Genetic Algorithms – Travelling Salesmen Vinyas Kaushik Tumakunte Raghavendrarao – 001216716
Team Number – 525 Kiran Chamaraja – 001813587

README

To develop the genetic algorithm for solving the Travelling Salesman Problem, we implemented 5 Classes: Genotype, TraversalPath, Population, City and Driver class.

Genotype mainly focus on defining the DNA and how the DNA was translated to real point of locations and the Crossover/ Mutate process. The Population class consists of methods to initialize the population, keeps list of genotypes and base order in which the initial list of cities is generated. TraversalPath contains methods to generate phenotype and methods to calculate fitness score and the distance. Driver class runs the main method to generate the population maintain generations and also contains a method which creates random cities for base order.

Implementation:

Considering the Population Class it contains five major methods: Constructor, initializePopulation, sortPopulation, regeneration and crossover. It also contains some other supplement methods like genoTypeComparator and getRandomGenotypeIndex.

- InitializePopulation method is used to initialize the initial population randomly. It creates Genotypes which has a 4-bit string representation of binary number of each city.
- SortPopulation method is used to sort the population based on the fitness score generated for each phenotype.
- Regeneration method is used to generate new population or generation using crossover method by choosing random parents from the population. The population is maintained constant in the process.
- Mutation – the population is sorted based on the fitness function and mutation is induced every 3 generations once on the population where the fitness function is highest.

In the Genotype Class, we are using two different constructors two differ the type to initialize a phenotype. The class contains phenotype generation method and compareTomethod.

- Phenotype is generated by converting the 4-bit binary code into index to get the city from the baseOrder ArrayList.
- CompareTo method is used to compare two genotypes.

In TraversalPath Class, we are having CalculateFitnessScore and CalculateDistance methods.

- CalculateFitnessScore is used to calculate FitnessScore based on the total distance calculated in CalculateDistance method. FitnessScore is obtained by using the follow formula. FitnessScore = (1/TotalDistance).
- CalculateDistance method is used to calculate Distance between a pair of cities using the xcoordinate and ycoordinate of the cities.

In Driver Class, we have implemented the main function which runs the whole algorithm. Along with the main function generateBaseOrder method is implemented in the driver class.

- Array of Random order of cities and along with (X,Y) coordinate is generated using generateBaseOrder.

Vinyas Kaushik Tumakunte Raghavendrarao – 001216716
Kiran Chamaraja – 001813587

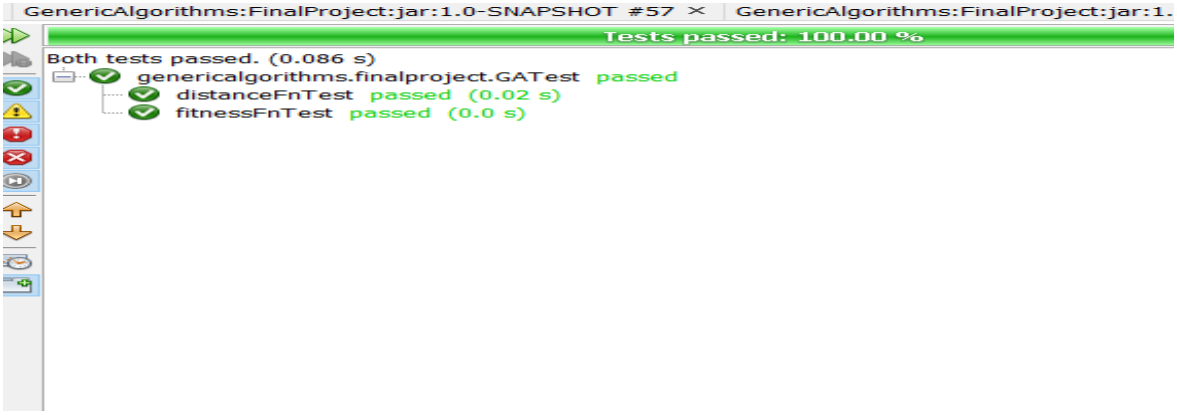
Conclusion

Our problem is to find a shortest path to access all the points without duplicate by using genetic algorithm. According to our tests, the genetic algorithm will generate a result which is quite close to the optimal result. In a relative small size of problem (5 locations), the program generates the result which is close to the optimal result (or maybe is the optimal result) in less than 10 generations (rare improvement after that) but in a larger size of problem (15 locations), the program will need more generations to reach a result (the program took around 24 generations)which is close to the optimal. After several generations, the best DNA remained unchanged and the crossover between other DNA type influenced little to the result which is a huge problem remaining unsolved.

