -> 9 -> 23 -> 47 -> 1 -> 28 -> 31 -> 38 -> 24 -> 4 -> 33 -> 40 -> 22 -> 12 -> 13 -> 14 -> 46 -> 10 -> 39 -> 17 -> 6 -> 18 -> 19 -> 0 -> 2 -> 20 -> 11 ->

----------------
Generation: 1000
Candidate solution: 306
Shortest distance: 19459.0
Path: 3 -> 25 -> 34 -> 44 -> 41 -> 9 -> 23 -> 47 -> 1 -> 28 -> 31 -> 38 -> 24 -> 4 -> 33 -> 40 -> 22 -> 12 -> 13 -> 14 -> 46 -> 10 -> 39 -> 17 -> 6 -> 18 -> 19 -> 0 -> 2 -> 20 -> 11 ->

Finish running the program using 10077 ms.

Program finished.

Process finished with exit code 0
```

Compilation completed successfully in 1s (moments ago)

Result 2

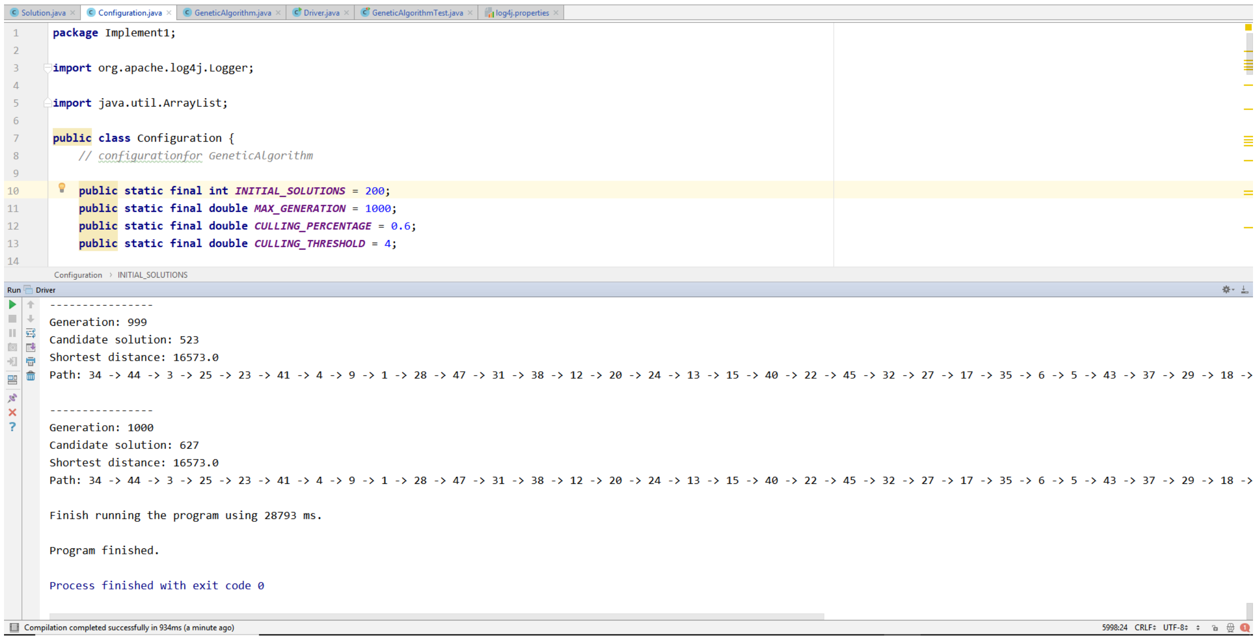


```
package Implement1;

import org.apache.log4j.Logger;

import java.util.ArrayList;

public class Configuration {
    // configurationfor GeneticAlgorithm

    public static final int INITIAL_SOLUTIONS = 200;
    public static final double MAX_GENERATION = 2000;
    public static final double CULLING_PERCENTAGE = 0.6;
    public static final double CULLING_THRESHOLD = 4;
```

