INFO6205\_506 Final Project Report

**GA to solve TSP**

**Team 506**

**001817649 Zhifei Lin**

**001814467 Zinan Wang**

**Directory**

[**1. Background 3**](#_Toc27384)

[(1) GA 3](#_Toc28198)

[(2) Traveling Salesman Problem 3](#_Toc29569)

[(3) Our Goal 3](#_Toc31130)

[**2. The flow of GA 4**](#_Toc4931)

[**3. Implementation 4**](#_Toc23316)

[(1) Gene Code 4](#_Toc23251)

[(2) Gene Expression 5](#_Toc2559)

[(3) Fitness Function 6](#_Toc23796)

[(4) Sort Function 6](#_Toc16278)

[(5) Evolution: 9](#_Toc20063)

[**4. Results 11**](#_Toc25193)

[**5. Unit Test 17**](#_Toc30669)

[**6. Conclusion 17**](#_Toc19212)

1. Background
2. **GA**

Genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms. 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.

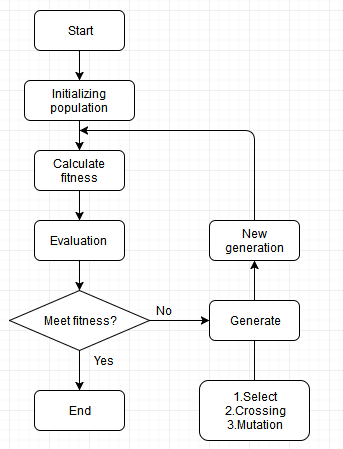
1. **Traveling Salesman Problem**

Travelling salesman Problem (TSP) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" It is an NP-hard problem in combinatorial optimization.

1. **Our Goal**

This project uses Genetic Algorithm to solve the Traveling Salesman Problem which means to find the shortest possible route for a salesman who travelled every city and back to his origin.

1. The flow of GA



First, we initialize a population randomly.

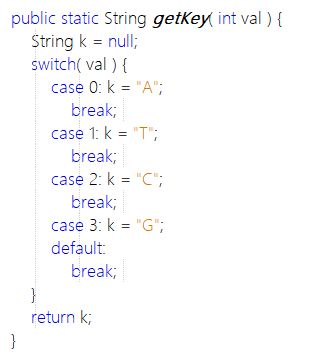
Then we calculate the individual fitness and do select function. We evaluate the fitness to figure out the results meet our goals or not.

If not, we get the child generation. By crossing and mutation(according to the mutation probability factor) and do this loop until meet goals.

1. Implementation
2. Gene Code

Our gene is coded as the four DNA bases (A, T, C, G).

We use 4 gene to show one genotype, such as [ATCG], [AGAA].



1. Gene Expression

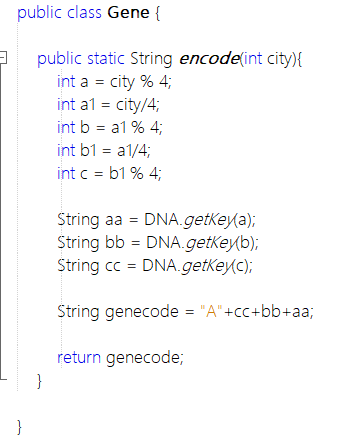
We use quaternary coding to show gene expression.

A→0， T→1， C→2, G→3.

For example:

[ATCG, ATAA, AGAC, AGAG] → [0123,0100,0302,0303] → [27,16,50,51]

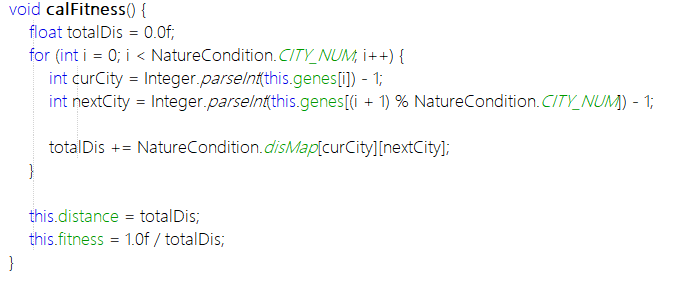
Since we don’t need so many cities, the first code of Gene is set to “A”.



1. Fitness Function

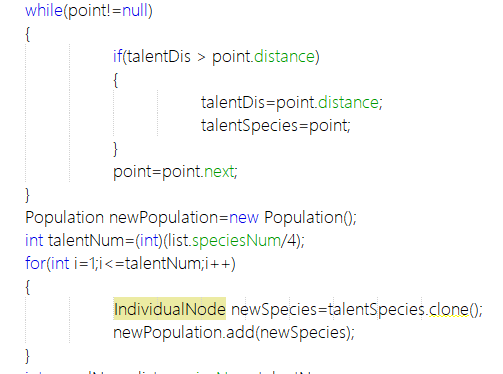
Calculate fitness of each individual:

We use the distance between two cities to show fitness. The fitness is the inverse of the sum of distance. Larger the fitness is, the better result we have.