The genetic algorithm in some cases can help generate relative optimal result for those with a high time complexity. But as it is hard to know whether the result is optimal or not, it would require extra effort to prove that. We consider the genetic algorithm will sink into the situation that, its result is only locally optimal solution not overall optimal, it requires a lot of space and calculation to approach a optimal result. Its result is also not stable, usually we will need to run the program several times to know whether the result is close to the optimal.

In conclusion, we believe the genetic algorithm is a wonderful way to get a result close to the optimal solution. With other constrains and optimization, this algorithm will help deal with various kinds of problems.

Test Cases Screen Shot.



Genetic Algorithms – Travelling Salesmen Vinyas Kaushik Tumakunte Raghavendrarao – 001216716
Team Number – 525 Kiran Chamaraja – 001813587

Test Results:

First Run

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ FinalProject ---
Please enter the Number of Cities:
5
Please enter Population Number:
10
Best possible Result in each Generation
Generation 0 <Order 3 : City{name='City1'}->City{name='City4'}->City{name='City0'}->City{name='City3'}->City{name='City2'}>< distance : 68.90623921551071>< fitness score : 0.0019433697007601198>
Generation 1 <Order 9 : City{name='City0'}->City{name='City3'}->City{name='City2'}->City{name='City1'}->City{name='City4'}>< distance : 32.255029339488786>< fitness score : 0.0019433697007601193>
Generation 2 <Order 2 : City{name='City2'}->City{name='City1'}->City{name='City4'}->City{name='City0'}->City{name='City3'}>< distance : 110.5007135113756>< fitness score : 0.0019433697007601193>
Generation 3 <Order 6 : City{name='City1'}->City{name='City2'}->City{name='City3'}->City{name='City0'}->City{name='City4'}>< distance : 145.13237401120517>< fitness score : 0.0019433697007601198>
Generation 4 <Order 1 : City{name='City2'}->City{name='City1'}->City{name='City4'}->City{name='City0'}->City{name='City3'}>< distance : 110.5007135113756>< fitness score : 0.0019433697007601193>
Generation 5 <Order 3 : City{name='City2'}->City{name='City1'}->City{name='City4'}->City{name='City0'}->City{name='City3'}>< distance : 110.5007135113756>< fitness score : 0.0019433697007601193>
Generation 6 <Order 4 : City{name='City2'}->City{name='City1'}->City{name='City4'}->City{name='City0'}->City{name='City3'}>< distance : 110.5007135113756>< fitness score : 0.0019433697007601193>
Generation 7 <Order 0 : City{name='City2'}->City{name='City1'}->City{name='City4'}->City{name='City0'}->City{name='City3'}>< distance : 110.5007135113756>< fitness score : 0.0019433697007601193>
Generation 8 <Order 4 : City{name='City2'}->City{name='City1'}->City{name='City4'}->City{name='City0'}->City{name='City3'}>< distance : 110.5007135113756>< fitness score : 0.0019433697007601193>
Generation 9 <Order 5 : City{name='City1'}->City{name='City0'}->City{name='City4'}->City{name='City2'}->City{name='City3'}>< distance : 136.9299378891978>< fitness score : 0.0015871293049699658>
Marginal Change in the fitness hence stopping the generation
------------------------------------------------------------------------
BUILD SUCCESS
```

Second Run

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ FinalProject ---
Please enter the Number of Cities:
15
Please enter Population Number:
1000
Best possible Result in each Generation
Generation 0 <Order 274 : City{name='City14'}->City{name='City7'}->City{name='City11'}->City{name='City10'}->City{name='City9'}->City{name='City1'}->City{name='City4'}->City{name='City3'}->City{n
Generation 1 <Order 525 : City{name='City4'}->City{name='City6'}->City{name='City2'}->City{name='City13'}->City{name='City3'}->City{name='City9'}->City{name='City1'}->City{name='City12'}->City{na
Generation 2 <Order 10 : City{name='City10'}->City{name='City14'}->City{name='City7'}->City{name='City5'}->City{name='City3'}->City{name='City12'}->City{name='City0'}->City{name='City13'}->City{n
Generation 3 <Order 849 : City{name='City14'}->City{name='City3'}->City{name='City2'}->City{name='City13'}->City{name='City12'}->City{name='City4'}->City{name='City0'}->City{name='City6'}->City{n
