GA METHOD SOLVING TRAVEL SALESMAN PROBLEM

Team member:
Ruofan Zhang 001470033
Xinyang Wu 001445433

Problem Description

- Suppose there is a traveler who wants to visit a city, ask him to start from a city, visit each city at most once, and finally return to the starting city to ensure that the selected path length is the shortest.
- Our goal is to use GA to find an optimal solution to this problem.

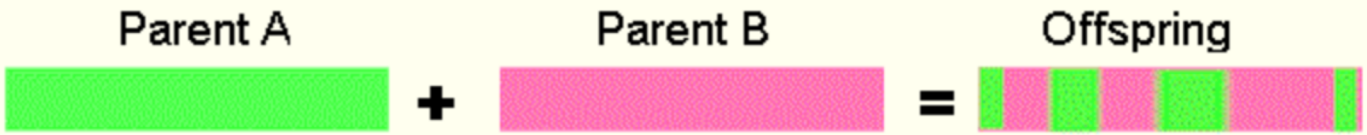
GA Implementation Description

- **Part 1. Class Description**
- To solve TSP problem, We implement Chromosome class, City class, Population class. These are Genetic Objects. We also implement Mutation class ,Crossover class, Selection class. And we use GeneticAlgorithm class to bring together the entire process of the Genetic Algorithm.

- **Part 2. Details of Class**
- **For GeneticObjects Package**
- **1.City class**
- It has methods to create cities with random names and random locations and find the Euclidean distance between two cities.
- **2.Chromosome class**
- It contains an array of City objects which represents a path through the cities. You can choose shuffle them or not.
- **3.Population class**
- It represents a Population of chromosomes.

- **For GeneticAlgorithms Package**
- **1. Crossover Class**
- It used for Chromosome reproduction. We create three methods to realize the crossover and implement the best of them(uniform order).
- 1) Uniform Order
- We use a bit mask to perform a uniform order crossover.

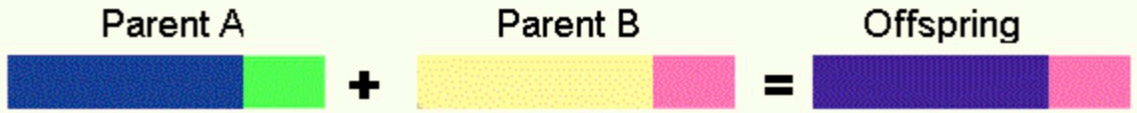
Uniform crossover - bits are randomly copied from the first or from the second parent



1**10**0101**1** + 1101**111**01 = 11011111

- 2) One Point Crossover
- It performs a crossover on all the cities after a particular point.

Single point crossover - one crossover point is selected, binary string from beginning of chromosome to the crossover point is copied from one parent, the rest is copied from the second parent



11001011+11011**111** = **11001111**

- 3) Order Crossover (Two Point Crossover)
- It performs a crossover on all the cities between two points.

Two point crossover - two crossover point are selected, binary string from beginning of chromosome to the first crossover point is copied from one parent, the part from the first to the second crossover point is copied from the second parent and the rest is copied from the first parent

11001**011** + 11**0111**11 = **11011111**

- **2. Mutation Class**
- It used for mutating the Chromosomes. We also created three different methods to realize the mutation and implemented one of them.
- 1) Insertion
- It selects a city and inserts it into a random place.
- This is a very effective mutation and is almost the same as Displacement Mutation, except here only one gene (i.e.; 2) is selected to be displaced and inserted back into the chromosome. In tests, this mutation operator has been shown to be consistently better than any of the alternatives mentioned here.
- 0 1 **2** 3 4 5 6 7
- Take the 2 out of the sequence,
- 0 1 3 4 5 6 7
- and reinsert the 2 at a randomly chosen position:
- 0 1 3 4 5 **2** 6 7

- 2) Reciprocal Exchange
- It swaps two randomly selected cities.
- Suppose we have a chromosome:
- 5 3 2 1 7 4 0 6
- We simply choose two genes at random (i.e.; 3 and 4) and swap them:
- 5 **4** 2 1 7 **3** 0 6

- 3) Scramble Mutation
- It picks a subset of Cities and randomly re-arrange them.
- Choose two random points (i.e.; positions 4 and 7) and "scramble" the genes located between them:
- 0 1 2 **3 4 5 6** 7
- becomes:
- 0 1 2 **5 6 3 4** 7

