**<Traveling Salesman Problem>**

**Genetic Algorithm Report**

**<13th, Apr 2018>**

**<Team 512>**

**<Yangyang Yi & Lu Bai>**

**Program Structure & Algorithms**

**Northeastern University**

Table of Contents

[1. Problem Description 3](#_Toc511499348)

[2. Methodology 3](#_Toc511499349)

[3. Basic Functions 3](#_Toc511499350)

[3.1 Initialize 3](#_Toc511499351)

[3.1.1 Genetic Code 3](#_Toc511499352)

[3.1.2 Gene Expression 4](#_Toc511499353)

[3.2 Select 4](#_Toc511499354)

[3.2.1 Fitness function 4](#_Toc511499355)

[3.2.2 Sort function 4](#_Toc511499356)

[3.2.3 Select Best Individual function 4](#_Toc511499357)

[3.2.4 Roulette Select function 4](#_Toc511499358)

[3.3 Evolution 5](#_Toc511499359)

[3.3.1 CrossOver 5](#_Toc511499360)

[3.3.2 Mutation 5](#_Toc511499361)

[4. Flow chart of TSP 6](#_Toc511499362)

[5. Result Display 7](#_Toc511499363)

[6. Test Case 9](#_Toc511499364)

[7. Conclusion 10](#_Toc511499365)

# Problem Description

The problem we focus on is traveling salesman, which is given a list of cities and the distances between each pair of cities, finding the shortest possible route that visits each city and returns to the origin city.

# Methodology

The Methodology we used to solve this problem is Genetic Algorithm. It simulates the process of natural selection, which includes initialization, selection, mutation and crossover. After a certain amount of generation replacement, eventually a best result will appear.

# Basic Functions

## Initialize

### Genetic Code

Our genetic type is represented in binary number, and we write a function to randomly generate binary numbers and calculate how many digits should be generated depends on the number of the city. A gene represents one city and an array of gene represents one route. In addition, to avoid duplicate genes, we check if it already has the same gene in this route when generate binary number.

### Gene Expression

To help translate the gene to the city location, we write a transfer function to transfer binary number to decimal number. In our program, we have an individual class to stand for one route. Each individual has their own properties like *binaryCity* array, *decimalCity* array, and distance of this route. Furthermore, we have a population class to store individual.

## Select

### Fitness function

A ***fitness()*** function is applied to calculate each route’s fitness according to their distance, and the distance we calculate is based on the formula . For example, a fitness of a route is (1/distance of the route).

### Sort function

***Mergesort()*** function will be used to sort the fitness, which can help the program to get the best route of current generation.

### Select Best Individual function

***SelectBest()*** function is the one that we used to select the top 10% routes according to the routes’ fitness.

### Roulette Select function

***rouletteSelect()*** function will be used to generate the parents for crossover. It select the route by calculate the likelihood of their fitness, for example, route 1 ‘s likelihood to be chosen is (route 1’s fitness/sum of all fitness of current generation). This method of choosing parents can ensure that good fitness of the individual will have greater probability to be chosen as a parent to crossover.

## Evolution

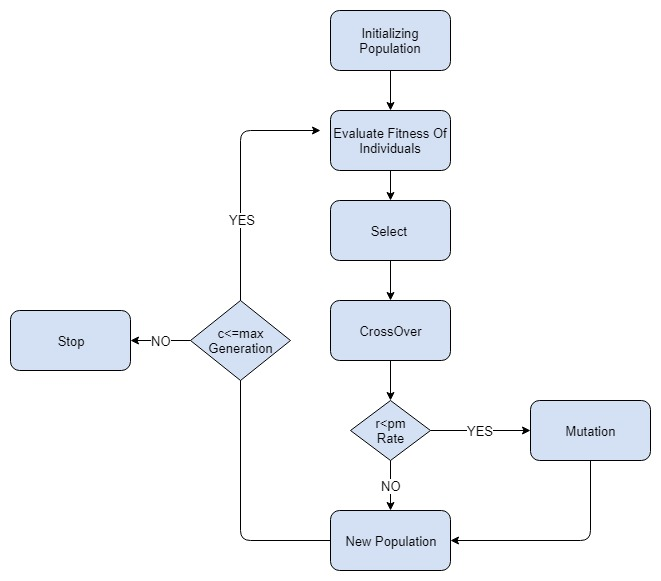
### CrossOver

In the ***crossover (Individual p1, Individual p2)*** function, Each individual indicate a route. 2 individuals will need to pass into it to be the parents, the way we do the crossover is firstly get 2 random numbers, mark them as start position and end position to get partial genes from parent p1. Then the rest of the genes will be obtained from parent p2. Eventually, we will get a new Individual(route) which combines the parents’ gene.

