INFO6205_213

Akash Anand: 001873134
Karan Saini: 001828012

PROBLEM

- Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.

Genotype and phenotype

- Genotype is the digital information that is passed down generation to generation
- Phenotype is an expression of data.
- For this TSP problem we have considered the order of array index as genotype, which intern are stored in an array.

Fitness Function

- A gene's fitness is calculated by inverting the sum of distance between all the vertices(cities) within a gene, which also includes distance between last and the first city (Hamiltonian cycle)
- The smaller the distance the higher fitness value and vice versa

- Fitness of a gene = $\frac{1}{\text{distance between cities}}$

Normalized fitness

- Each gene contains a normalized fitness value (between 0-1), which determines its probability of getting chosen for crossover over other genes within a population.
- It is calculated by dividing a gene's fitness value with the sum of all genes fitness value.
- Let's say A, B and C are genes then,

$$\text{Normalized fitness value of A} = \frac{\text{fitness of A}}{fitness\ of\ A + fitness\ of\ B + fitness\ of\ C}$$

Crossover

- For crossover, it is not always best to choose the fittest among the population, every gene should be considered for better variation in the population
- But the genes with high fitness value should have high chances of being selected and the gene with low fitness value should have less chances. Therefore, normalized fitness values are assigned to each gene.
- Two genes (parent1, parent2) are chosen according to their "normalized fitness value" and following steps are performed:

Two random indexes are selected, start and end

Let start = 2, end = 4, then the values are copied to child

Parent1:

1	0	2	4	3
		Start		**End**

Child:

2	4	3		

- The remaining values are copied from Parent2

Parent2:

1	4	3	0	2

Child:

2	4	3	1	0

Mutation:

- After crossover mutation on the child takes place
- A “mutation rate” is provided which determines the probability whether the mutation process should occur or not
- Mutation is done by swapping the values at two indices selected randomly
- Let random index be 0 and 3 then:

Child:

2	4	3	1	0
0	1	2	3	4

Swap values 0 and 3

Child:

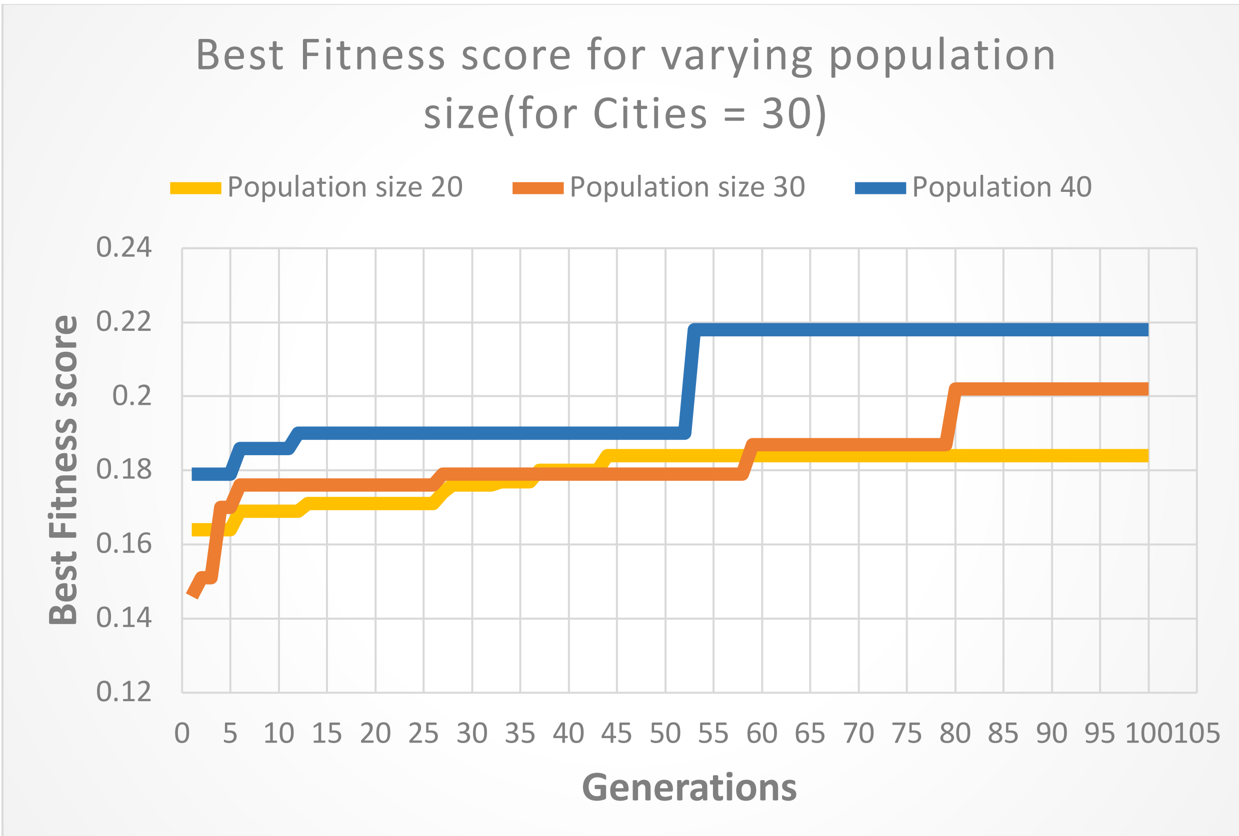
1	4	3	2	0

Evolution:

- For the evolution of next generation, individuals, selected as per normalized fitness value, sexually reproduces, and the child may or may not undergo mutation, depending on mutation rate
- Only 2-4 fittest gene (Elite genes) of the present generation are passed to next generation, all the other weak genes are removed. The process is repeated for "n" number of generations.

