# A study of Genetic Algorithm solution approaches to the TSP

*Sanjyot Godbole (m20201103)*

*A project for the CIFO course, as part of the MDSAA programme*

## 1. Introduction:

Travelling Salesman problem (TSP) is a non-deterministic minimization problem which, given a set of locations and a starting point, attempts to find the shortest and most efficient route to visit all the locations and return to the starting point. TSP problems are tackled using several different optimization algorithms, one of them being Genetic Algorithm (GA). The results of a GA can change significantly based on its hyper-parameters, i.e. parameters used to control the learning process. Hence, it is vital to understand how to set the hyperparameters. By running some experiments, the author investigates the results from multiple datasets in an attempt to find the best control parameter combination.

## 2. Experiment Setup:

The experiment at-hand involves testing the performances of different control parameter values on three different datasets using the "*charles*" Genetic Algorithm Python library developed during the semester as a part of coursework.

### Data:

As per the directions supplied, three datasets were chosen. from the website by the University of Heidelberg. The following decision points were taken into account in selecting the datasets.

1. Choose problems of increasing size - roughly divided into "small, "medium" and "large". The small instance had 52 locations, medium had 101 locations and large had 280[1].
2. Availability of optimal solution for comparison
3. Data in Euclidean format

Based on these criteria, the datasets listed below were chosen.

1. 52 locations in Berlin (Groetschel); short name (Berlin52)[2]
2. 101-city problem (Christofides/Eilon) short name (eil101)[3]
3. Drilling problem (Ludwig) short name (a280)[4]

The first two are city tour problems and the last one is a drilling problem. The number of locations is available in the filename.

---

[1] Of course, there are still larger instances possible but given the time and computational resource constraints, it was decided to restrict the problem to a manageable size.

[2] elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/berlin52.tsp

[3] elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/eil101.tsp

[4] elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/a280.tsp

## 3. Methods - Models and Evaluation:

The first step in solving the problems is to calculate the distance matrices. Because all the problems are in the Euclidean space, the distance matrices are calculated by taking euclidean distances using the coordinates of every location. Distance matrices were created with the help of a freely available Excel utility called the FLP spreadsheet solver[5] and are made available for import in the code by a manual step to convert to CSV. The code is well documented in terms of implementation detail.

Depending on which dataset is chosen, the distance matrix to the corresponding dataset is passed to the Population class constructor in the "my_tsp.py" file. The testing for all the datasets is done for a constant population size of 100 and varying number of generations, selection algorithm, crossover operator and its rate, mutation operator and its rate, and either enabling or disabling elitism during the model run. A parameter grid consisting of 800 sets of different combinations of parameters is generated using the ParameterGrid from scikit-learn's"model_selection" module[6]. For every parameter set, the fitness of the best individual is extracted and stored in a Pandas DataFrame. Finally, this DataFrame is written to a CSV file for further analysis through visualization. The logs from Charles library are turned off.