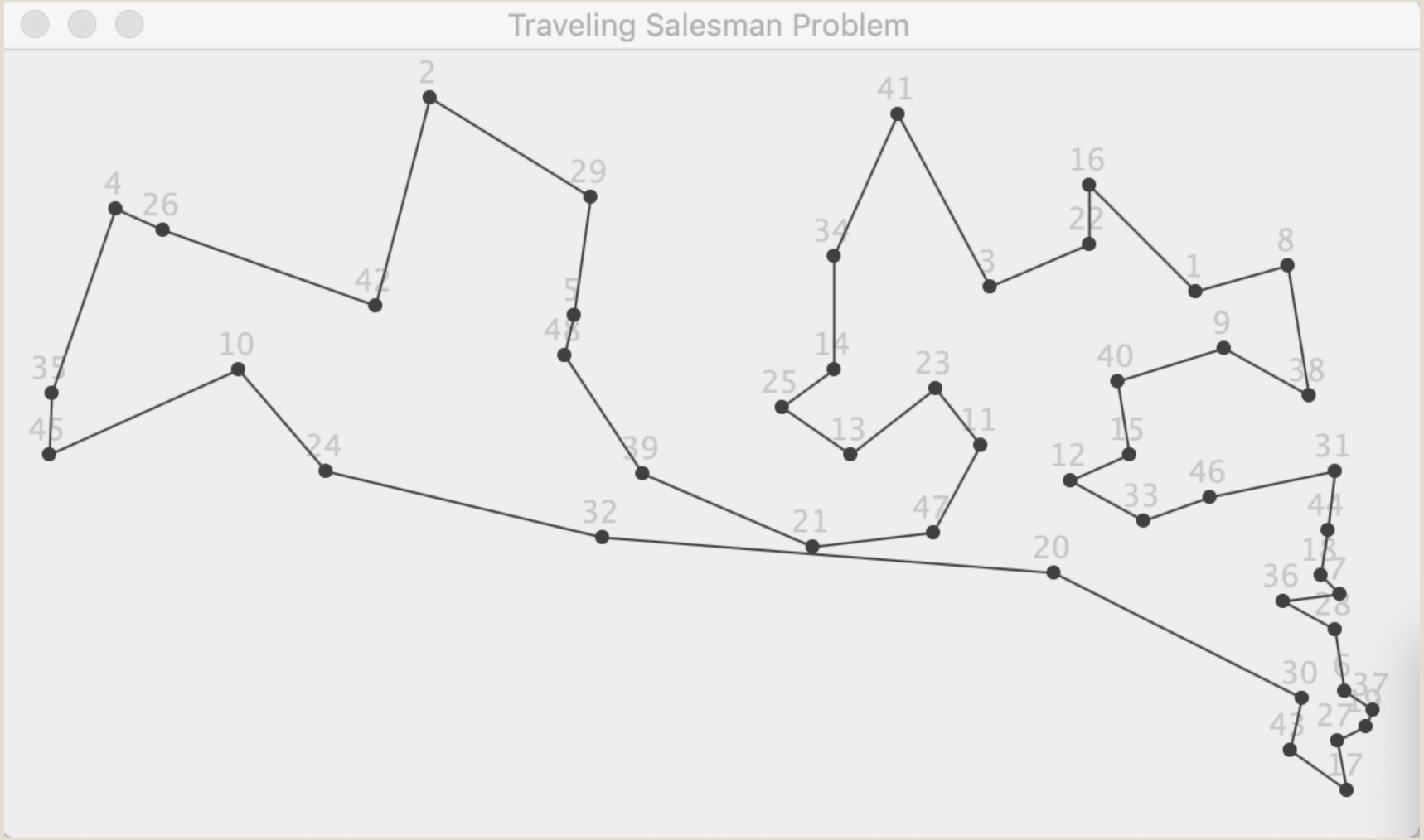
- **3. Selection Class**
- It determines which chromosomes will survive and potentially reproduce.
- We implement tournament selection in this part.
- 1)Select k random Individuals from the population.
- 2)Select the best Individual from the k Individuals.
- 3)Repeat from 1 until you have selected the desired amount of Individuals.