Result1:

Number of cities (n)	30	30	30
Population size	20	30	40
Generation (m)	100	100	100
Survivors	326	330	331
Fitness score (at 1 - m generation)	0.164 - 0.184	0.146 - 0.202	0.170 – 0.218



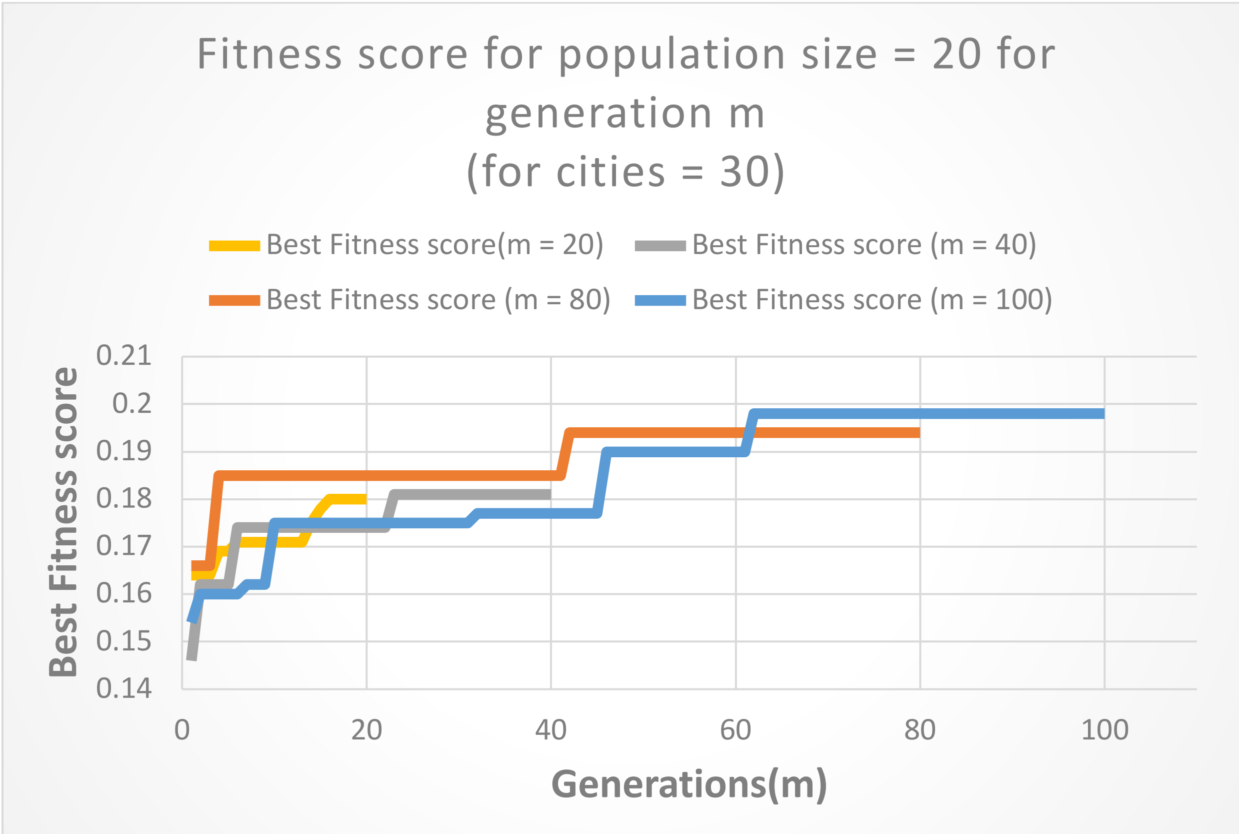
<u>Observations</u>:

- From Graph 1 it can be observed that, with increased in initial population size (i.e. population size at generation 0) and with constant generations (m = 100) there is a high chance of possibility to get a optimum solution
- With increase in initial population, variation in population exits, hence improving the results

Result2:

Constant number of cities(n =30) and initial population size(= 20), varying number of generations(m)

