Project 5 IT3708 Torgeir

A)

For this implementation, I created a class called Genotype. Objects of this class hold a byte array which represents the actual genotype of the individual. The array is initialized to simply hold the indices of the cities, and it is then shuffled to a random order. Conceptually, the sequence will always contain the same values, where the information lies in what order they are in. The mutation operator is implemented in a way where there is a probability given by the parameter for mutation rate, with this probability there will be chosen two indices in the sequence. The sequence of indices that lies between the first and second of these, with wrap around being taken into consideration, will be reversed in order. Worth noting is that my implementation of the mutation and crossover operators is that they do not modify the existing genes, they create new ones, then make sure the new ones end up as copies of the existing ones before making modifications. The crossover operator will similarly with a given probability mark two random indices. The area between these cuts is copied from the first parent onto the child. For the remainder of the parent, the order is given by the second parent. The number after the last value from the first parent will be the first value after that value in the second parent. Then the second parent's sequence is traversed until the child has all the unique values.

My operators will not create solutions that are invalid TSP solutions. They are all different permutations of the same indices. Making sure this happens seemed like a much better strategy than having to handle invalid solutions in some way.

The program essentially works by initializing the population with the given number of individuals. Individuals have their fitness scores evaluated as soon as they are created, and this is stored in the individuals, so it does not have to be calculated again. The rank and crowing distance is calculated before starting the loop. Then for however many generations that will be run, the first thing that happens is a tournament selection. A group of size equal to one tenth of the population size is filled with random individuals from the population. With a probability of 5%, a random one of these is chosen as a parent, and with 95% chance, the one that is best according to rank and crowding distance within the group is chosen as a parent. Then a second parent is chosen using the same method from a new group. These parent create a new child by the use of crossover then mutation, which creates one child which is added to the child list. This is repeated until the child list has the same size as the population. Then all the children are added to the population and the child list is emptied.

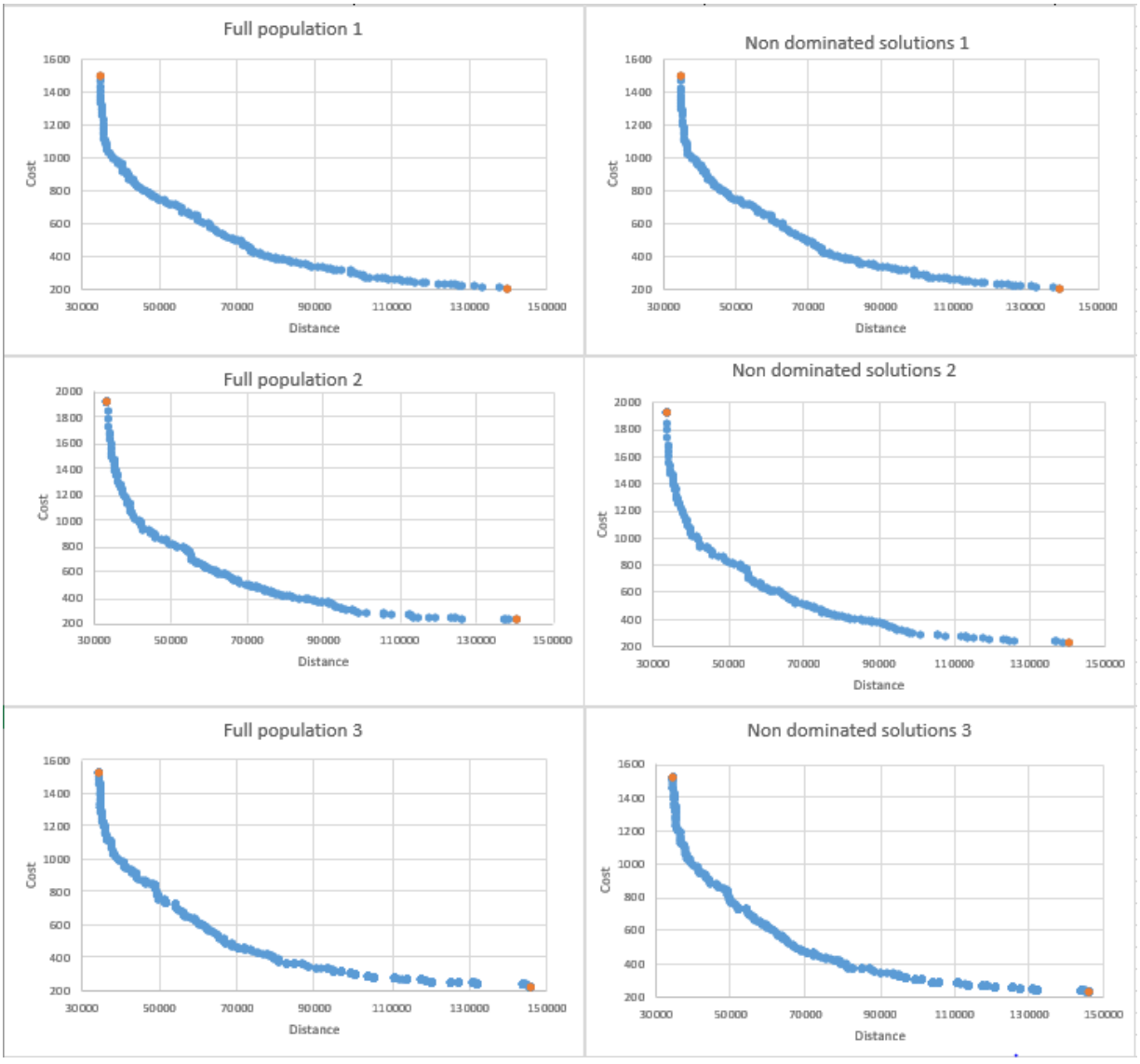
The next step is calculation of rank and crowing distance. The rank is calculated first, placing the individuals in a two dimensional ArrayList where the outer dimension is the rank, and the inner lists are the lists of individuals with the given rank. Calculating the crowing distance is then done for each of these lists. The list is sorted by one of the two score dimensions. Since they do not dominate each other, this means that the other score dimension. The first and last individual in the front get crowding distance set to Double.MAX_VALUE. The other individuals have their crowing distance set based on how big of a difference the next and previous individuals have in the scores compared to the first and last. Afterwards, the population is trimmed. This is done by creating a new list which is filled with individuals. As long as the fronts calculated in the previous step fit entirely into the new population without having the new population go above the given population size, they can be added in. When some front is too big, then the individuals are picked based on crowding distance. The one that has the highest crowding distance is removed from the front and added to the new population.