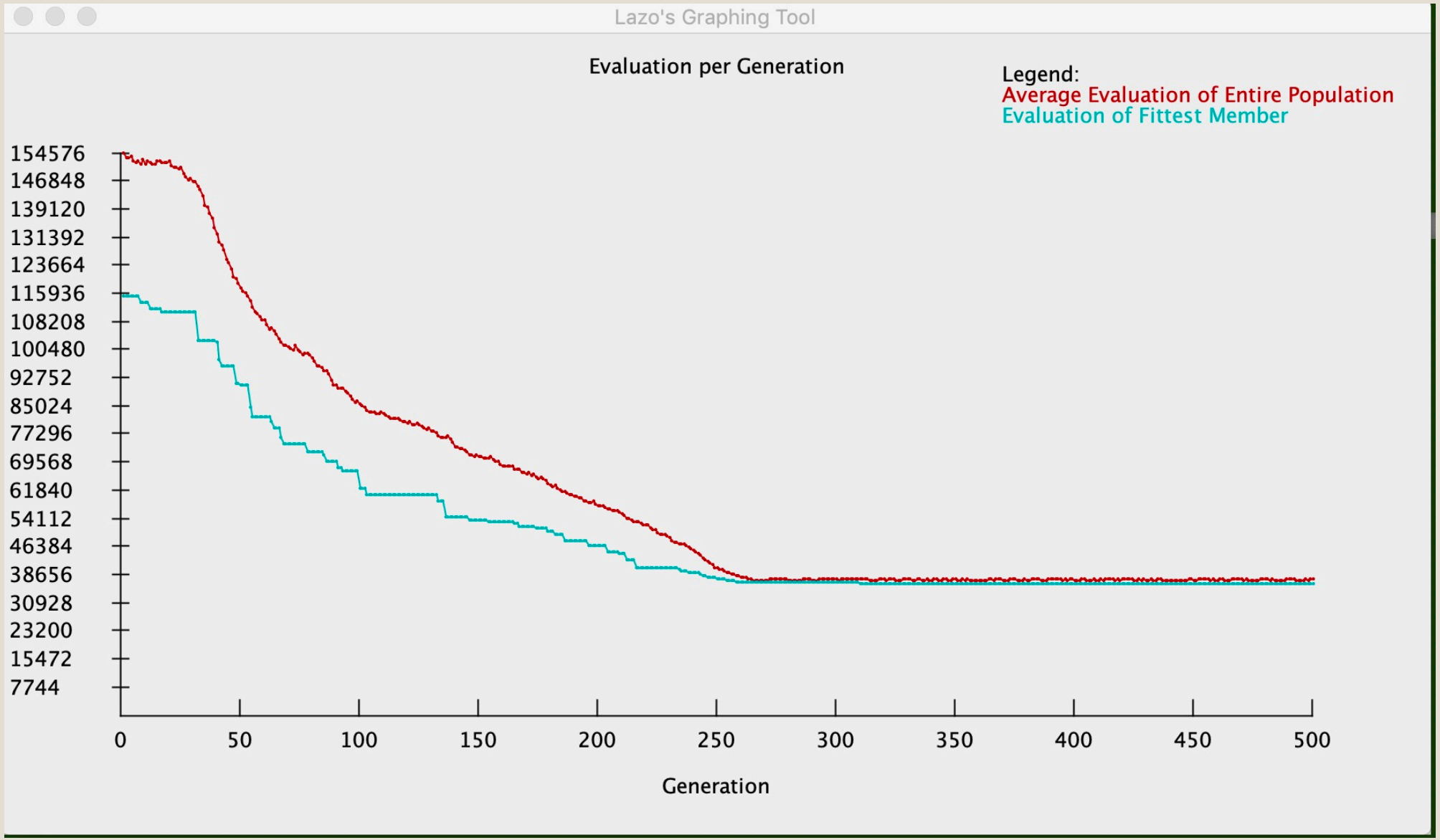
Result

- **No.1**
- **Preset**
- 1. We set the size of the population to 500, number of generations to run to 500, odds that crossover will occur to 0.90 and odds that mutation will occur to 0.04 in our Preset class.
- 2. We use the UNIFORM_ORDER method for crossover and INSERTION method for mutation.

- **Run**
- 1. We run the Main class.

