**Final Project--Genetic Algorithm: The Traveling Salesman Problem**

**Team Number**: INFO6205_306
**Team Member**: Heng Wang 001822598
Yize Wang 001822633
Dingting Huang 001824907

**Problem**

Genetic algorithms are evolutionary techniques used for optimization purposes according to the survival of the fittest idea. In this project, we created a genetic algorithm to find a solution to the Traveling Salesman Problem. The content of this research project: given a set of randomly generated cities and a distance between each of them, there is a salesman whose job is to visit each of these cities and make a sell, he needs to plan a road so he can try his best to minimize his travel time and miles. He starts from his hometown and he will back to his hometown after work. We build a model based on sexual reproduction(not only crossover, also mutation).

**Implementation Design**

*Genotype:* Each gene actually is a point in the map, which has their own x and y position. A chromosome is a specific traveling route which contains a special order of all points in the map. Since each city will be arrived just once, a chromosome will just contain each point once.

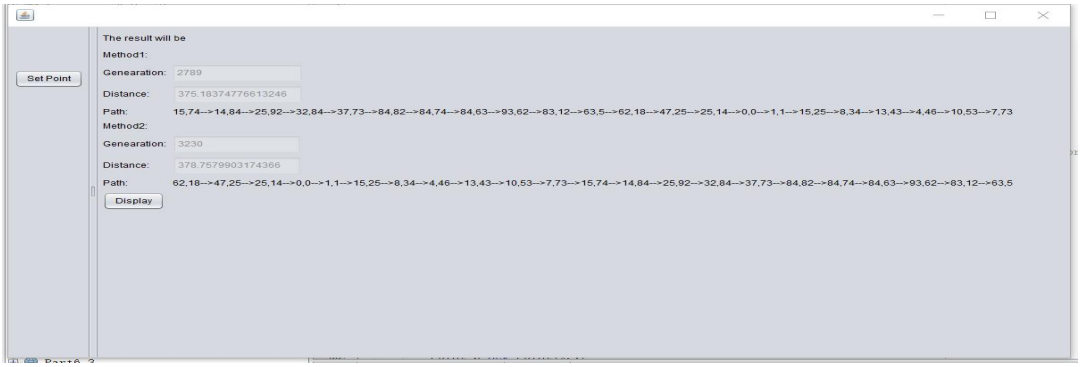
*Phenotype:* Since each chromosome is a route, its phenotype will be the total distance of this route. By the way, one interesting thing is that two different chromosomes may have same phenotype. For example, a chromosome which has gene order of (1-2-3-4-5) and a chromosome has gene order of (3-4-5-1-2) will have the same phenotype since their distances are the same.

*Gene Expression:* In order to connect genotype and phenotype, we use the distance calculated function e(g)= Σ(edge weights) as the gene expression.

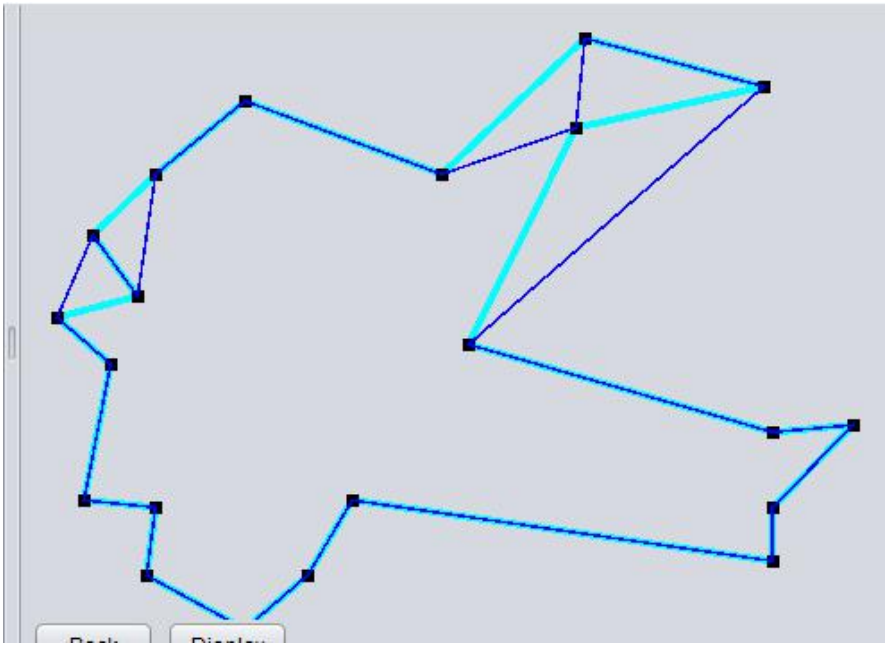
*Fitness function:* Decided by path length, f(g) = 1/w(g) where w(g) = Σ(edge weights). A cycle with a smaller weight will have a larger fitness value and chance of survival.

*GA Thread:* For each program running, a ThreadRunning class will be called which creates two separate threads and uses different crossover methods(OX, OBX) to finish the evolution process.

*UI:*



After user input his position, he will be able to see the result calculated by two crossover methods. The result contains the optimal path, minimum distance and the generation.