## 4. Results and discussion:

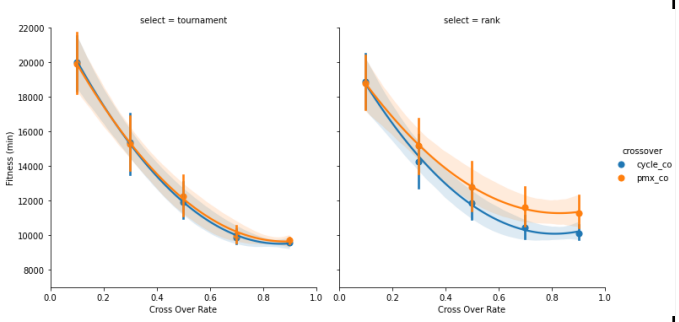
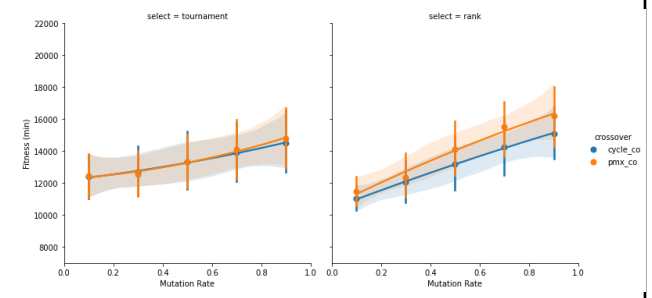
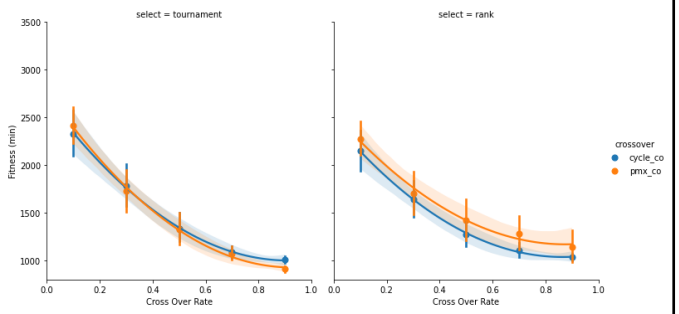
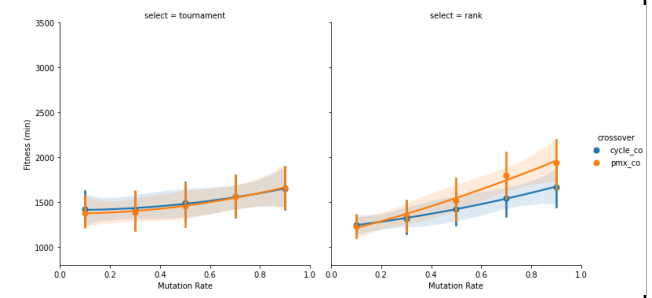
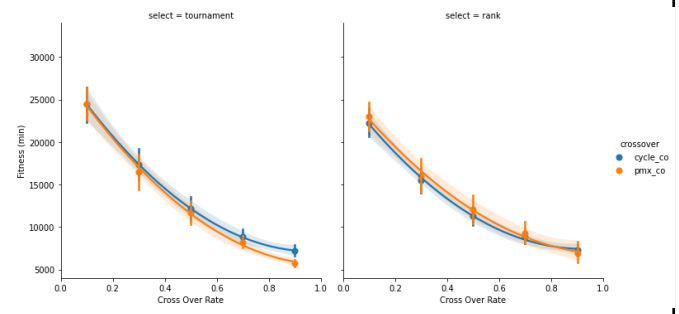
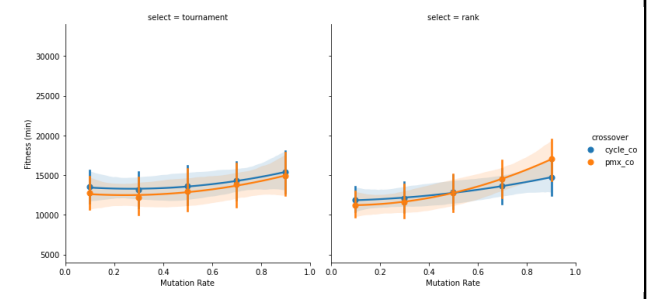
The behaviour of the model upon modifying parameters was consistent across all the datasets (problem instances)Some salient observations are:

1. For high crossover rate, cycle crossover operator when combined with the ranking selection algorithm performs better for a smaller tour, however, as the tour length increases, partially matched crossover operator paired with tournament selection algorithm returns shorter paths for the tour. Crossover rate has a significant effect on the performance of the genetic algorithm.
2. Lower mutation rate leads to the best fitness values, however as the distance matrix gets bigger or the number of stopovers in the tour increases, the effect of the mutation rate on the overall performance of the genetic algorithm decreases.
3. A lower mutation rate paired with partially matched crossover operators leads to a better solution for bigger distance matrices. When a partially matched crossover operator is used the performance of the genetic algorithm results in little improvement. Hence, a cycle crossover operator paired with a ranking selection algorithm may lead to a more optimal solution.
4. However, as the number of generations increases, the partially matched crossover operator performs well especially when selection is done using the ranking algorithm.

[5] https://www.euro-online.org/websites/verolog/flp-spreadsheet-solver/

[6] The FPS selection algorithm was not used for this modeling study because in the author's understanding it is not appropriate to be used for a minimisation problem such as the TSP. Additionally, only the "swap" mutation operator was used for the evolution scheme.

5. Elitism leads to significantly good solutions for a larger number of generations, especially for larger datasets.

| | Crossover Rate<br>(Hue: Crossover ) | Mutation Rate<br>(Hue: Crossover ) |
|---|---|---|
| Berlin52<br>(Col:<br>Selection) |  |  |
| Eli101<br>(Col:<br>Selection) |  |  |
| A280<br>(Col:<br>Selection) |  |  |

## 5. References

Due to the fact that this project was and empirical study implemented by the author herself, there are no references as such to cite other than those linked in the project description document supplied