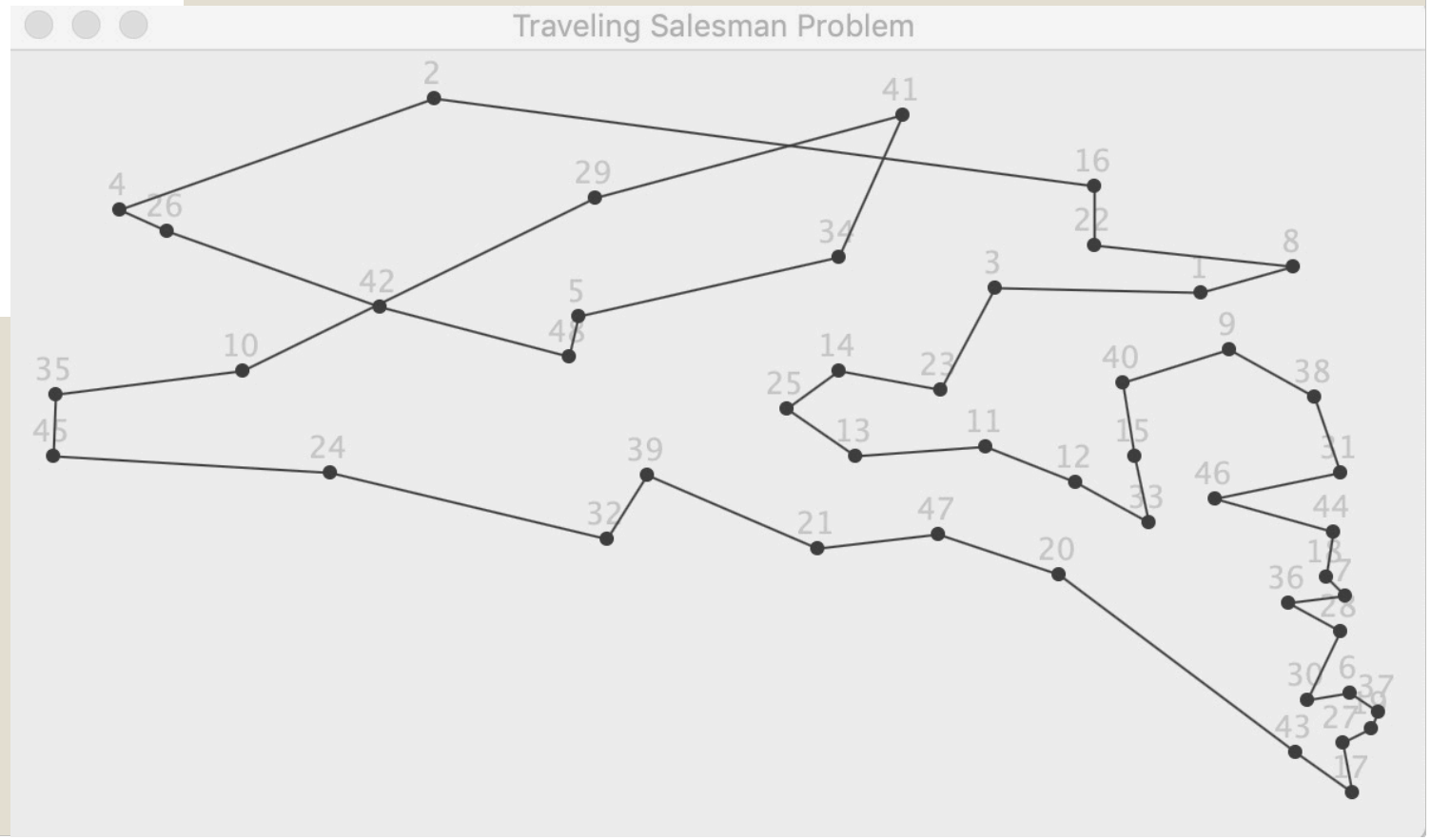
```
Seed: 4614414559665292519
----------Genetic Algorithm Properties----------
Number of Cities:   48
Population Size:    500
Max. Generation:    500
k Value:            3
Elitism Value:      1
Force Uniqueness:   false
Local Search Rate:  0.0
Crossover Type:     UNIFORM_ORDER
Crossover Rate:     90.0%
Mutation Type:      INSERTION
Mutation Rate:      4.0%
-----------Genetic Algorithm Results-----------
Average Distance of First Generation:  158130
Average Distance of Last Generation:   37820
Best Distance of First Generation:     126195
Best Distance of Last Generation:      36668
Area Under Average Distance:           31654059
Area Under Average Distance:           26129249
```