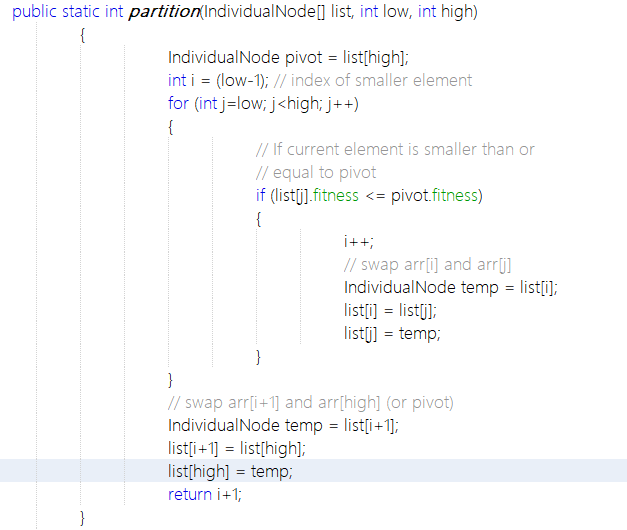


1. Sort Function

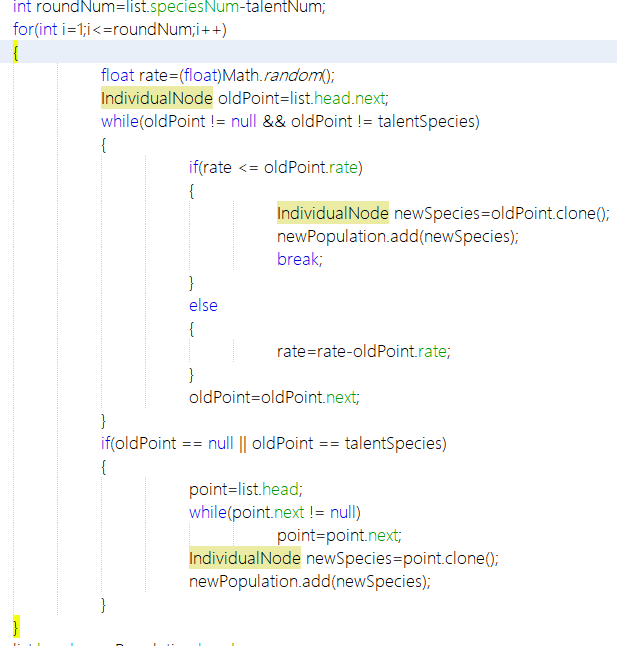
Select individuals with the highest fitness in the generation and copy them directly to the child generation. And those good individuals are also used to generate children.

  
We have two select method: Quicksort and Roulette Wheel Selection.

Quicksort:

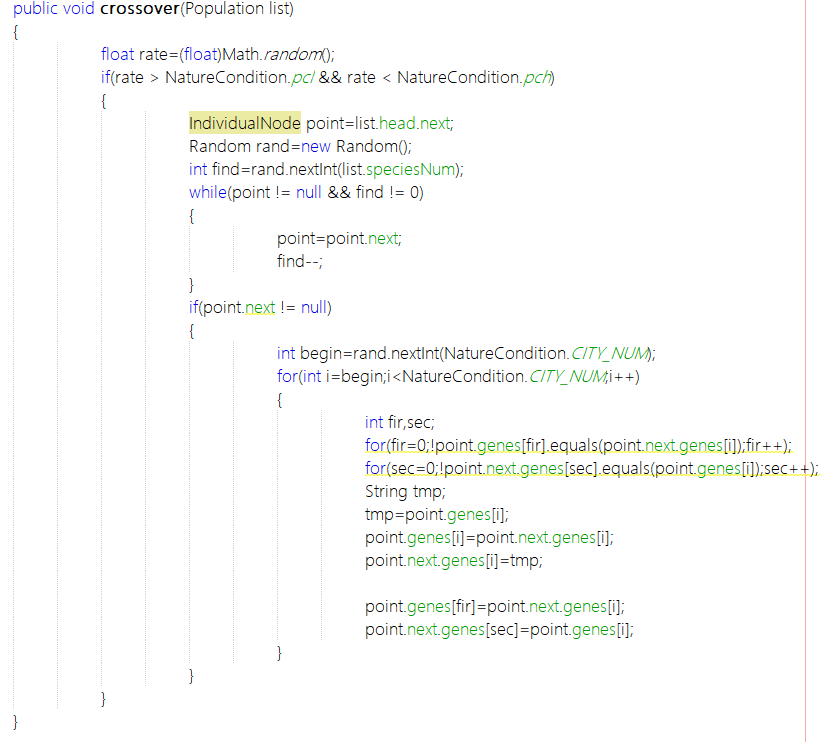


Roulette Wheel Selection:



1. Evolution:

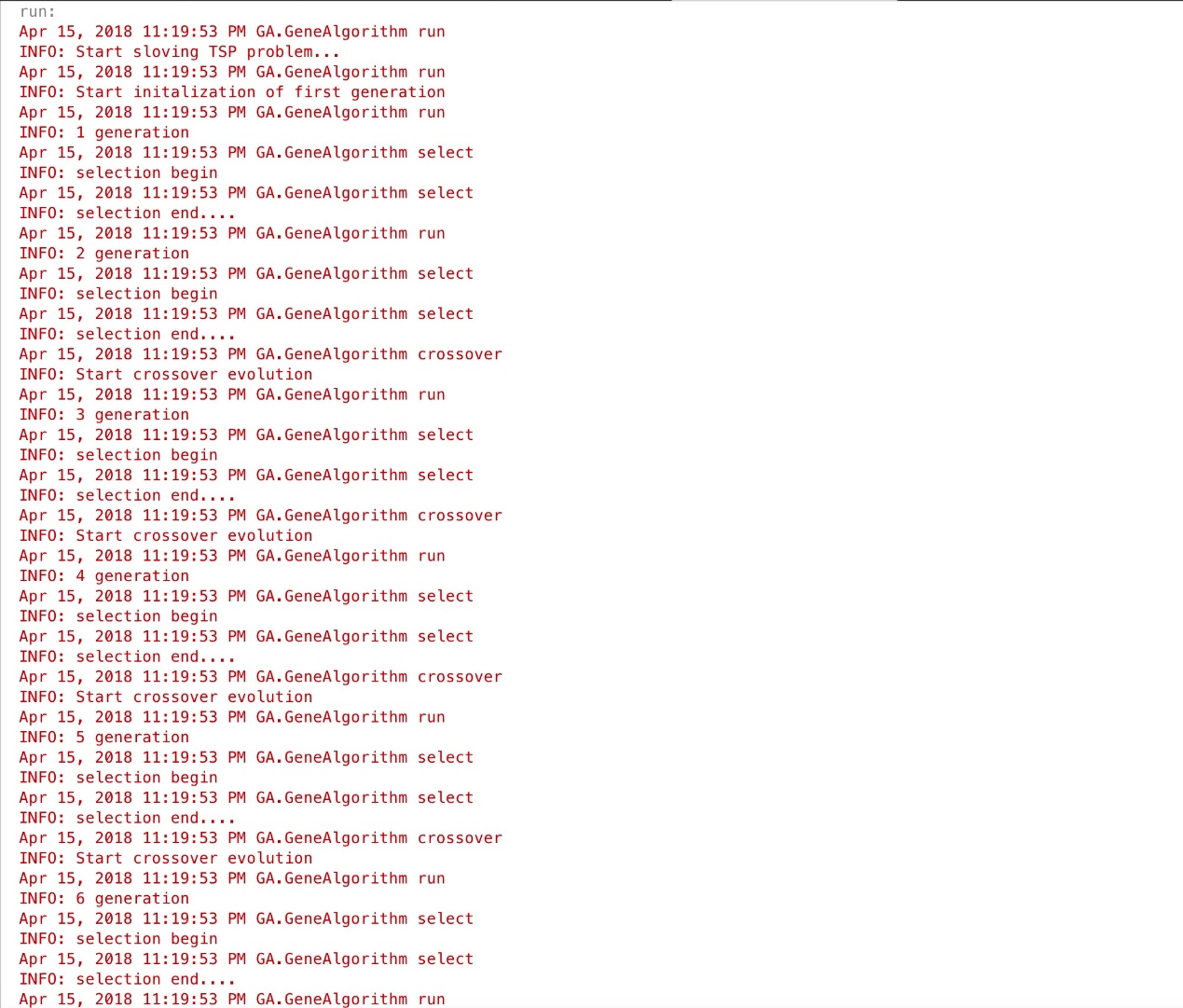
It includes crossover and mutation.   
We use linked list to store individuals, so it will reduce time when we do crossover and mutation.  
For crossover, we select two parent chromosomes and generate a random number as the crossover point. Then the parents exchange their gene to generate the child.

  
For mutation, we swap two cities which are chosen randomly.