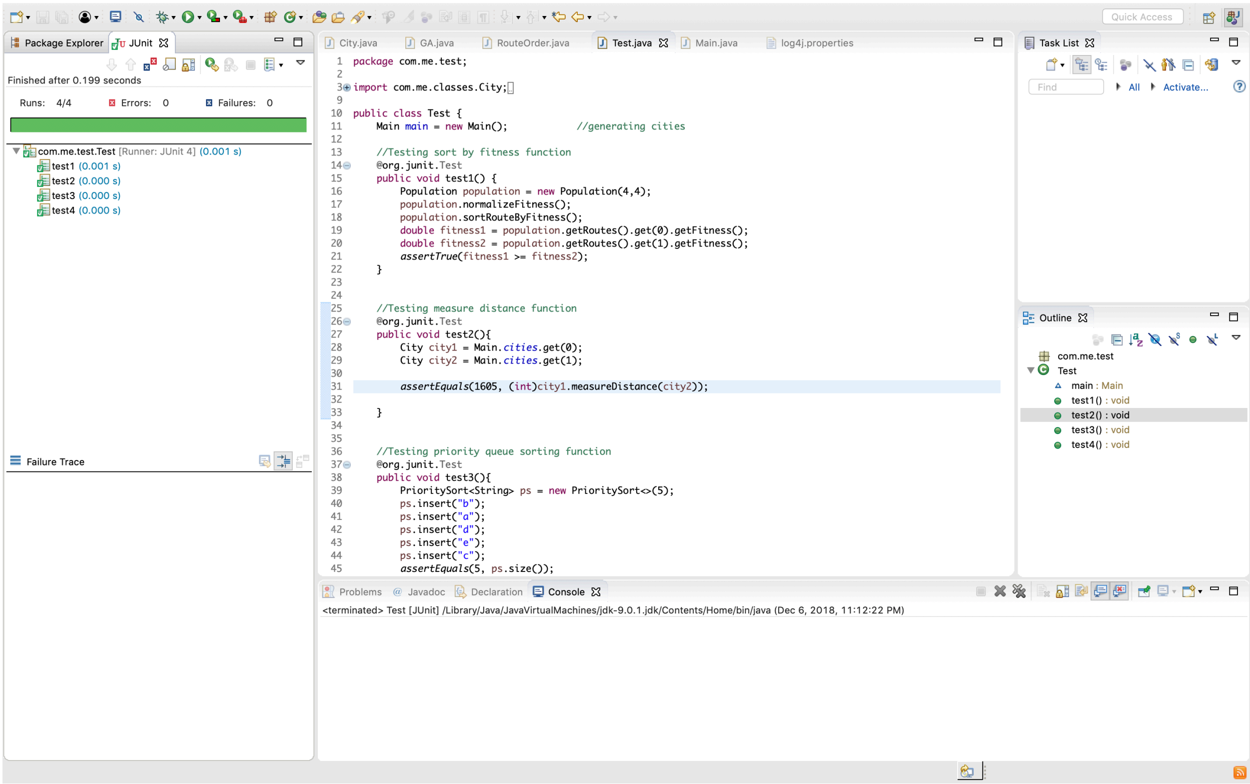
Number of cities (n)	30	30	30	30
Population size	20	20	20	20
Generation (m)	20	40	80	100
Survivors	81	142	251	328
Fitness score (at 1 - m generation)	0.164 - 0.180	0.140 - 0.181	0.166 – 0.194	0.154- 0.198



<u>Observation:</u>

- From Graph 2 and Result 2, it can be observed that with increase in generation the number of survivors increases, due to which the variation in the population increases in every new generation
- Increase in variation of population, improves fitness value

Test Cases:


Package Explorer
JUnit
Finished after 0.199 seconds
Runs: 4/4
Errors: 0
Failures: 0
com.me.test.Test [Runner: JUnit 4] (0.001 s)
test1 (0.001 s)
test2 (0.000 s)
test3 (0.000 s)
test4 (0.000 s)
Failure Trace
City.java
GA.java
RouteOrder.java
Test.java
Main.java
log4j.properties
package com.me.test;
import com.me.classes.City;
public class Test {
Main main = new Main(); //generating cities
//Testing sort by fitness function
@org.junit.Test
public void test1() {
Population population = new Population(4,4);
population.normalizeFitness();
population.sortRouteByFitness();
double fitness1 = population.getRoutes().get(0).getFitness();
double fitness2 = population.getRoutes().get(1).getFitness();
assertTrue(fitness1 >= fitness2);
}
//Testing measure distance function
@org.junit.Test
public void test2(){
City city1 = Main.cities.get(0);
City city2 = Main.cities.get(1);
assertEquals(1605, (int)city1.measureDistance(city2));
}
//Testing priority queue sorting function
@org.junit.Test
public void test3(){
PrioritySort<String> ps = new PrioritySort<>(5);
ps.insert("b");
ps.insert("a");
ps.insert("d");
ps.insert("e");
ps.insert("c");
assertEquals(5, ps.size());
Problems
Javadoc
Declaration
Console
<terminated> Test [JUnit] /Library/Java/JavaVirtualMachines/jdk-9.0.1.jdk/Contents/Home/bin/java (Dec 6, 2018, 11:12:22 PM)
Task List
Find
All
Activate...
Outline
com.me.test
Test
main : Main
test1() : void
test2() : void
test3() : void
test4() : void
Quick Access