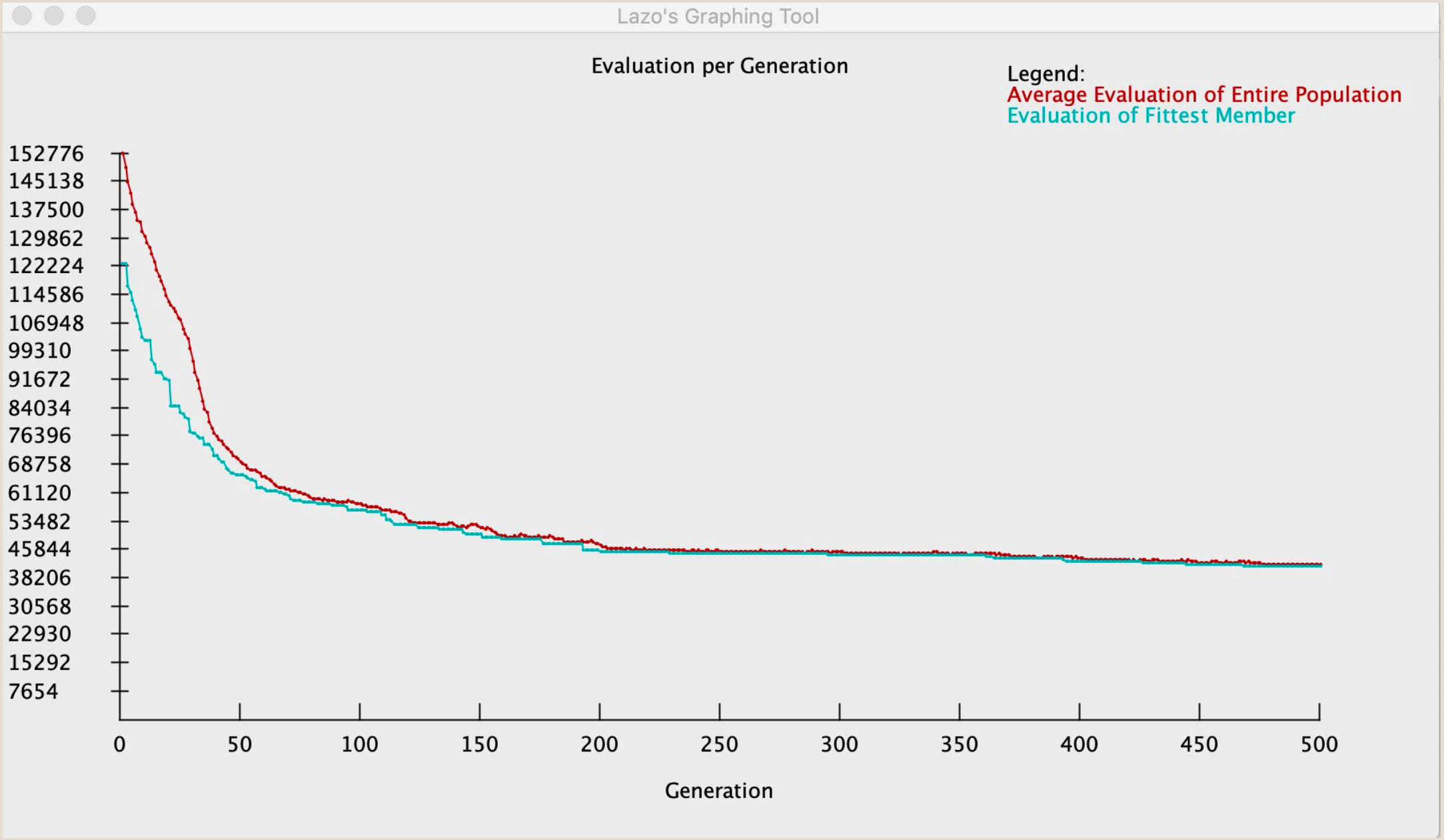

Traveling Salesman Problem
2
41
16
4
26
29
34
22
8
3
1
42
9
10
14
40
38
23
25
13
11
15
12
31
46
24
39
33
32
21
47
20
44
36
28
30
37
27
17


Lazo's Graphing Tool
Evaluation per Generation
Legend:
Average Evaluation of Entire Population
Evaluation of Fittest Member
154576
146848
139120
131392
123664
115936
108208
100480
92752
85024
77296
69568
61840
54112
46384
38656
30928
23200
15472
7744
0
50
100
150
200
250
300
350
400
450
500
Generation

- **No.2**
- **Preset**
- 1. We set the size of the population to 500, number of generations to run to 500, odds that crossover will occur to 0.90 and odds that mutation will occur to 0.04 in our Preset class. (no change with no.1)
- 2. We use the ONE_POINT method for crossover and RECIPROCAL_EXCHANGE method for mutation.(change the methods)
- **Run**
- 1. We run the Main class.