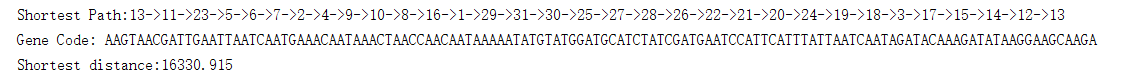
1. Results

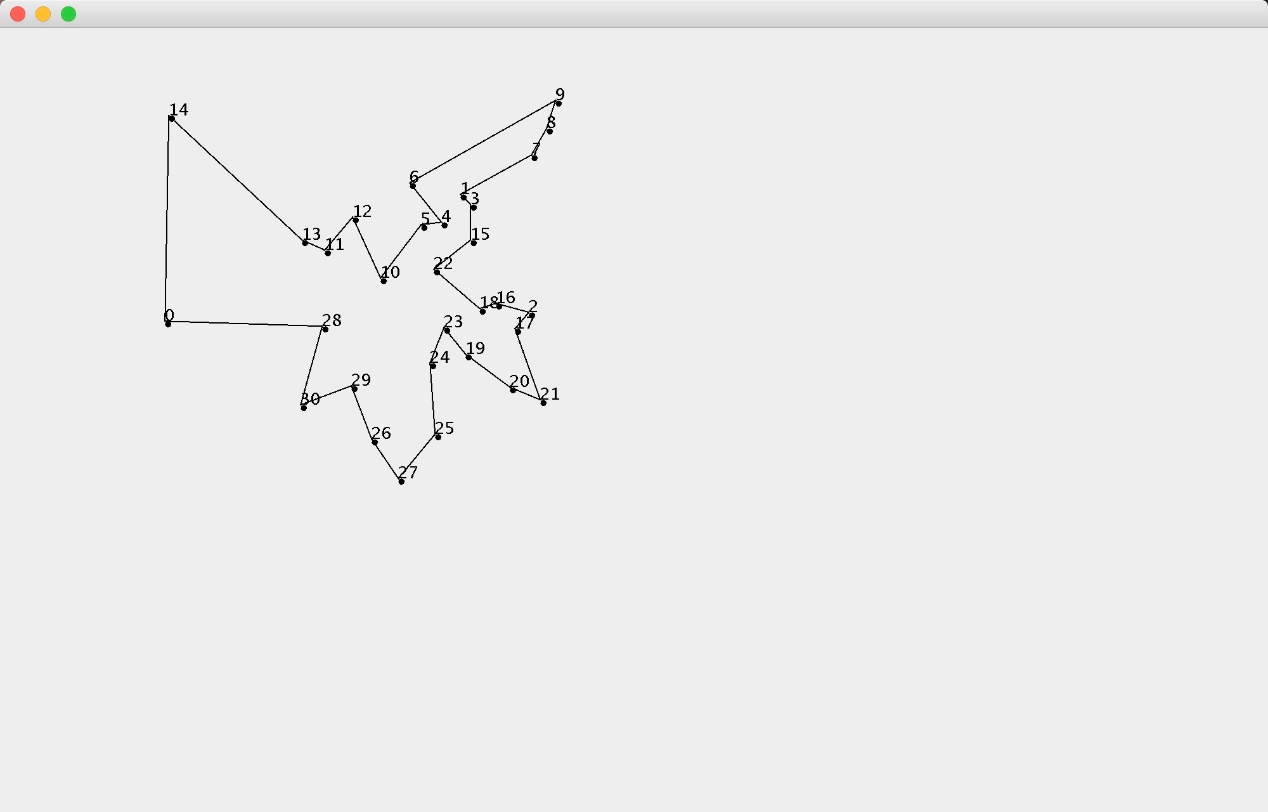
logging function



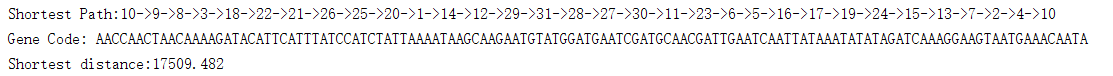
We have 31 cites in our project, and the correct TSP result is 14705.55.

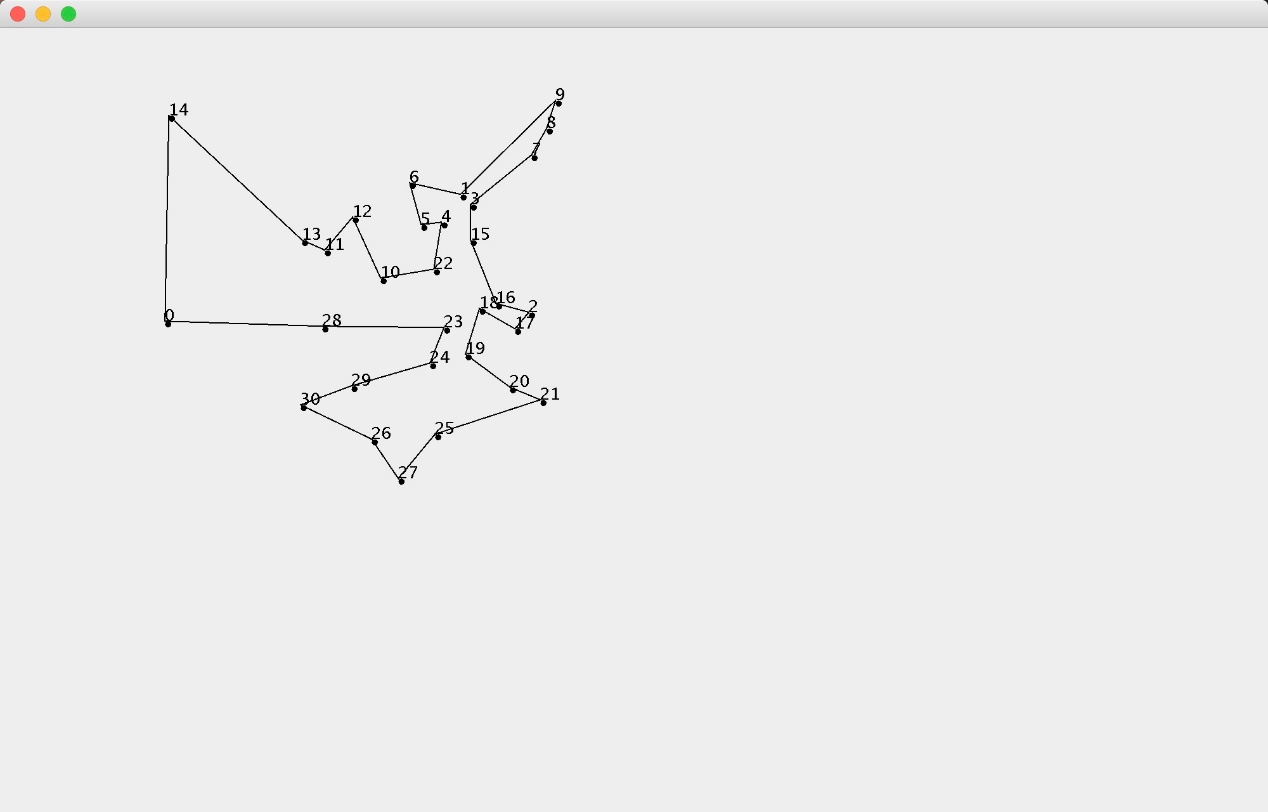
1. When initial population=200, generation=100, the factors of Roulette Wheel Selection are 0.2 & 0.95, the possibility of mutation is 0.4.