After click on the display button, a simple map will be shown. In the above figure, black points represent cities, the blue line is the result of crossover mehod2(OBX) and the cyan line is the result of crossover method1(OX)

Instead of seeing the TSP as a simple and pure genetic algorithm, we give it more depth. We make use of multi-thread implement method and several crossover approaches to surmount effectively the local convergence problem of the standard genetic algorithm and improves the test generation speed. However, this program was born out of the genetic algorithm, and its structure can be defined in the following steps：

Step 1. Create an initial population of P chromosomes.
Step 2. Evaluate the fitness of each chromosome.
Step 3. Choose P/2 parents from the current population via proportional selection.
Step 4. Randomly select two parents to create offspring using crossover operator.
Step 5. Apply mutation operators for minor changes in the results.
Step 6. Combine new generation with old one.
Step 7. Determine whether the gene is from the new generation or not. If it is the new generation, it will not participate in the crossover.
Step 8. Evaluate the fitness of each chromosome in the new population.
Step 9.Determine whether the current number of population is larger than the capacity of the environment and remove individuals with less fitness
Step 10. Repeat step 3, 4, 5, 6, 7, until it reaches the maximum number of generation or the best gene has not changed for 2000 generations.

*Evolution*: Given some initial values as the first generation, the population will evolve with sexual reproduction, each increase is half of the last population. Each child will experience two different crossover method and random variation.

Previously, we should calculate the value of fitness in advance. That is,f(g) = 1/w(g) where w(g) = Σ(edge weights). A cycle with a smaller weight will have a larger fitness value and chance of survival. I perform an insertion sort on the array and the environmental capacity set at 1000, no matter how many offspring have existed, based on environment capacity and fitness value, the best 1000 genes will leave and form the new generation (apart from the latest generation).

Firstly, the mutation method should only be capable of shuffling the route, it shouldn't ever add or remove a location from the route, otherwise it would risk creating an invalid solution. One type of mutation method we could use is swap mutation.

With swap mutation two location in the route are selected at random then their positions are simply swapped. For example, if we apply swap mutation to the following list, [1,2,3,4,5,6,7] we might end up with, [1,2,6,4,5,3,7].

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Parent** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **Child** | **1** | **2** | **6** | **4** | **5** | **3** | **7** |

Here, positions 3 and 6 were switched creating a new list with exactly the same values, just a different order. Because swap mutation is only swapping pre-existing values, it will never create a list which has missing or duplicate values when compared to the original.

Now we've dealt with the mutation method we need to pick a crossover method which can enforce the same constraint. In this segment, we build two different crossover method to probe the diversification of environment, because one single way can not meet the evolution strategy needs.

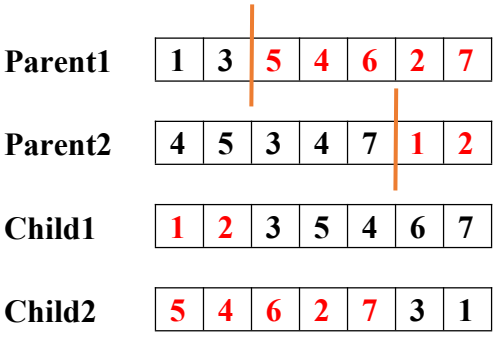
- Order Crossing(OX)

In this crossover method we select a subset from the first parent, and then add that subset to the offspring. Any missing values are then adding to the offspring from the second parent in order that they are found.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Parent1** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **Parent2** | **7** | **6** | **5** | **4** | **3** | **2** | **1** |
| **Child1** | **3** | **2** | **1** | **4** | **5** | **6** | **7** |
| **Child2** | **5** | **6** | **7** | **4** | **3** | **2** | **1** |

- Order-Based Crossing(OBX)

In this crossover method we randomly select different subsets from the different parents, such as the following mapping. Here a subset of route is taken from parent1(5,4,6,2,7) and add it to the child2's route. Next, the missing route locations are added in order from parent2 (3,1).




