E.C. Assignment 1:

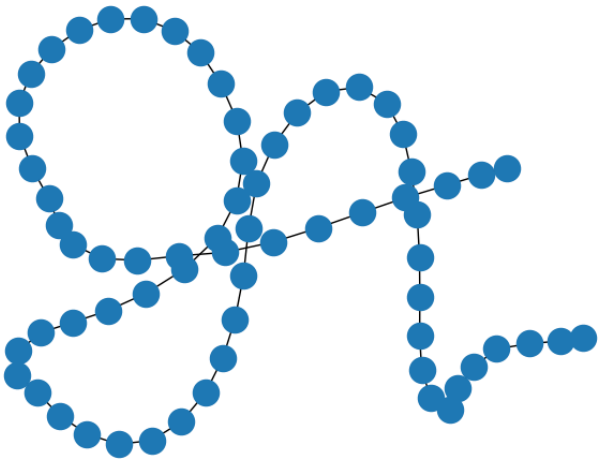
Group Niners: A1865053, A1886739, A1716006, A1845834, A1878373, A1853059

Exercise 1:

In this assignment all members played a part in the success of the project. Damian (A1865053) wrote the individual and population class with Rishi. He also combined the mutation, operator and selection classes to provide the foundation for each experiment. Rishi (A1886739) wrote the Individual and Population classes with Damian and also implemented the Inver-Over operator for exercise 7. Further, Rishi improved the previous code speed by multi-threading parts of the process. Mitchell (A1716006) implemented methods for the mutation class (Mutation.py) and played a large role in documentation of the report and the source code as well as debugging. Runze (A1845834) wrote the selection class and commented on it. Thai Duy Anh (A1878373) implemented the crossover operations and debugged code. Further, Thai Duy Anh helped build the evolving pipeline for exercise 6. Gerald (A1853059), wrote the TSProblem class in TSProblem.py, and parts of the experiments, he also ran and recorded results from the experiments.

Exercise 2:

The TSP problem class is represented by TSProblem.py. This class takes in an input file and calculates the edge costs, as well as visualizes a random-walk path using Networkx. An example of the random walks to represent the problem - where nodes are cities and an edge is a walk from one city to another - may look like:



Exercise 3:

Here we used a random walk approach to the problem in the Individual.py file where the Individual class exists. Here, you can print the random walk for a given file through the main.py file.
Likewise, the Population class to model a collection of individuals is within the Population.py file.

Exercise 4:

Operations for generating mutations in the genome of individuals can be found in the Mutation.py file. Similarly, the Crossover operators to generate new children can be found in the Crossover.py file.

Exercise 5:

As with the mutation and crossover operators, the selection methods to determine who survives and who dies can be found in the Selection.py file.

Exercise 6:

A series of tests and experimentations were set up to help determine the optimal combination of different evolutionary algorithm strategies. results of the experiments can be found in experiment_1.xlsx, experiment_2.xlsx and experiment_3.xlsx. As usa13509 is such a large problem, only some measurements were able to be recorded such as, population size = 50 at 20000 generations. However, none of the population size = 100 at 20000 generations measurements were able to be taken. For all generation limits of [5000, 10000, 20000] the best individual cost, the average cost, max cost, median cost and standard deviation were taken. The aforementioned information can be found in the experiment files mentioned above. Each experiment can be run using the experiment_(n).py file for whichever experiment you would like to try, with n in [1, 2, 3].

Intuition for experiment 1:
Experiment 1 (experiment_1.py) uses swap mutation, as it was believed that order should be preserved, specifically later into the generations where nearby cities should be adjacent to one another in the tour. Hence, minimal perturbation to the walk should be performed as possible. Likewise, order crossover was used for experiment 1, so that adjacent cities (which should form over time) are typically maintained in order, although the set to swap is randomly chosen. It was thought that this combination of randomness with maintained order would perform well.

Intuition for experiment 2:
Experiment 2 (experiment_2.py) uses inversion mutation as it is a good way to explore local changes, where local denotes small changes between two points. This mutation operator is useful for finding better arrangements of local cities. Comparably, PMX crossover was used for experiment 2, as it ensures that each offspring takes segments from the two parents while preserving some ordering of the cities. PMX is useful for passing good blocks of information from the parents to the offspring. The mixture between maintaining local order and changing global order is believed to be a good balance of exploration and exploitation such that we would get a strong evolutionary algorithm for the TSP.

Intuition for experiment 3:
Experiment 3 uses order crossover and scramble mutation. Order crossover, again, maintains a relative ordering of cities in the parents. This is central to ensuring that a good solution as the TSP tour quality is directly influenced by the ordering of the cities. scramble mutation was used here so that large diversity is mixed into the relatively maintained ordering. The problem with order crossover is that it can get stuck in a local minima quickly. Scramble mutation however should be able to move out of that local minima by increasing variability without drastically disrupting the tour's structure.

Best model:
The best model was experiment 2 ,inversion mutation with pmx crossover. The results of the models are presented beneath. It was found that most algorithms converge or become stuck in a local minima which likely is the cause of no deviation. To run it just run experiment_2.py, which outputs to a csv file more data than the table beneath provides. Experiment_2.py provides minimum fitness, max fitness, median fitness among other things.

	File	Mean Fitness	STDEV Fitness
121	./pr2392.tsp	28789722.84	351.7523
125	./pcb442.tsp	29940639.62	0
129	./lin105.tsp	29984955.72	0
133	./kroC100.tsp	29977499.29	0
137	./kroA100.tsp	29977759.44	0
141	./st70.tsp	29999274.99	0
145	./eil101.tsp	29999310.54	4.82E-14
149	./eil76.tsp	29999406.52	0
153	./eil51.tsp	29999543.31	2.73E-14
157	./usa13509	405910337.7	34914.98

Exercise 7:

As required the group has implemented the inver-over algorithm in accordance with the paper. You can call the Inver-Over class to run it according to specifications, but we have automated this in experiment_4.py, where it is called and all values are recorded in a csv file.

	File	Population Size	Generation	Mean Fitness	STDEV Fitness
59	eil51.tsp	50	20000	1.638183e+03	110.668508
139	eil76.tsp	50	20000	2.514062e+03	109.518222
219	eil101.tsp	50	20000	3.454188e+03	113.166330
299	st70.tsp	50	20000	3.672792e+03	193.634653
379	kroA100.tsp	50	20000	1.709919e+05	9180.694389
459	kroC100.tsp	50	20000	1.697786e+05	7562.229815
539	lin105.tsp	50	20000	1.237789e+05	6053.924872
619	pcb442.tsp	50	20000	7.684347e+05	16397.208917
699	pr2392.tsp	50	20000	1.526484e+07	130439.445527