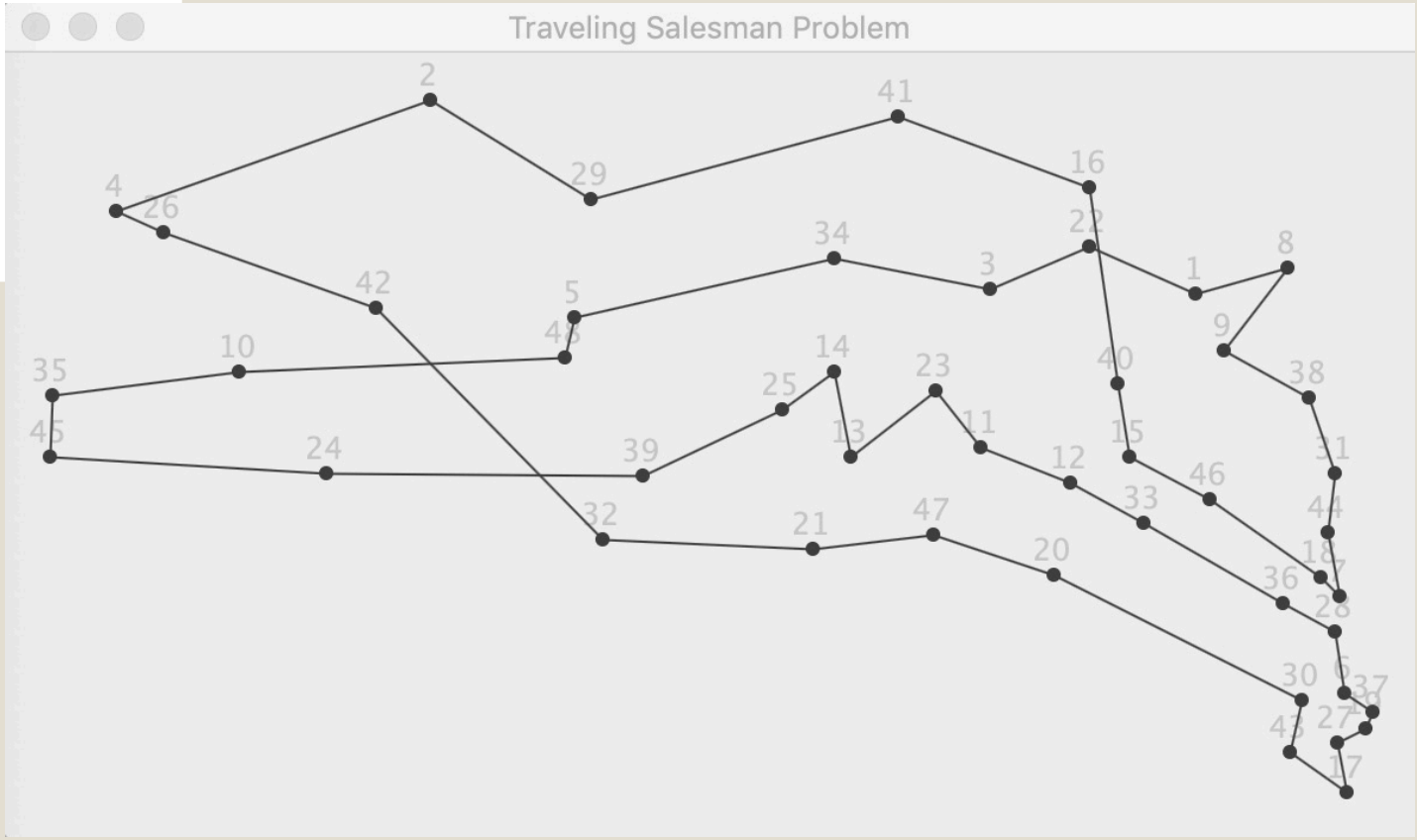
```
Seed: -3446331182324932219
----------Genetic Algorithm Properties----------
Number of Cities:   48
Population Size:    500
Max. Generation:    500
k Value:            3
Elitism Value:      1
Force Uniqueness:   false
Local Search Rate:  0.0
Crossover Type:     ONE_POINT
Crossover Rate:     90.0%
Mutation Type:      RECIPROCAL_EXCHANGE
Mutation Rate:      4.0%
-----------Genetic Algorithm Results-----------
Average Distance of First Generation:  158131
Average Distance of Last Generation:   42200
Best Distance of First Generation:     125795
Best Distance of Last Generation:      41547
Area Under Average Distance:           26859439
Area Under Average Distance:           25531906
```