**Results&Conclusion**

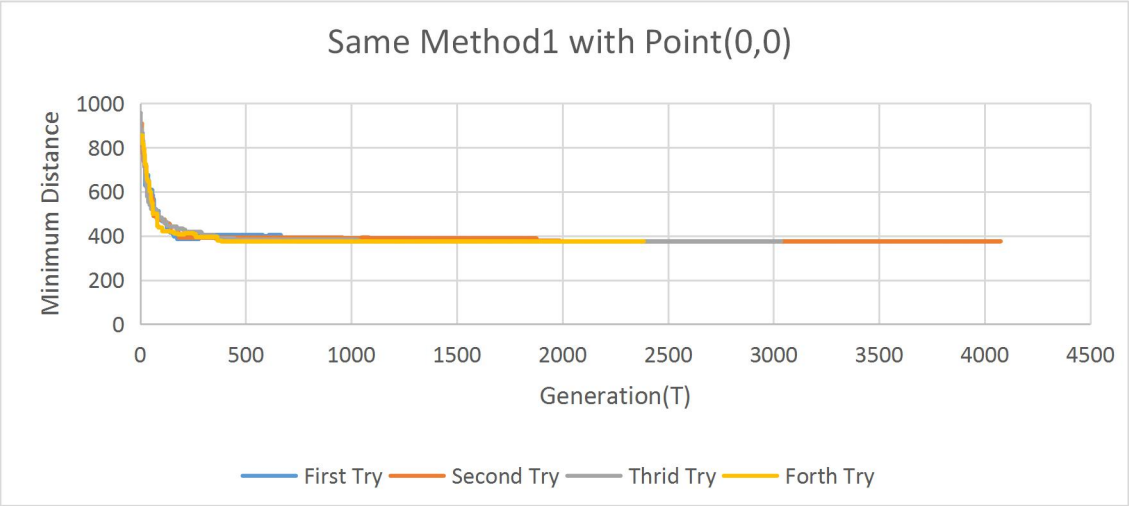
Results of executing unit test

```
Building FinalProject 1
------------------------------------------------------------------------

--- maven-resources-plugin:2.5:resources (default-resources) @ FinalProject ---
[debug] execute contextualize
Using 'UTF-8' encoding to copy filtered resources.
Copying 0 resource

--- maven-compiler-plugin:2.3.2:compile (default-compile) @ FinalProject ---
Nothing to compile - all classes are up to date

--- maven-resources-plugin:2.5:testResources (default-testResources) @ FinalProject ---
[debug] execute contextualize
Using 'UTF-8' encoding to copy filtered resources.
skip non existing resourceDirectory F:\6205 Program Structure&Algorithms\Final Project\123\FinalProject\src\test\resources

--- maven-compiler-plugin:2.3.2:testCompile (default-testCompile) @ FinalProject ---
Nothing to compile - all classes are up to date

--- maven-surefire-plugin:2.10:test (default-test) @ FinalProject ---
Surefire report directory: F:\6205 Program Structure&Algorithms\Final Project\123\FinalProject\target\surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.me.test.EnvironemtTest
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.138 sec
Running com.me.test.GeneTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.002 sec
Running com.me.test.GraphTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.002 sec
Running com.me.test.InsertionSortTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.001 sec
Running com.me.test.PointTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.001 sec

Results :

Tests run: 9, Failures: 0, Errors: 0, Skipped: 0

------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 2.836s
Finished at: Sat Apr 14 21:18:38 EDT 2018
Final Memory: 8M/245M
```

In order to check our two crossover methods and compare their performance,we use two different variables(initial place and different crossover method) to do three kinds of tests.

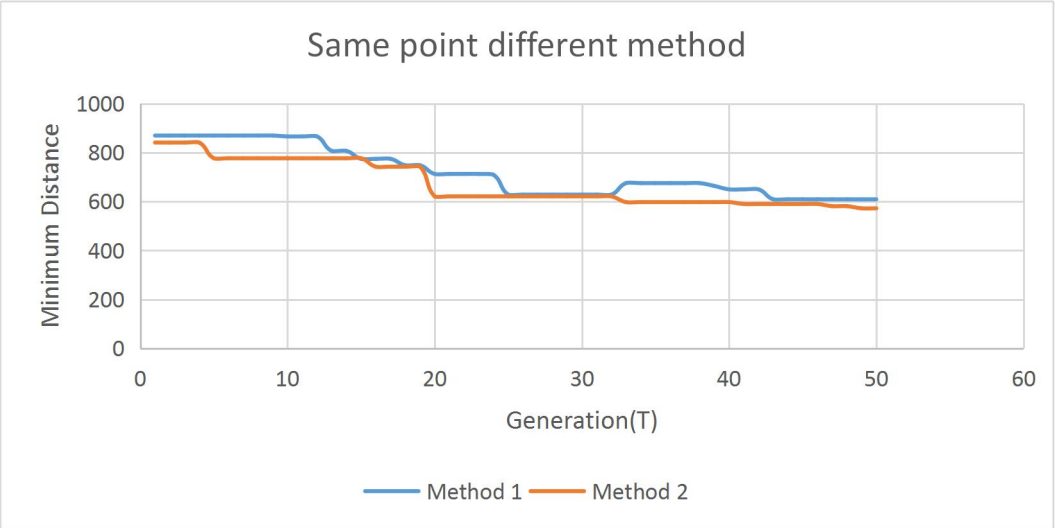
- **Same initial vertice, same crossover method**

I consistently ran the algorithm for graphs four times with 20 vertices from the start point(0,0). These running results is very close and always get the best distance. Note: Because of the existence of mutation, the difference will naturally arise, nevertheless, the evolutionary direction will converge to a global optimum solution.




- **Same initial vertice, different crossover methods**

We can find clearly the conclusion, the second method runs better than the first method. The difference between two types of crossover operators is that one is simple order crossing(OX), another is improved. The same chromosome will produce different progeny, in fact, these two method has no too much effect on the obtained results.




- **Different initial vertices, same crossover method**

One interesting effect is the genetic algorithm has certain reliance on the initial points. The greater the probability of the effective gene of a chromosome in the initial population, the higher the adaptability. If the initialization method is effective, the initial population can be concentrated to the optimal solution of the adjacent regions, and the search speed will be improved.