### Mutation

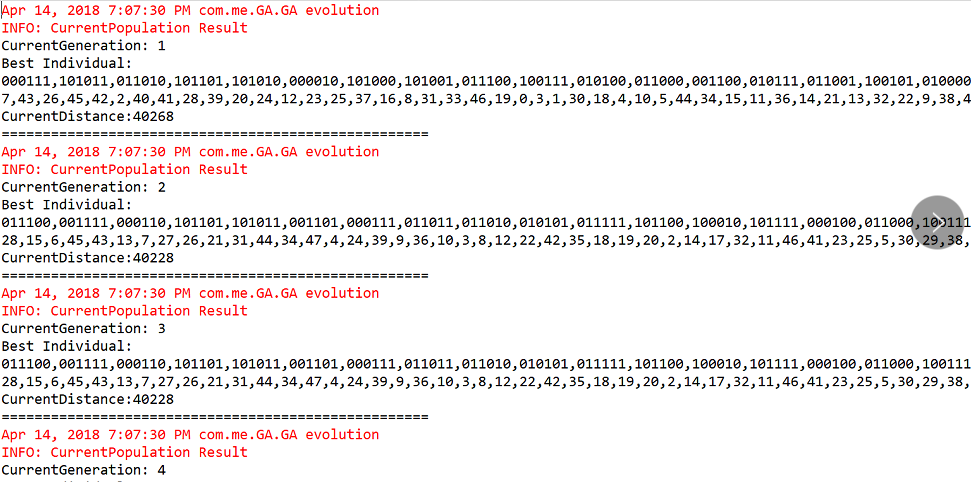
After *crossover()* function generated a child, a random number will be generated and then compared with the mutation probability of the population to decide whether the child need to mutation. If the random number is smaller than the mutation probability, this child will be pass into the ***mutation(Individual in)*** function. The ***mutation(Individual in****)* function is basically get 2 random number as index of the array of the route, swap these 2 cities. Then we will get a new Individual(route) after mutation.

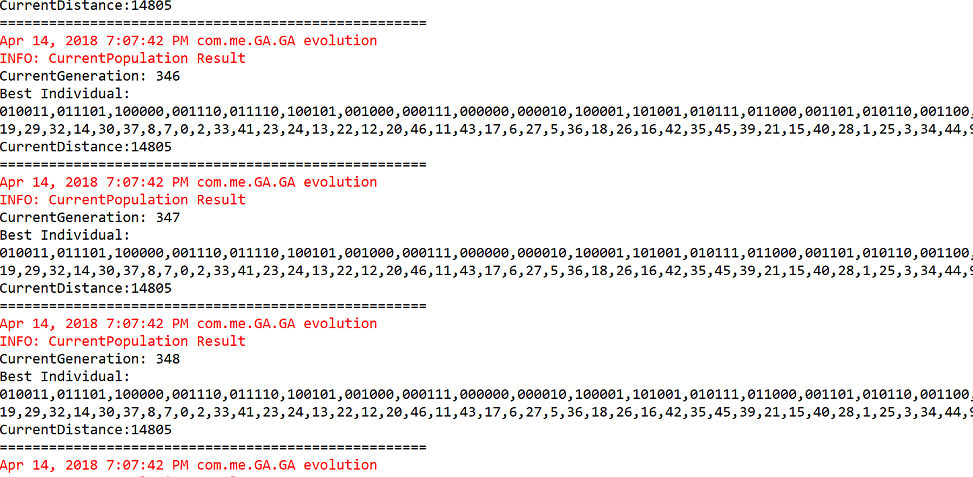
# Flow chart of TSP

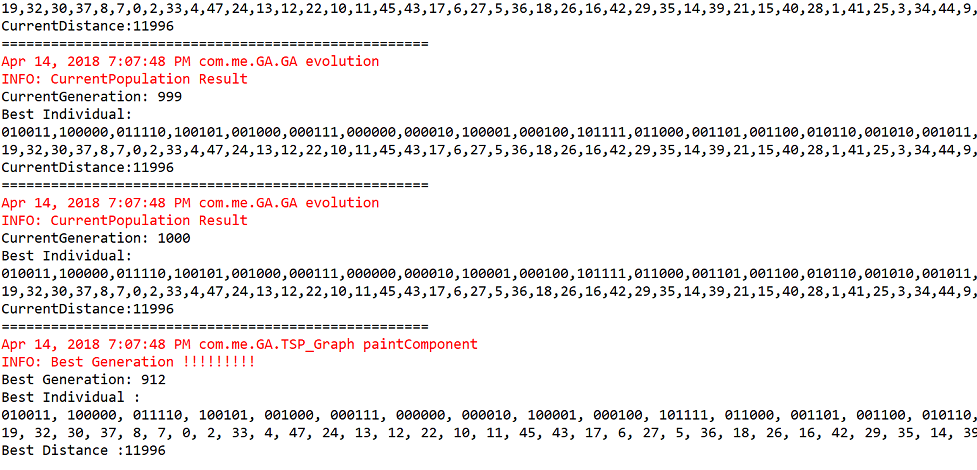


The flow chart listed above is shown the process of our program. At first, we will randomly initialize a population. Then, we will calculate the fitness of each individual and do select method. After selecting the parents, it will do cross over and generate one child. Furthermore, we will random a float number and check it with the mutation rate. If it less than the rate, child will do mutation. Vice versa. In the end, we generate the new population and do the loop until the generation times is equal or lesser than the time you want to do.