```
----------------
Generation: 1999
Candidate solution: 489
Shortest distance: 14688.0
Path: 31 -> 23 -> 44 -> 34 -> 3 -> 25 -> 1 -> 40 -> 28 -> 9 -> 41 -> 4 -> 47 -> 33 -> 12 -> 38 -> 24 -> 13 -> 22 -> 39 -> 0 -> 8 -> 7 -> 15 -> 2 -> 10 -> 11 -> 14 -> 21 -> 20 -> 46 -> 3

----------------
Generation: 2000
Candidate solution: 586
Shortest distance: 14688.0
Path: 31 -> 23 -> 44 -> 34 -> 3 -> 25 -> 1 -> 40 -> 28 -> 9 -> 41 -> 4 -> 47 -> 33 -> 12 -> 38 -> 24 -> 13 -> 22 -> 39 -> 0 -> 8 -> 7 -> 15 -> 2 -> 10 -> 11 -> 14 -> 21 -> 20 -> 46 -> 3

Finish running the program using 113806 ms.

Program finished.

Process finished with exit code 0
```

Compilation completed successfully in 927ms (2 minutes ago)

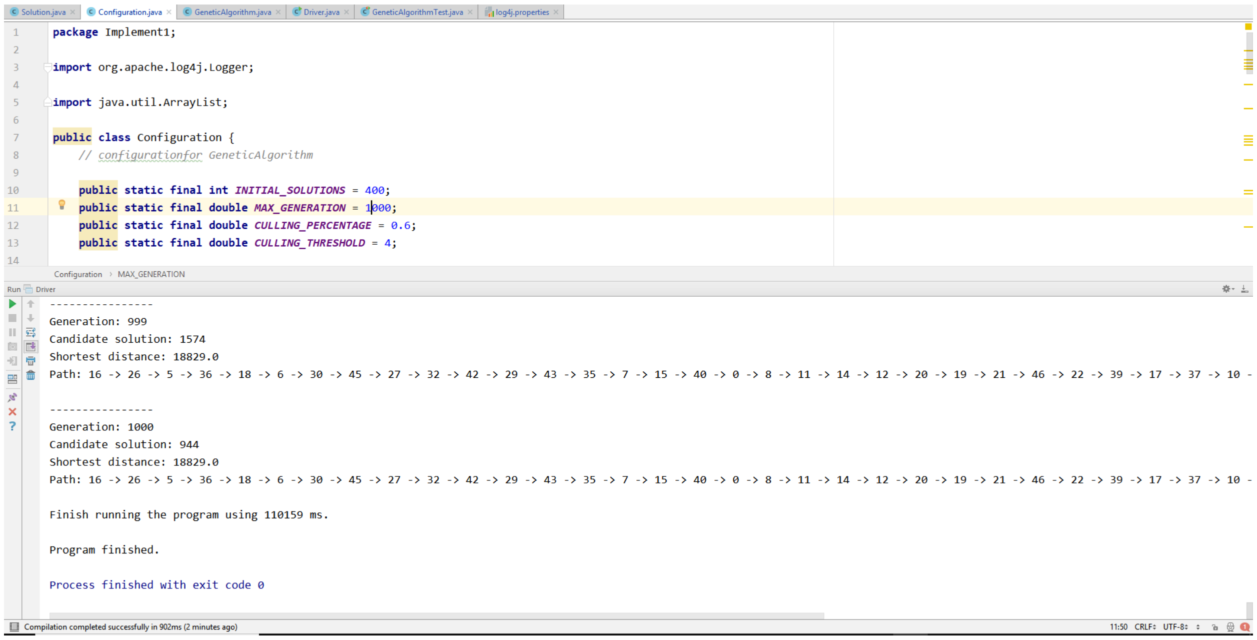
Result 3


Solution.java
Configuration.java
GeneticAlgorithm.java
Driver.java
GeneticAlgorithmTest.java
log4j.properties
Configuration › INITIAL_SOLUTIONS
Run Driver
Compilation completed successfully in 934ms (a minute ago)
5998:24 CRLF UTF-8