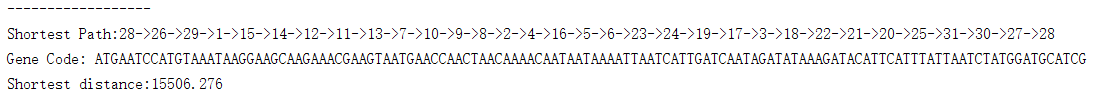


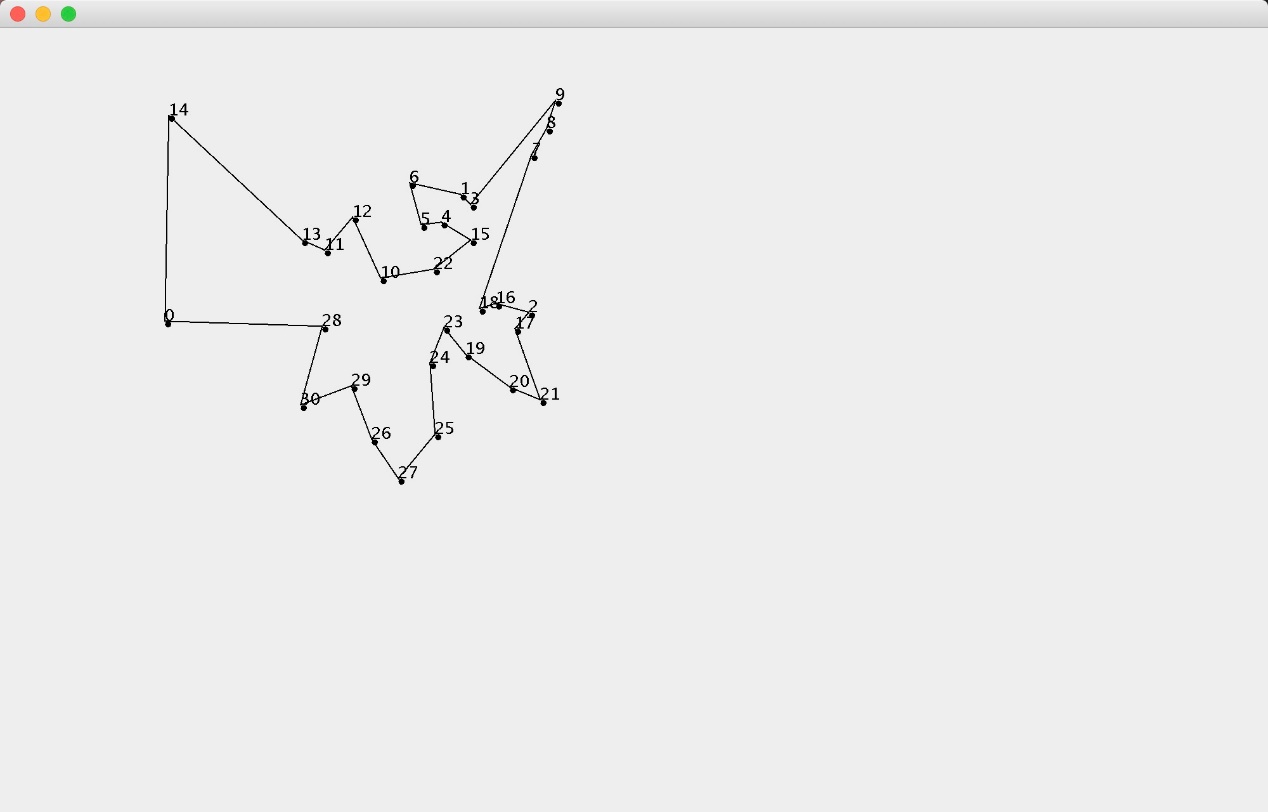
1. When initial population=400, generation=100, the factors of Roulette Wheel Selection are 0.2 & 0.95, the possibility of mutation is 0.4.



