# **README**

**Student ID : R00182505**

**Genetic Algorithm for Traveling Salesman Problem**

**Description**: The project works on implementing the genetic algorithm on Travelling salesman problem along with use of various selection, crossover and mutation techniques.

**Compiling Environment:**

Running code requires Python 3 to be installed on the machine. Install python if it’s not already present

Follow the instructions to install Python

<https://realpython.com/installing-python/>

**Directory Structure:**

├── TSP\_R00182505.py

├── Individual.py

├── inst-0.tsp

├── inst-5.tsp

├── inst-13.tsp

├── Test Runs

│ ├── run\_experimental\_test\_population\_size.txt

│ ├── run\_experimental\_test\_Mutation\_Rate.txt

│ ├── run\_experimental\_test\_population\_size.txt

│ └── run\_performance\_test\_config\_3.txt

│ └── run\_performance\_test\_config\_6.txt

│ └── run\_tsp0\_1.txt

│ └── run\_tsp5\_1.txt

│ └── run\_tsp13\_1.txt

│ └── Heuristic\_experiment.txt

**Parameter List:**

The call to the Genetic Algorithm is made by passing the below set of parameters. The Parameters are mentioned in order they appear in the call

Parameter List:

1. Filename

2. Population Size

3. Mutation Rate

4. Maximum Iterations

5. Initial Solution = {0: Random, 1: Heuristic}

6. Selection = {0: Random, 1: Stochastic}

7. Crossover type = {0: Uniform Crossover, 1: PMX Crossover}

8. Mutation Type = {0: Inversion Exchange, 1: Reciprocal Exchange}

**Modifications in TSP\_R00182505.py:**

In order to run any set of configuration, the parameters mentioned above related to that configuration need to be passed to the BasicTSP().

ga = BasicTSP*(<problem file instance>, <population size> ,<mutation rate>, <no. of iterations>,< Initial Solution = {0: Random, 1: Heuristic}>, < Selection = {0: Random, 1: Stochastic}>, < Crossover type = {0: Uniform Crossover, 1: PMX Crossover}>, < Mutation Type = {0: Inversion Exchange, 1: Reciprocal Exchange}>)*

Example:

To run configuration 1, do the following changes in the script

ga = BasicTSP(problem\_file, 100, 0.1, 500, 0, 0, 0, 0)  
ga.search()

To run configuration 7, do the following changes in the script

ga = BasicTSP(problem\_file, 100, 0.1, 500, 1, 1, 1, 1)  
ga.search()

**How to run:**

On the command line/Shell run the below command:

> python TSP\_R00182505.py [instance]

**Example**: >python TSP\_R00182505.py inst-0.tsp

**Source File description:**

1. **Individual.py**:
   1. class Individual

The class contains an \_\_init\_\_ method that generates the chromosome which will be the part of the population in the form of object of class Individual

* 1. def copy()

The method is used to create the copy of the Individual class object over which the method is called

* 1. def euclideanDistance()

The method is used to create the distance between two cities by Euclidean formula using the x and y vertices of the node

* 1. def computeFitness()

The method is used to compute the fitness i.e. the total path length of the chromosome in population

* 1. def getFitness()

Return the fitness of the individual object

1. **TSP\_R00182505.py**:

The python script is responsible for performing the genetic algorithm over the TSP. The scrip has functionality related to initializing population, selection, parent selection, crossover, mutation, new population generation

* 1. def readInstance()

The method is used to read the .tsp file to generate dictionary with key as city and the value as (x ,y) coordinates

* 1. def initPopulation()

The method is responsible for initializing the population with either random or heuristic approach

* 1. def randomSelection()

Returns parents required for mating process by selecting them from the mating pool

* 1. def stochasticUniversalSampling()

Returns two selected parents from the mating pool generated using selection probability

* 1. def uniformCrossover()

Returns 2 offsprings after perfroming uniform Crossover on the Parent A and B

* 1. def pmxCrossover()

Returns 2 offsprings after performing PMX Crossover on the Parent A and B

* 1. def pmx\_mapper()

The method is used by pmxCrossover() in order to update individual after the mapping cycle procedure

* 1. def reciprocalExchangeMutation()

Mutate an individual by swapping two cities with certain probability (i.e., mutation rate)

* 1. def reciprocal\_mutation\_flip()

This method is used to swap the values in the Reciprocal Mutation process for the Child

* 1. def inversionMutation()

Mutate the individual by inversing the order of cities between two points

* 1. def updateMatingPool()

Updates the mating pool before creating a new generation

* 1. def newGeneration()

Creates new generation by calling methods for selection, crossover, mutation

* 1. def search()

Iterates to run the genetic algorithm for the specified number of times

**Test Run:**

The genetic algorithm was tested with all the 8 configurations and the results were stored in text files. The directory ‘Test Runs’ contains the details regarding that.

Environment: Python 3.6.8

├── Test Runs

│ ├── run\_experimental\_test\_heuristic.txt –> test with heuristic approach

│ ├── run\_experimental\_test\_Mutation\_Rate.txt -> Varying Mutation Rate

│ ├── run\_experimental\_test\_population\_size.txt -> Varying population size

│ └── run\_performance\_test\_config\_3.txt -> Performance test on config 3

│ └── run\_performance\_test\_config\_6.txt -> Performance test on config 6

│ └── run\_tsp0\_1.txt -> Test result for inst-0.tsp

│ └── run\_tsp5\_1.txt -> Test result for inst-5.tsp

│ └── run\_tsp13\_1.txt -> Test result for inst-13.tsp

│ └── Heuristic\_experiment.txt -> test with varying pop. for heuristic approach