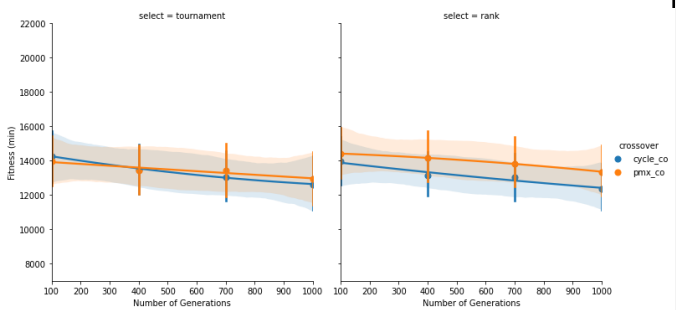
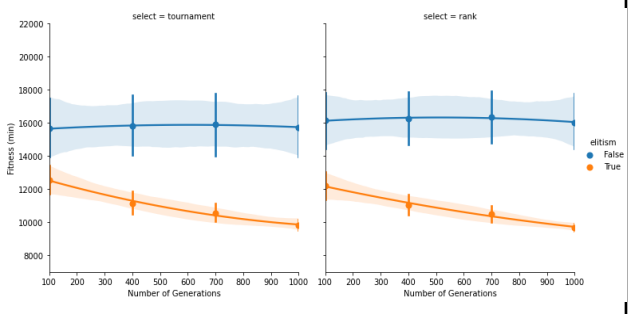
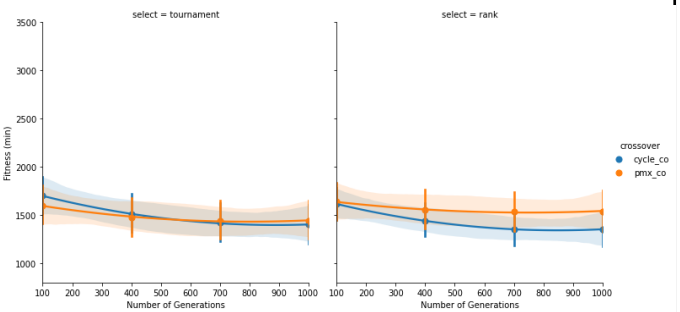
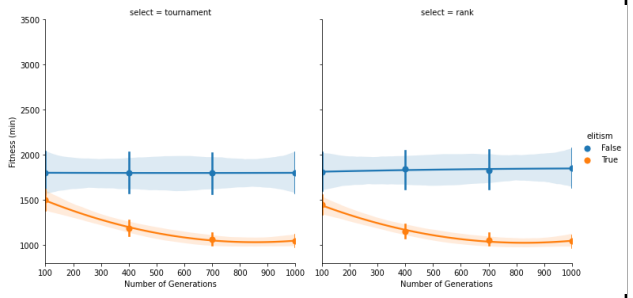
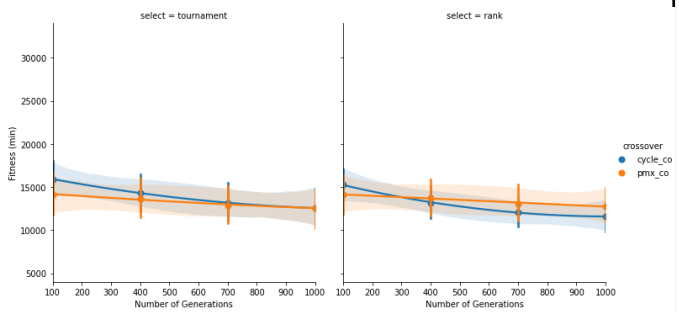
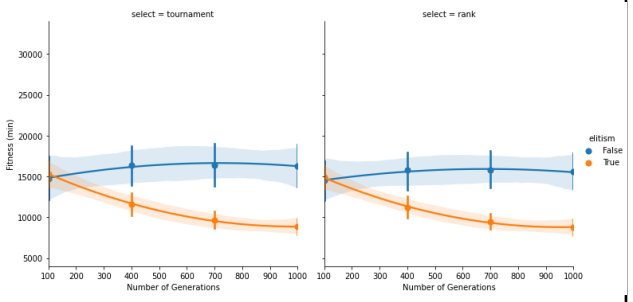
## 6. Appendix

Code assets (the code is available on GitHub[7]):

1. *my_tsp.py*: Contains the main code that executes the model. When the user runs the code, he/she must select the problem when prompted in the command line.
2. *Visualisation.ipynb*: Contains the code to generate the visualisation presented in this report. If the reader finds it difficult to

[7] https://github.com/SanjyotGodbole/cifo_tsp_ga_experiment

3. The "*charles*" module folder contains the code that was developed for the course. A small customisation was made in the code wherein a class attribute was added, and is initialised in the constructor. This attribute prevents the code from printing the best individual in every iteration of the population's generation evolution.

| | Number of Generations | |
|---|---|---|
| | Crossover Rate<br>(Hue: Crossover ) | Mutation Rate<br>(Hue: Elitism) |
| Berlin52<br>(Col:<br>Selection) |  |  |
| Eli101<br>(Col:<br>Selection) |  |  |
| A280<br>(Col:<br>Selection) |  |  |