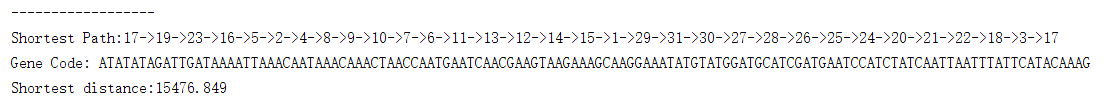


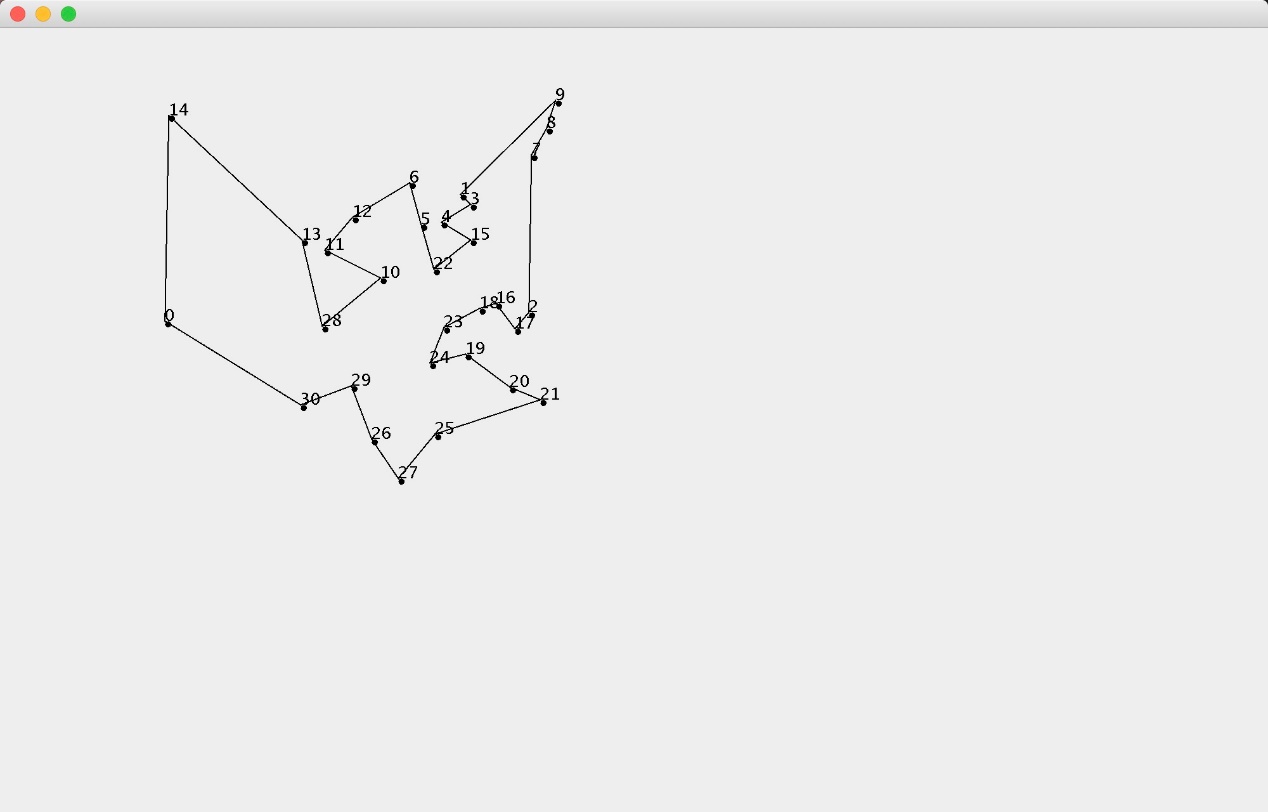
1. When initial population=1000, generation=100, the factors of Roulette Wheel Selection are 0.2 & 0.95, the possibility of mutation is 0.4.



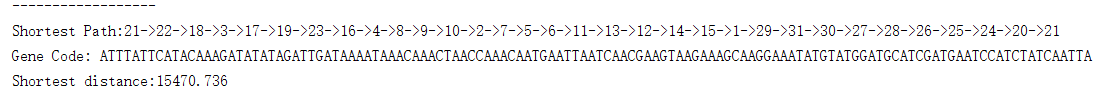


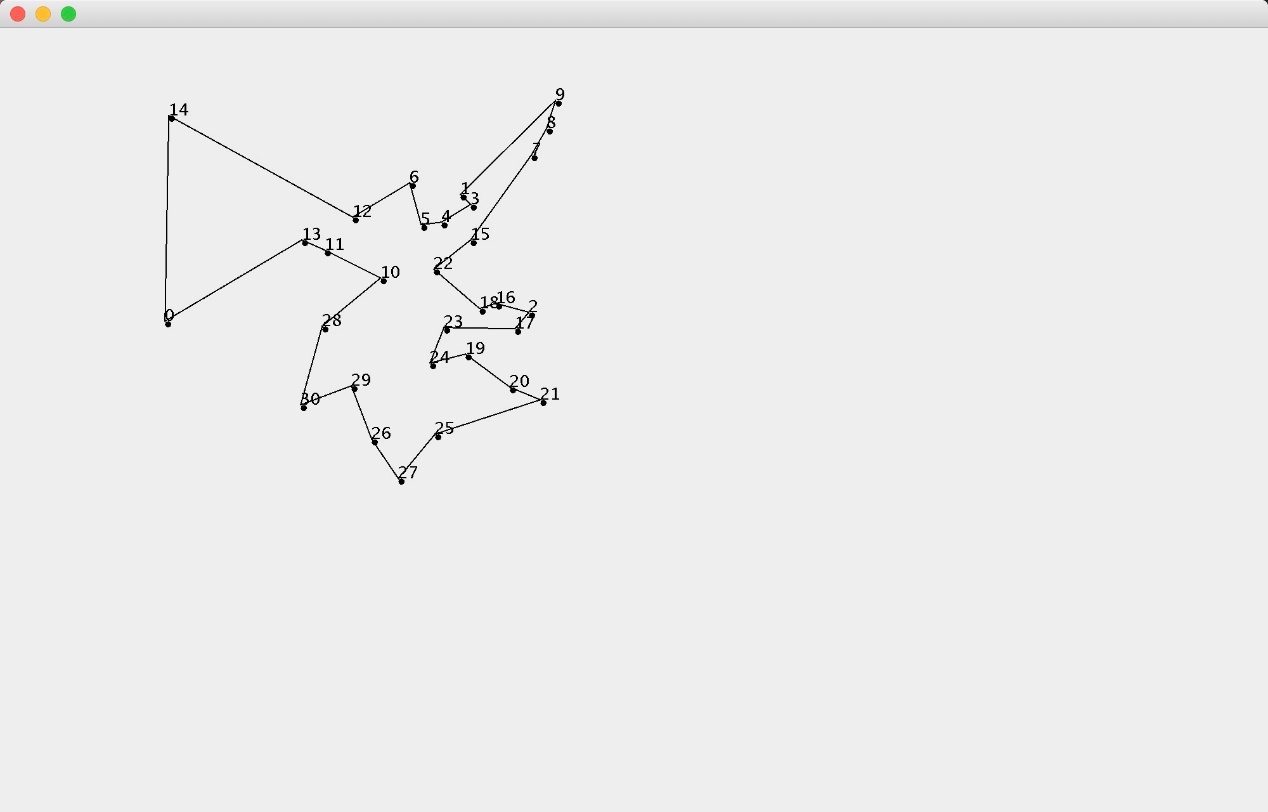
1. When initial population=2000, generation=100, the factors of Roulette Wheel Selection are 0.2 & 0.95, the possibility of mutation is 0.4.