Generation 4 <Order 160 : City{name='City10'}->City{name='City5'}->City{name='City8'}->City{name='City11'}->City{name='City6'}->City{name='City2'}->City{name='City0'}->City{name='City13'}->City{n
Generation 5 <Order 75 : City{name='City14'}->City{name='City9'}->City{name='City6'}->City{name='City12'}->City{name='City10'}->City{name='City13'}->City{name='City0'}->City{name='City3'}->City{n
Generation 6 <Order 304 : City{name='City5'}->City{name='City14'}->City{name='City11'}->City{name='City10'}->City{name='City9'}->City{name='City1'}->City{name='City3'}->City{name='City12'}->City{
Generation 7 <Order 847 : City{name='City3'}->City{name='City0'}->City{name='City2'}->City{name='City6'}->City{name='City13'}->City{name='City1'}->City{name='City5'}->City{name='City7'}->City{nam
Generation 8 <Order 637 : City{name='City11'}->City{name='City8'}->City{name='City7'}->City{name='City6'}->City{name='City12'}->City{name='City5'}->City{name='City4'}->City{name='City13'}->City{n
Generation 9 <Order 713 : City{name='City11'}->City{name='City12'}->City{name='City2'}->City{name='City4'}->City{name='City7'}->City{name='City8'}->City{name='City5'}->City{name='City13'}->City{n
Generation 10 <Order 590 : City{name='City13'}->City{name='City0'}->City{name='City12'}->City{name='City11'}->City{name='City9'}->City{name='City10'}->City{name='City8'}->City{name='City7'}->City
Generation 11 <Order 498 : City{name='City9'}->City{name='City10'}->City{name='City11'}->City{name='City1'}->City{name='City6'}->City{name='City13'}->City{name='City0'}->City{name='City3'}->City
Generation 12 <Order 426 : City{name='City2'}->City{name='City13'}->City{name='City0'}->City{name='City3'}->City{name='City6'}->City{name='City12'}->City{name='City5'}->City{name='City7'}->City{n
Generation 13 <Order 645 : City{name='City7'}->City{name='City5'}->City{name='City14'}->City{name='City11'}->City{name='City1'}->City{name='City3'}->City{name='City4'}->City{name='City6'}->City{n
Generation 14 <Order 205 : City{name='City2'}->City{name='City13'}->City{name='City0'}->City{name='City3'}->City{name='City6'}->City{name='City12'}->City{name='City5'}->City{name='City7'}->City{n
Generation 15 <Order 938 : City{name='City11'}->City{name='City14'}->City{name='City7'}->City{name='City13'}->City{name='City0'}->City{name='City12'}->City{name='City4'}->City{name='City2'}->City
Generation 16 <Order 802 : City{name='City2'}->City{name='City13'}->City{name='City0'}->City{name='City12'}->City{name='City1'}->City{name='City4'}->City{name='City3'}->City{name='City5'}->City{n
Generation 17 <Order 784 : City{name='City0'}->City{name='City2'}->City{name='City3'}->City{name='City13'}->City{name='City5'}->City{name='City12'}->City{name='City9'}->City{name='City7'}->City{n
Generation 18 <Order 923 : City{name='City11'}->City{name='City10'}->City{name='City6'}->City{name='City1'}->City{name='City9'}->City{name='City14'}->City{name='City2'}->City{name='City0'}->City
Generation 19 <Order 719 : City{name='City8'}->City{name='City9'}->City{name='City14'}->City{name='City4'}->City{name='City0'}->City{name='City12'}->City{name='City13'}->City{name='City3'}->City
Generation 20 <Order 311 : City{name='City8'}->City{name='City7'}->City{name='City6'}->City{name='City1'}->City{name='City9'}->City{name='City3'}->City{name='City2'}->City{name='City13'}->City{na
Generation 21 <Order 284 : City{name='City3'}->City{name='City4'}->City{name='City1'}->City{name='City10'}->City{name='City14'}->City{name='City11'}->City{name='City5'}->City{name='City13'}->City
Generation 22 <Order 5 : City{name='City3'}->City{name='City4'}->City{name='City6'}->City{name='City8'}->City{name='City7'}->City{name='City14'}->City{name='City11'}->City{name='City1'}->City{nam
Generation 23 <Order 234 : City{name='City7'}->City{name='City5'}->City{name='City8'}->City{name='City3'}->City{name='City0'}->City{name='City2'}->City{name='City1'}->City{name='City11'}->City{na
Generation 24 <Order 698 : City{name='City7'}->City{name='City5'}->City{name='City12'}->City{name='City13'}->City{name='City3'}->City{name='City2'}->City{name='City4'}->City{name='City0'}->City{n
Marginal Change in the fitness hence stopping the generation
------------------------------------------------------------------------
```