# Result Display



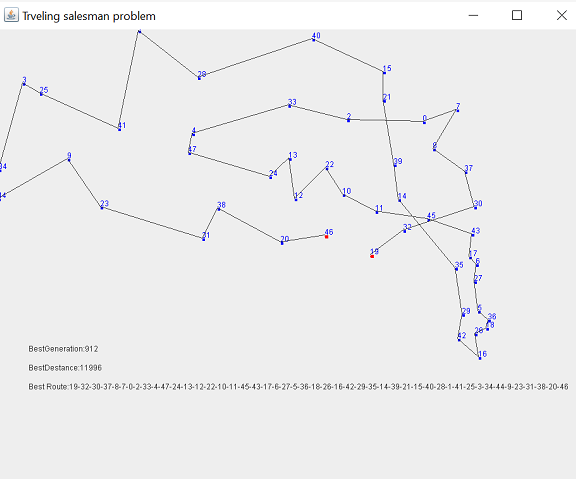




The pictures listed above shows the evolution of the population, we set the population size to be 500, and the max generation is 1000. We can see that in the first generation, the best route’s distance is 40268. However, after the evolution, the best generation appears in 912, and the distance became 11996, which shows that the best distance did not change after the generation 912.

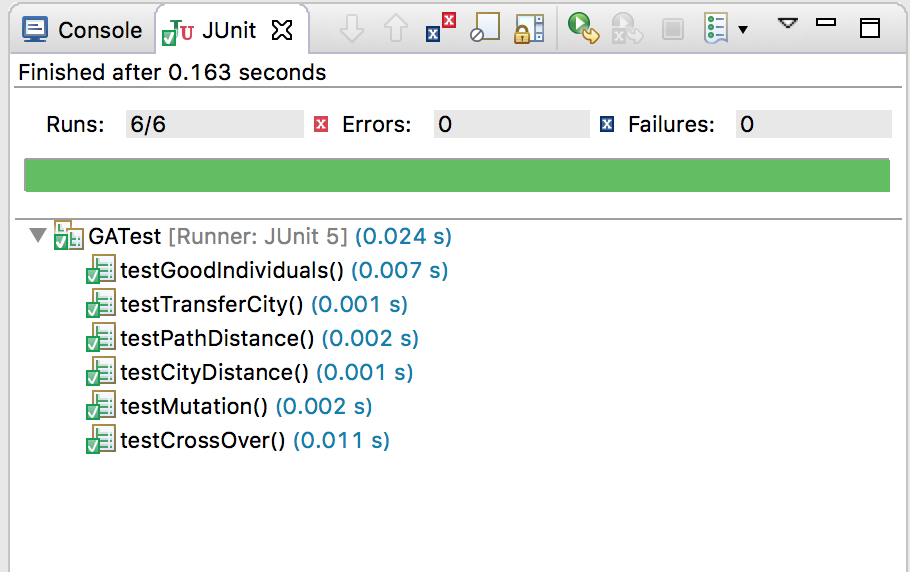
Furthermore, we also use JPanel to draw the best route, the red point is the start and end point. Each blue point indicates a city.

The picture of the path list as follow:



# Test Case

We write 6 test cases to test our program, and all of them has passed the test.



# Conclusion

The purpose of our project is using Genetic Algorithm to find the shortest path that loop all the cities without duplication. The result from our program is approximately to the real optimal shortest path of the data we obtained from internet. Since we only have 48 cities, we initiate the total route number to 500 which is the population size in our program. Moreover, we set the max generation to be 1000. As a result, we can get the best result from our program since approximately 900th generation. According to our log, we can see that the result became closer to the real shortest path by each evolution.

However, the result is varying but not much differences when we run the program, we think it is hard to find the real shortest path by genetic algorithm, because the result we get might be a local optimal result, rather than an overall optimal result. And why this situation happed is that it is hard to generate a new genetic after a certain amount of generation from our opinion. Even though the result might not be the best, it is already a very good result we think.

As a result, we deem that genetic algorithm is worth doing to acquire a good result from traveling salesman problem.