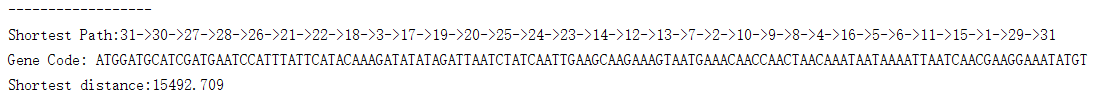


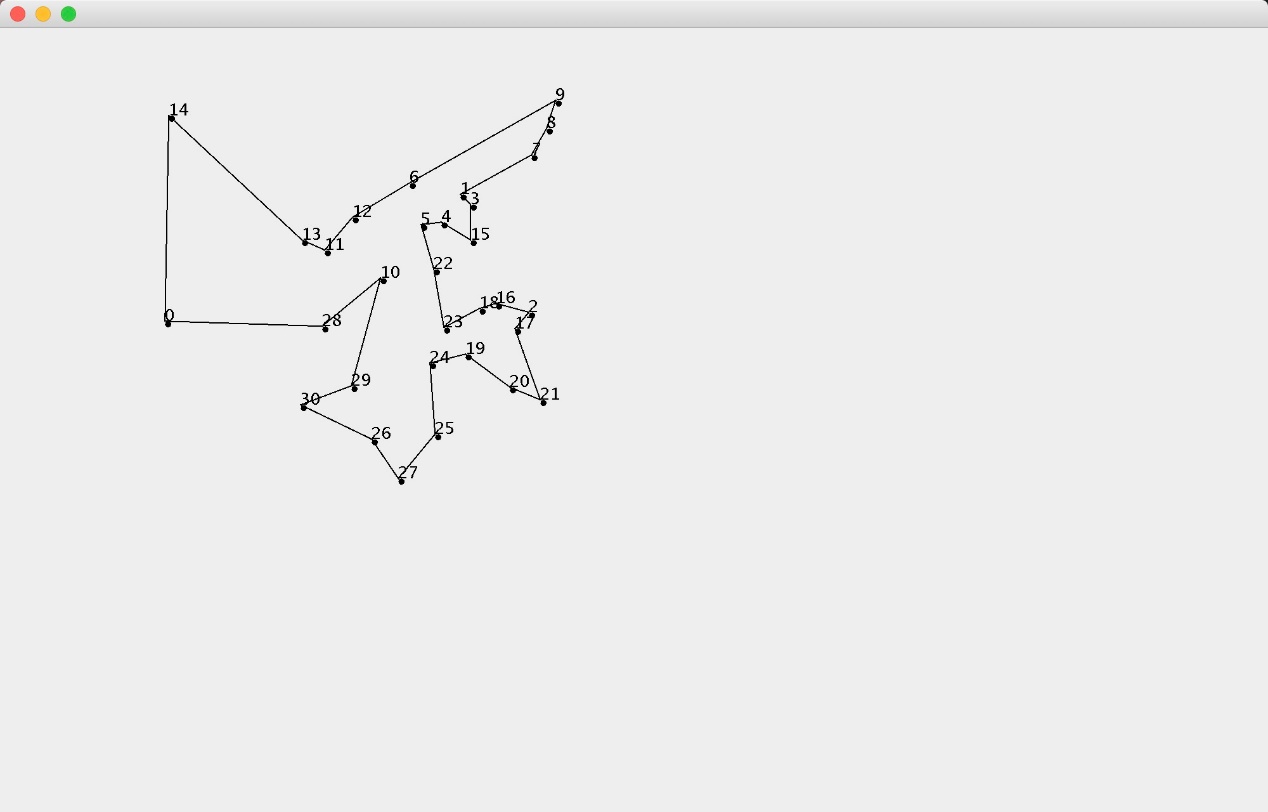
1. When initial population=3000, generation=100, the factors of Roulette Wheel Selection are 0.2 & 0.95, the possibility of mutation is 0.4.