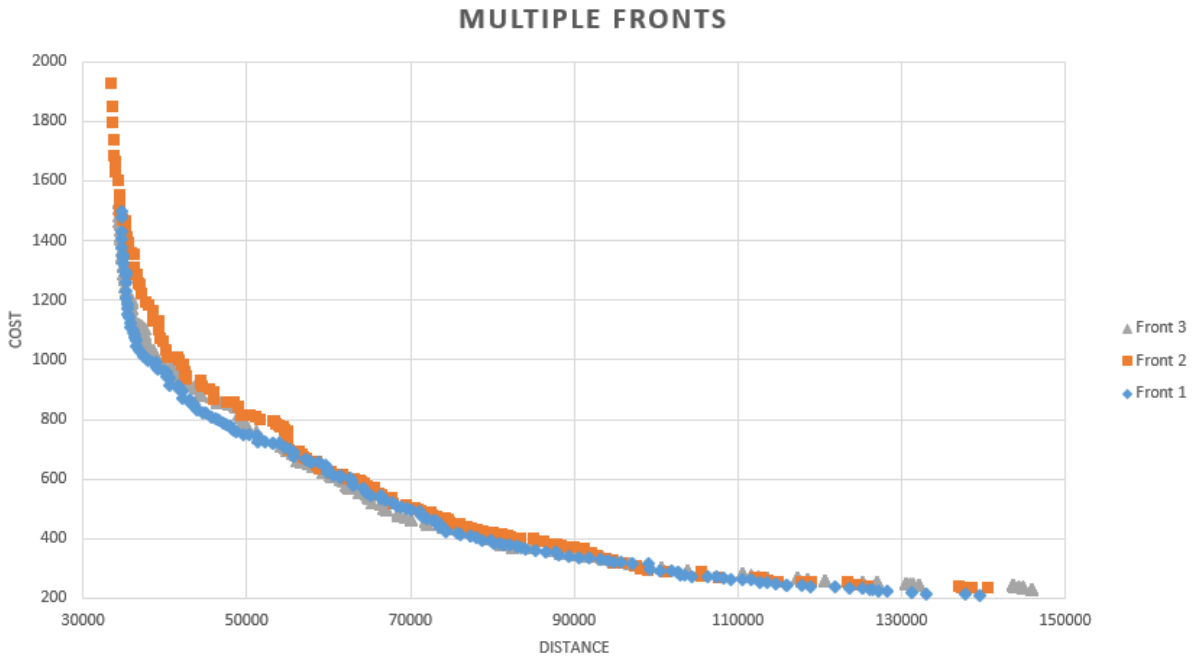
If no individual with a crowding distance above 0.0 is found, then a random one is picked. This is to ensure that if there are duplicates, they should not be picked from the same region all the time.

B)

Run	Population size	Generations	Crossover rate	Mutation rate	Best cost	Worst cost	Best distance	Worst distance	Non dominated	Unique solutions
1	300	4500	0.6	0.5	211	1498	43673	139552	300	175
2	1000	2000	0.3	0.5	232	1925	33551	140639	1000	166
3	2000	500	0.7	0.6	231	1528	34292	145840	2000	171

C)




MULTIPLE FRONTS
2000
1800
1600
1400
1200
1000
800
600
400
200
COST
30000
50000
70000
90000
110000
130000
150000
DISTANCE
Front 3
Front 2
Front 1