Below detail the changes to the provided code and how each method is implemented in a brief way. To directly jump to know how to run the code please [**click here**](#How_to_run_the_code)

Details of my implementation of the methods initialization, crossover, mutation methods and changes done to the **Individual.py** and **TSP\_R00195877.py** files.

**Methods of file TSP\_R00195877.py:**

1. **\_\_init\_\_**: Apart from the default parameters, I am passing 3 new parameters:
   1. **\_ini\_genr:** Determines how the initial set of population is generated (could be “random” for random generation of tours or “NN” for Nearest Neighbor insertion heuristics)
   2. **\_crs\_over:** Determines which cross over method to be called. (Could be either “order1” for order-1 Crossover and “uniform” for Uniform order based Crossover)
   3. **\_mut\_meth:** Determines which mutation method to be called. (Could be either “inverse” for Inverse mutation or “scramble” for Scramble mutation)
2. **readInstance(self):** There is no change to this method, used as it is, which will read the file passed and update genSize and data variables accordingly
3. **initPopulation(self):** This will create the individuals as many as specified in \_popSize parameter depending on the \_ini\_genr value (either “random” or “NN”)
4. **updateBest(self):** Used as it is without any change. This will create a copy of fittest individual in the population
5. **binaryTournamentSelection(self):** A typical implementation of binary tournament selection procedure, where 4 individuals are selected at random from mating pool, create 2 pairs at random and return the fittest parent from each pair
6. **uniformCrossOver(self):** Implements the uniform Crossover method. This method creates and returns only one child
7. **order1Crossover(self):** Implements the order1Crossover method. This method creates and returns only one child
8. **scrambleMutation(self):** Implements the scramble Mutation
9. **inverseMutation(self):** Implements the inverse Mutation
10. **updateMatingPool(self):** No change to this method, using the default one as it was supplied. This method updates the mating pool for each iteration
11. **newGeneration(self):** This will call the methods **binaryTournamentSelection**, use the returned parents for performing specified crossover **(self.crs\_over)** and pass on the result of crossover to specified mutation method **(self.mut\_meth)**.After performing selection, crossover and mutation, we will calculate the fitness (by calling **computeFitness** method defined in the module **Individual.py**) of resulting individual and call **updateBest method** to update the best solution. This process is done for whole population size for each iteration
12. **GAStep(self):** This one is used without any change (used as is from the provided code), which first updates the matingPool with the population by calling **updateMatingPool(self)** method and then calls **newGeneration(self)** method. These two methods are called in sequence (one after the other)
13. **Search(self):** This will call the method **GAStep(self)** for **self.max\_iterations** times

**Methods of file Experimental\_TSP\_R00195877.py:**

No new methods are added to this file when compared to “**TSP\_R00195877.py**” module, only change is in the implementation logic for uniformCrossOver and order1CrossOver. Instead of creating and returning one child, these methods in this module will create two children and return the fittest among them. Rest all logic remains same as in “**TSP\_R00195877.py**” module. In this way we are biasing our population towards the fittest individual, to update the mating pool at each iteration.

Please look into below to know how to run the modules to get the same results as presented in the report:

**1) Source code files included:**

1. Individual.py
2. TSP\_R00195877.py
3. Experimental\_TSP\_R00195877.py
4. TestTSP\_R00195877.py
5. EvaluationTSP\_R00195877.py

First two files are imported as modules in the last two files

1. TestTsp\_R00195877.py is used to run the basic configuration by importing the module TSP\_R00195877.py
2. TestTsp\_R00195877.py is also used to run the Experimental part by importing the module Experimental\_TSP\_R00195877.py
3. EvaluationTSP\_R00195877.py can be used to play around with the configuration dictionary specified in this file, to change population size, mutation rate and max\_iterations

**2) Requirements:** numpy, plotly graph objects, argparse

**3) How to run the code:** please refer to the Requirements above

Place all the source code files and data files in the same folder

For exact commands to run, to see the same result as shown in the report, pleas refer the file [“commands to execute.txt”](Commands%20to%20execute.txt)

|  |
| --- |
| **Conditions for error free execution:**  1. number of arguments passed should be exactly 3 (which configuration to run, file name and how many times we want to run that configuration for)  2. First argument helps in choosing the configuration to be run. It must be passed as an integer and it must be between 1 and 6 (both inclusive)  3. Second argument must be a file name  4. Third argument specifies how many times we want to run the configuration (specified in Frist argument), it must an integer  Failing to meet above conditions will result in error |

For **SECTION 1** part of the report:

To run the basic asked configurations please use below command:

>> python TestTSP\_R00195877.py <config number> <filename> <iterations>

* + - Config number **must be any one from [1,2,3,4,5,6]**
    - Filename : tsp instance file name
    - Iterations: how many times we want to run the specified config

For **SECTION 2** part of the report:

To Evaluate the performance of GA, please use below command:

>> python EvaluationTSP\_R00195877.py <config number> <filename> <iterations>

* + - Config number **must be any one from** [1,2,3,4,5,6]
    - Filename : tsp instance file name
    - Iterations: how many times we want to run the config
    - **Note:** Please make sure to alter the configuration dictionary in the file EvaluationTSP\_R00195877.py as required to alter the values of population size, mutation rate and max\_iterations

For **SECTION3** part of the report

To Analyse the performance over of GA on each configuration run over 5 times use below command:

>> python TestTSP\_R00195877.py <config number> <filename> <iterations>

* + - Config number **must be any one from [1,2,3,4,5,6]**
    - Filename : tsp instance file name
    - Iterations: how many times we want to run the specified config

For **SECTION4** part of the report

To run the experiment part of the report, please use below command:

* 1. >> python TestTSP\_R00195877.py <config number> <filename> <iterations>
     1. config number any one from [1,2,3,4,5,6] as as asked in the assessment document
     2. filename : tsp instance file name
     3. iterations: how many times we want to run the config

**Note:** Please make sure to import the module **Experimental\_TSP\_R00195877.py** instead of **TSP\_R00195877.py** (you can just comment and uncomment the import sections in the file **TestTSP\_R00195877.py**)

**Dictionary Configurations:**

1. "**eval\_configurations**" dictionary defined in the file "EvaluationTSP\_R00195877.py" as per required settings

eval\_configurations" dictionary key values represents the configuration number, and value for each key is a list as mentioned below

[popSize, mutation\_probability, max\_iterations, pop\_initialization\_method,

crossover, mutation]

1. “**basic\_configurations**” dictionary defined in the file “TestTSP\_R00195877.py” defines the basic configurations as it is asked in the assignment

“basic\_configurations " dictionary key values represents the configuration number, and value for each key is a list as mentioned below

[popSize, mutation\_probability, max\_iterations, pop\_initialization\_method,

crossover, mutation]