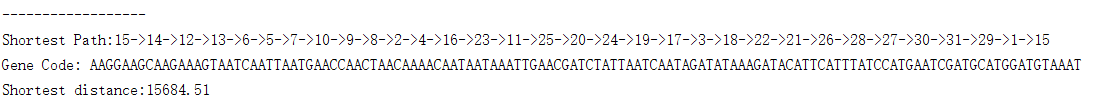


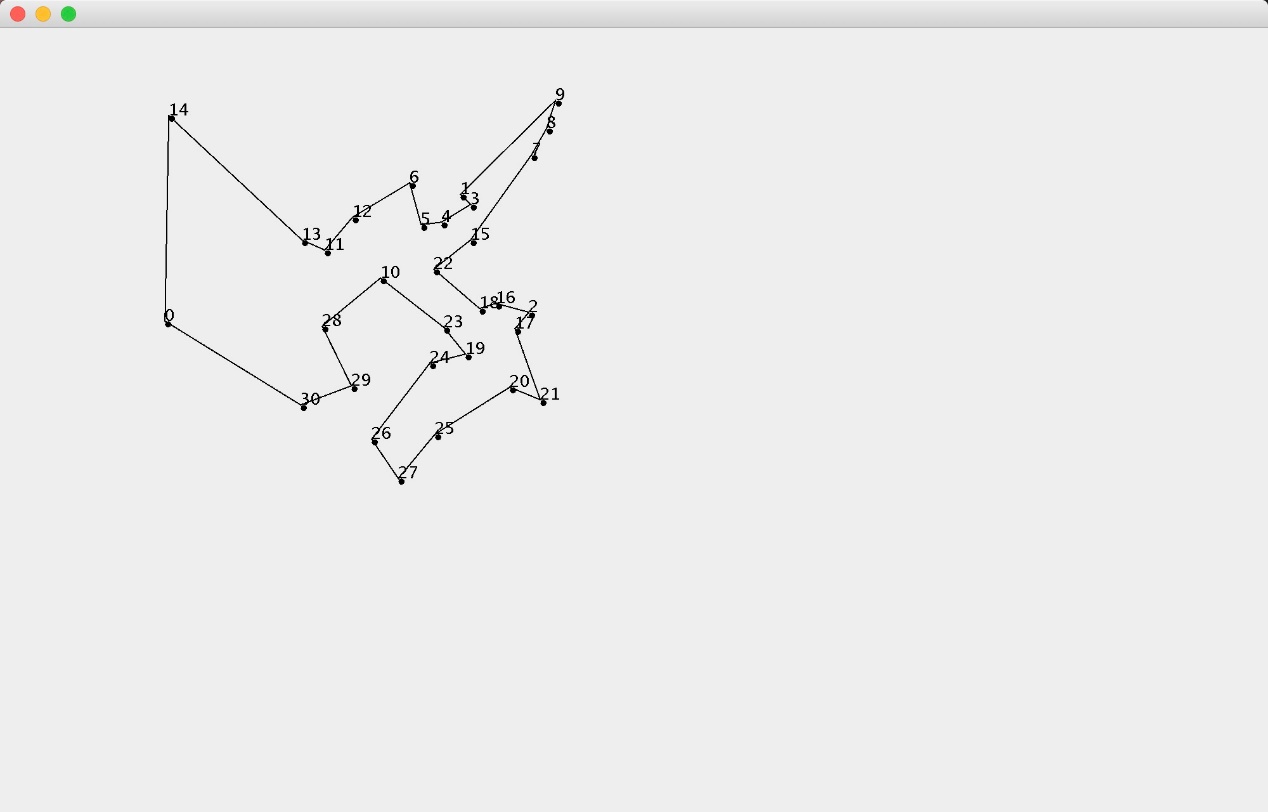
1. When initial population=1000, generation=100, the factors of Roulette Wheel Selection are 0.2 & 0.75, the possibility of mutation is 0.4.





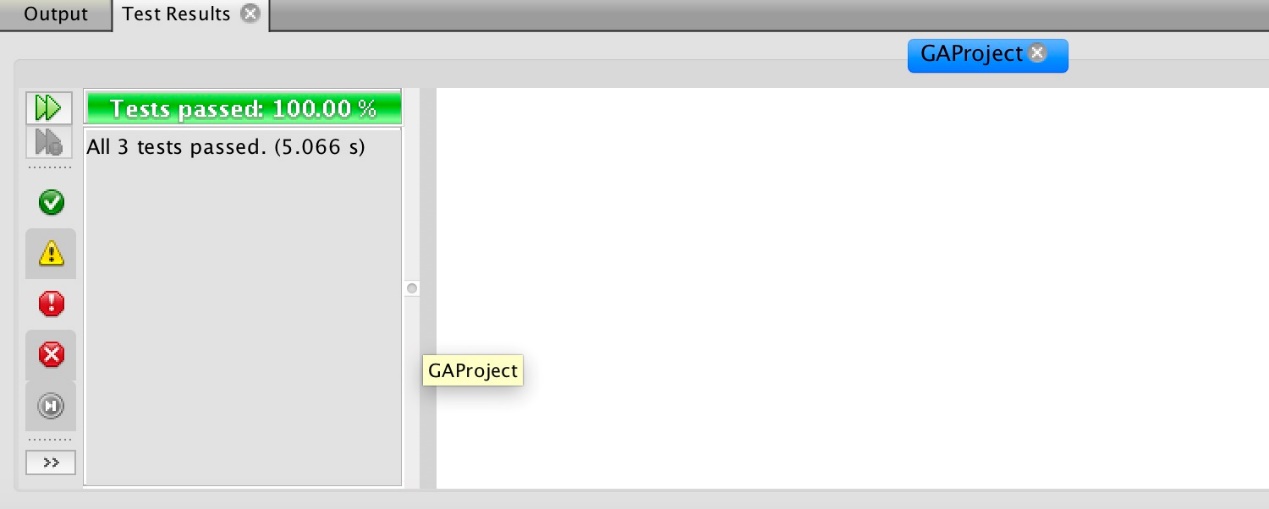
1. When initial population=1000, generation=100, the factors of Roulette Wheel Selection are 0.2 & 0.95, the possibility of mutation is 0.2.





1. Unit Test

It shows that all tests are passed.



1. Conclusion

During those tests, we can get the result which is nearly to the best route, so we can say that that genetic algorithm can solve the Travelling Salesman Problem. The result can be influence by the change of parameters such as population number, generation number, crossover possibility and mutation possibility. When the population number is larger, the result is more accurate. But the larger population means more running time, so we need to choose a suitable size of population.

There is also one thing we need to improve that TSP requires that salesman should start from a designated city and return to the same city, but our project just find the shortest route between N cities.

In conclusion, GA is very useful in our daily life, it is great to solve such TSP or Knapsack problem. We can only change the fitness function of GA to meets our different need.