```java
package Implement1;

import org.apache.log4j.Logger;

import java.util.ArrayList;

public class Configuration {
    // configurationfor GeneticAlgorithm

    public static final int INITIAL_SOLUTIONS = 200;
    public static final double MAX_GENERATION = 1000;
    public static final double CULLING_PERCENTAGE = 0.6;
    public static final double CULLING_THRESHOLD = 4;

```

```
----------------
Generation: 999
Candidate solution: 523
Shortest distance: 16573.0
Path: 34 -> 44 -> 3 -> 25 -> 23 -> 41 -> 4 -> 9 -> 1 -> 28 -> 47 -> 31 -> 38 -> 12 -> 20 -> 24 -> 13 -> 15 -> 40 -> 22 -> 45 -> 32 -> 27 -> 17 -> 35 -> 6 -> 5 -> 43 -> 37 -> 29 -> 18 ->

----------------
Generation: 1000
Candidate solution: 627
Shortest distance: 16573.0
Path: 34 -> 44 -> 3 -> 25 -> 23 -> 41 -> 4 -> 9 -> 1 -> 28 -> 47 -> 31 -> 38 -> 12 -> 20 -> 24 -> 13 -> 15 -> 40 -> 22 -> 45 -> 32 -> 27 -> 17 -> 35 -> 6 -> 5 -> 43 -> 37 -> 29 -> 18 ->

Finish running the program using 28793 ms.

Program finished.

Process finished with exit code 0
```

Result 4


Solution.java
Configuration.java
GeneticAlgorithm.java
Driver.java
GeneticAlgorithmTest.java
log4j.properties
Configuration › MAX_GENERATION
Run Driver
Compilation completed successfully in 902ms (2 minutes ago)
11:50 CRLF UTF-8

```java
package Implement1;

import org.apache.log4j.Logger;

import java.util.ArrayList;

public class Configuration {
    // configurationfor GeneticAlgorithm

    public static final int INITIAL_SOLUTIONS = 400;
    public static final double MAX_GENERATION = 1000;
    public static final double CULLING_PERCENTAGE = 0.6;
    public static final double CULLING_THRESHOLD = 4;

```

```
----------------
Generation: 999
Candidate solution: 1574
Shortest distance: 18829.0
Path: 16 -> 26 -> 5 -> 36 -> 18 -> 6 -> 30 -> 45 -> 27 -> 32 -> 42 -> 29 -> 43 -> 35 -> 7 -> 15 -> 40 -> 0 -> 8 -> 11 -> 14 -> 12 -> 20 -> 19 -> 21 -> 46 -> 22 -> 39 -> 17 -> 37 -> 10 -

----------------
Generation: 1000
Candidate solution: 944
Shortest distance: 18829.0
Path: 16 -> 26 -> 5 -> 36 -> 18 -> 6 -> 30 -> 45 -> 27 -> 32 -> 42 -> 29 -> 43 -> 35 -> 7 -> 15 -> 40 -> 0 -> 8 -> 11 -> 14 -> 12 -> 20 -> 19 -> 21 -> 46 -> 22 -> 39 -> 17 -> 37 -> 10 -

Finish running the program using 110159 ms.

Program finished.

Process finished with exit code 0
```

Conclusion

1. From the results and experiments we have done here, we can see that the genetic algorithms do have a boost for finding a better path and a better solution. But it is obvious that we cannot find the best solution for this “tsp” problem.

2. Also from the test results we can find that, as the mutation of individuals has no direction, so sometimes the mutation even makes the result get worse.

3. The number of original paths and the times of generation do have the influence on finding the better solution, bigger the value, better the solution for this problem. Because the mutation will influence the result of the iteration as it has no direction, so for each original path we get different results because of these unsure factors.