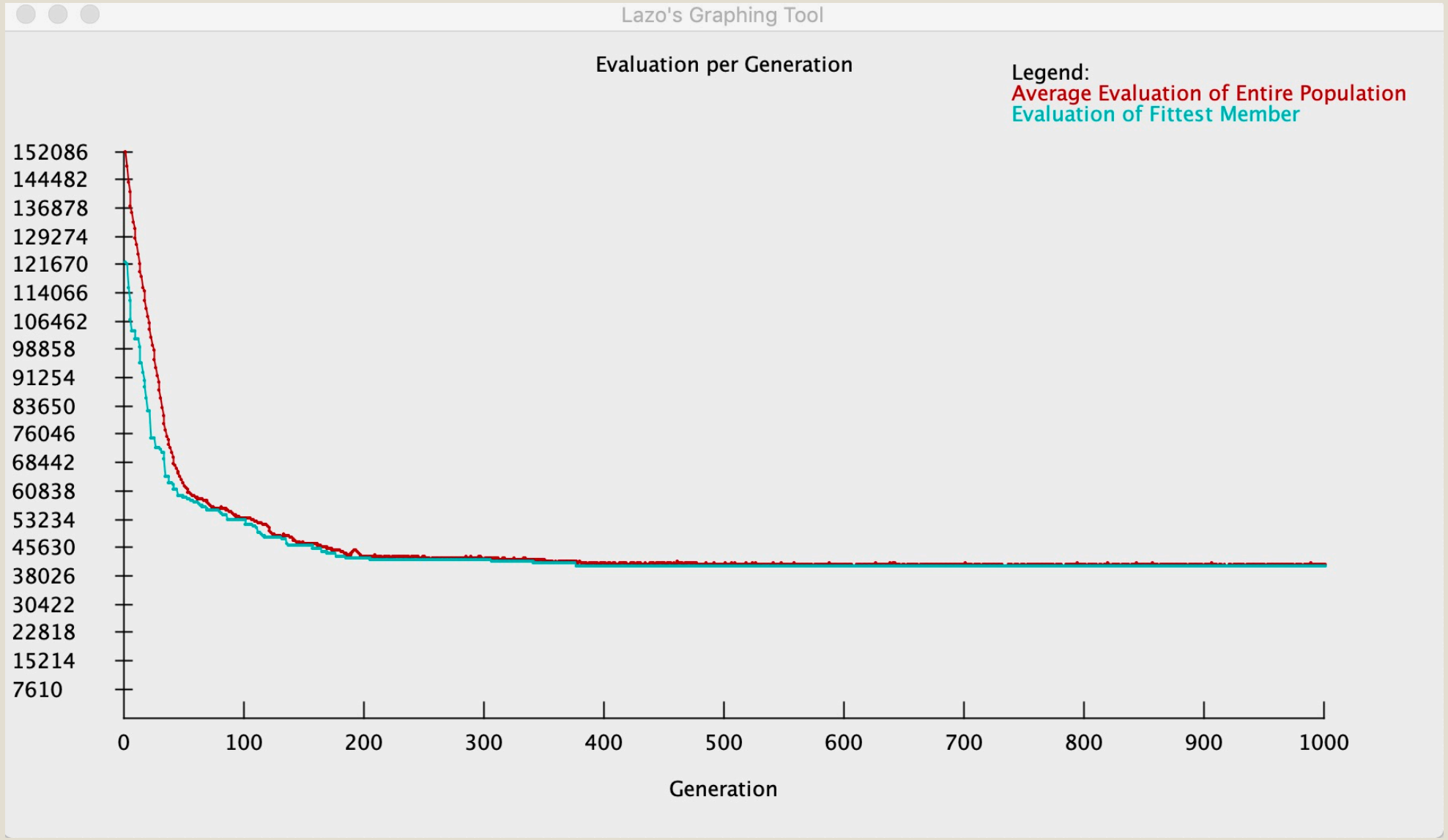



Lazo's Graphing Tool
Evaluation per Generation
Legend:
Average Evaluation of Entire Population
Evaluation of Fittest Member
152776
145138
137500
129862
122224
114586
106948
99310
91672
84034
76396
68758
61120
53482
45844
38206
30568
22930
15292
7654
0
50
100
150
200
250
300
350
400
450
500
Generation

- **No.3**
- **Preset**
- 1. We set the size of the population to 500, number of generations to run to 1000, odds that crossover will occur to 0.90 and odds that mutation will occur to 0.04 in our Preset class. (change the preset data)
- 2. We use the ONE_POINT method for crossover and RECIPROCAL_EXCHANGE method for mutation.(no change with no.2)
- **Run**
- 1. We run the Main class.

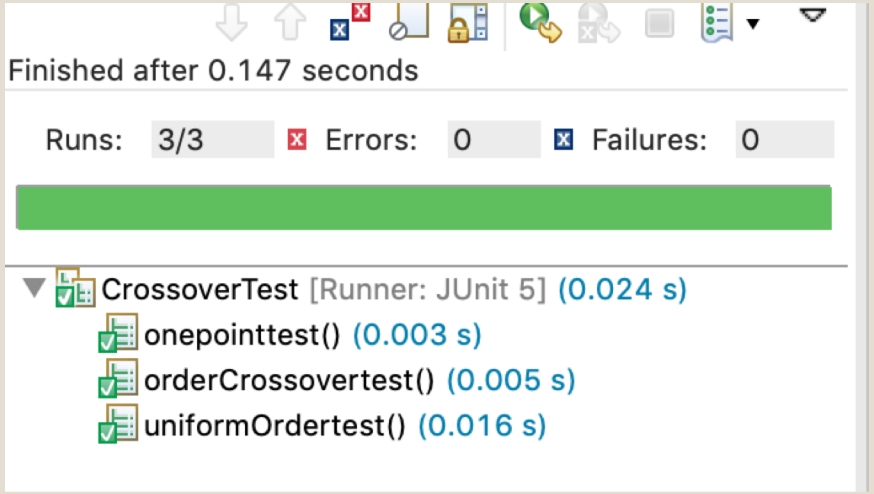
```
Seed: -8372246724595848914
----------Genetic Algorithm Properties----------
Number of Cities:   48
Population Size:    500
Max. Generation:    1000
k Value:            3
Elitism Value:      1
Force Uniqueness:   false
Local Search Rate:  0.0
Crossover Type:     ONE_POINT
Crossover Rate:     90.0%
Mutation Type:      RECIPROCAL_EXCHANGE
Mutation Rate:      4.0%
-----------Genetic Algorithm Results-----------
Average Distance of First Generation:  157088
Average Distance of Last Generation:   41383
Best Distance of First Generation:     123605
Best Distance of Last Generation:      40838
Area Under Average Distance:           46013647
Area Under Average Distance:           44478834
```



