Assignment 2 Report

Andrew Watton 100517870
Mirical Williams-Causton 100552082
Matthew McCormick 100448975

Overview & Termination Condition

Our algorithm is a fairly simple genetic algorithm. It is partially steady state in that the population's member of best fitness is always selected and never mutated. We implemented two methods for selection and two methods for crossover and will display test results to determine which are most effective. Our algorithm uses a set maximum number of generations as a main termination condition. Once an optimal solution is known, however, the algorithm can substitute it as a secondary termination condition in order to assess its performance.

1. Initialization

To initialize a population, our algorithm creates a given number of permutations (or different orderings) of the set of cities. When a set is created, it is only added to the population if it is unique with respect to the existing members of the population. This ensures that no two members of the initial population are the same. The number of members for the initial population, and the number of cities each member is to have are passed as parameters. This process was completed similarly for both problem sets.

2. Evaluation

The fitness of a given solution to the travelling salesman problem is determined by how short a distance the salesman must travel using that solution. This number is calculated by adding the distance between each consecutive pair of cities in the set, then adding the distance between the final city and the first city (to complete the loop). Since higher fitness is better, we need to define fitness by 1/ distance. For example:

```
TotalDistance([city1, city2, city3, city4])
=    DistanceBetween(city1, city2)    +    DistanceBetween(city2, city3)
+    DistanceBetween(city3, city4)    +    DistanceBetween(city4, city1)

Fitness = 1.0/TotalDistance([city1, city2, city3, city4])
```

For the first problem domain, the distances between each pair of cities is calculated from the given coordinates and stored in a multidimensional list for later use. For the second problem set, this set is given and does not need to be calculated.

3. Selection

Originally, our algorithm employed a simple truncation selection. Given N original members, our crossover section builds the population up to either 2N or 3N (depending which crossover method is used). Our truncation selection would order the population by fitness and select the best N members.

After doing some research, we realised that truncation methods are rarely used because they eliminate the possibility for weak members to be selected entirely. This doesn't sound like a problem, but some of these weak members might be very close in structure to the optimal solution (with a few crossovers and mutations), whereas members with better fitness might be further in structure from the optimal solution.

Instead, we wanted to give better solutions a much higher chance of being selected, but not completely throw out all less fit solutions. We implemented a modified version of a roulette selection algorithm, but left in the truncation method for testing.

The roulette wheel selection algorithm works by assigning a probability to each member based on fitness. To do this, we add the fitness of each member together, then each members probability becomes its fitness/ total fitness.

Members are then selected for the next generation semi-randomly, weighted by their probability. The most fit member is selected automatically, however, which is one way our algorithm is partially steady-state and not generational.

4. Crossover

Two different crossover methods were implemented. The first method - which we're calling **Twin Crossover** - pairs up every set of two members in the population and creates two children for each pair. One child keeps the order of the best pairing of cities in the first parent, while filling in the rest of the values in roughly the order they take in the second parent. The second child is the same thing but with the parents switched.

The second method - which we're calling **Only Child Crossover** - only creates one child for each pair of parents. This child contains the first parent's best paring and as close as possible to the second parent's best pairing (without causing a conflict), then fills in the rest of the cities randomly.

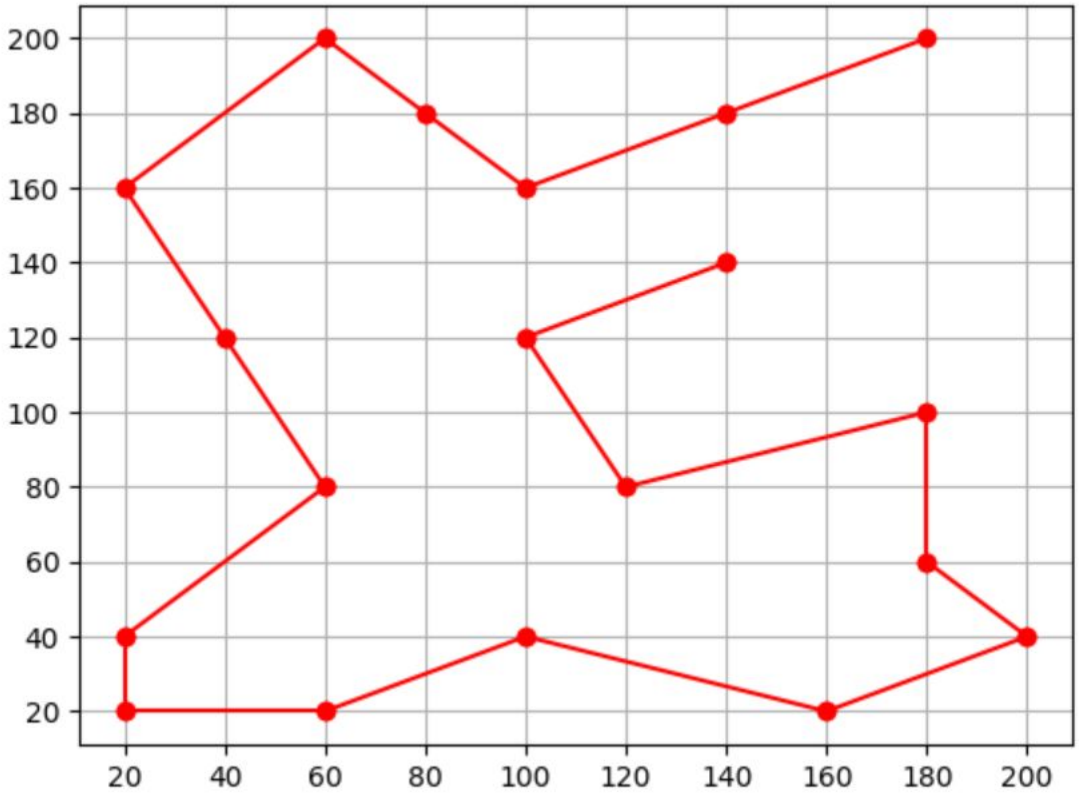
Both methods retain all of the parent members at least until the next selection time (which continues the partially steady state nature of our algorithm). Both methods also include a loop to add unique new members until they achieve their population size goal. This is done so that we can exclude children who already exist in the set, while keeping strict control over our population size.

5. **Mutation**

We only implemented one mutation method. We start by randomizing an integer with a small range. Then every member of the population that is divisible by that integer gets mutated. The mutation process involves simply selecting two random cities from the member and switching their index. We will test different ranges for the original integer to see just how much mutation is optimal.

Graphical Representation of Solutions

Best Path for First Problem Domain



Best Path for Second Problem Domain

```
Shortest path: Oxford -> Bristol -> Liverpool -> Glasgow -> Manchester -> Cambridge -> London -> Brighton
Total Distance:  1355.0
```

Results

The following table shows the effectiveness of each genetic algorithm strategy. Effectiveness is measured by the average number of generations the strategy requires to arrive at the optimal solution. Each strategy is run 200 times to find its effectiveness. Truncation/ Roulette Selection and Twin/ Only Child Crossover are described in the algorithm overview. Mutation Modulus is a measure of how many members of the population get mutated every iteration.For Mutation Modulus = N, every Nth member of the population is modified, excluding the current best fitness solution.

Average # of Generations Taken to Arrive at Optimal Solution

Mutation Modulus	Truncation & Only Child	Truncation & Twin	Roulette & Only Child	Roulette & Twin
1	18	7	22	17
3	46	9	57	28
5	54	12	89	30
10	71	15	163	41

Division of Work

Andrew - Main algorithms, crossover and selection methods
Mirical - Initialization and evaluation methods
Matthew - Mutation method and graphical output