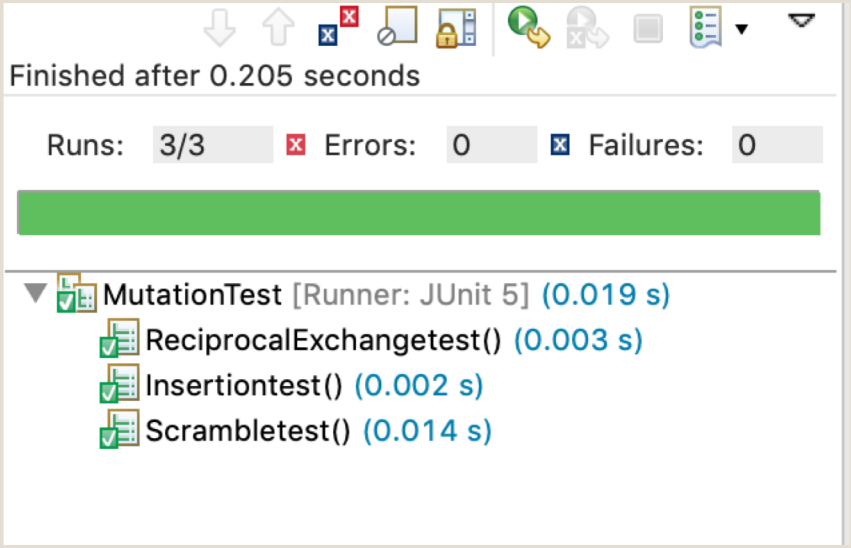

Lazo's Graphing Tool
Evaluation per Generation
Legend:
Average Evaluation of Entire Population
Evaluation of Fittest Member
152086
144482
136878
129274
121670
114066
106462
98858
91254
83650
76046
68442
60838
53234
45630
38026
30422
22818
15214
7610
0
100
200
300
400
500
600
700
800
900
1000
Generation

Test

Crossover test

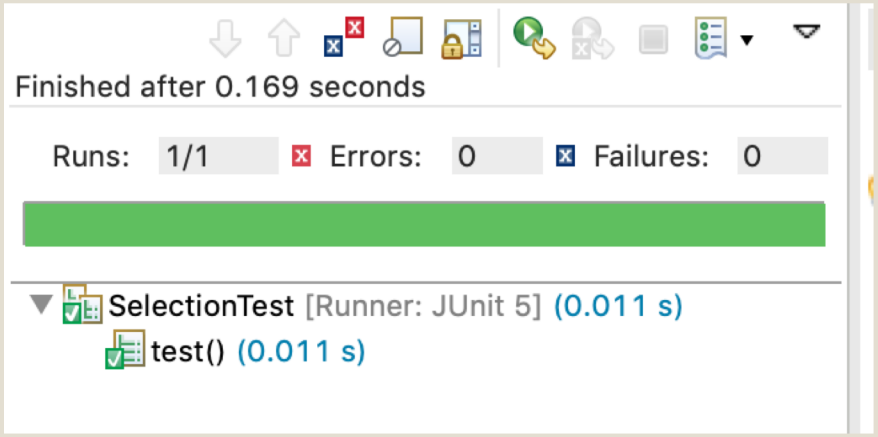


Mutation test



Test

Selection test



Conclusion

- Genetic algorithms are evolutionary techniques for survival purposes based on the most appropriate ideas for survival. These methods do not guarantee the best solution; however, they usually give good approximations in a timely manner. Genetic algorithms are useful for NP-hard problems, especially for traveling salesmen. Genetic algorithms depend on selection criteria, crossover and mutation operators.
- According to the specified number of iterations, when the algorithm reaches the number of iterations, the algorithm ends and outputs the current optimal solution. When calculating and selecting from the adaptive values, the current best value of the record is added to the updated group after ensuring that the TSP solution is getting better (no worse) in the new iteration loop.
- We tried different methods, but in reality they have little effect on the results.
- Through testing, we can get the best path no matter what method we use, although sometimes the results are almost optimal.