VERKFRÆÐILEGAR BESTUNARAÐFERÐIR


# Day 8 - Group 2

T-423-ENOP

*Arnar Gylfi Haraldsson*
arnarh23@ru.is

*Hafþór Árni Hermannsson*
hafthorh20@ru.is

*Ragnheiður Gná Gústafsdóttir*
ragnheidurg@ru.is

May 12, 2024

# Exercise 1

## Description

Implement the genetic algorithm using floating point representation and self-adaptive mutation rate. Test the following functions:

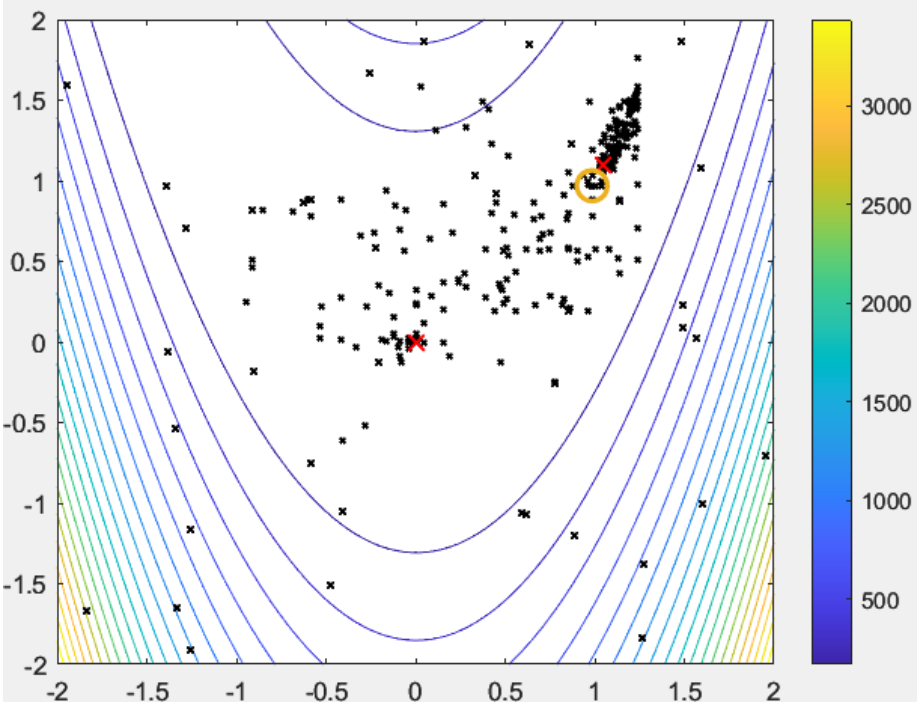
1. Rosenbrock function (n = 2, $-2 \leq x_i \leq 2$),
2. Function fp (n = 1, 2, $-2 \leq x_i \leq 2$),
3. Auckley function (n = 1, 2 and 3, $-10 \leq x_i \leq 10$).

## Solution

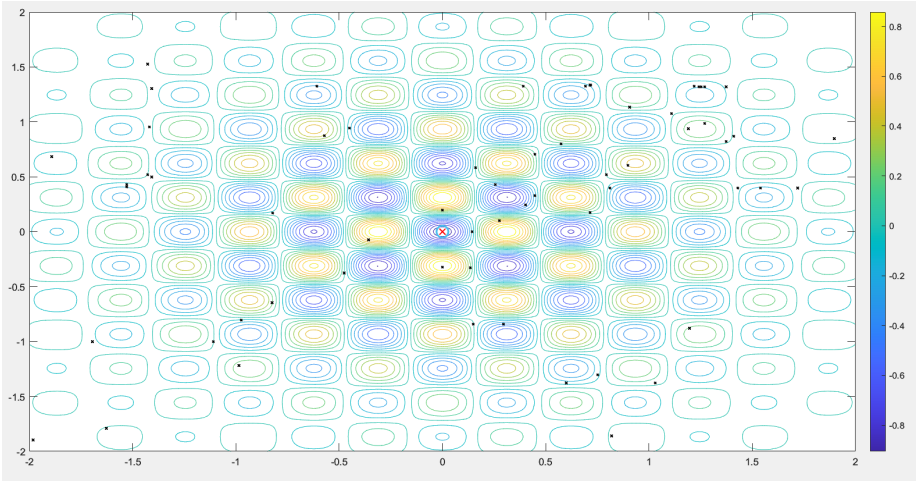
The algorithm starts by initializing a population of random individuals within the specified search range. It then evolves the population over a number of generations, using tournament selection, crossover, and mutation operations. The self-adaptive mutation rate is designed in the exact same way as in the example given by lecture slides for day 8.

The algorithm prints the iteration number, the number of function evaluations, the current best solution (xbest), and the corresponding best fitness value (fbest). Additionally, for two-dimensional problems, it displays the contour plot of the objective function along with the population and the best solution found so far.

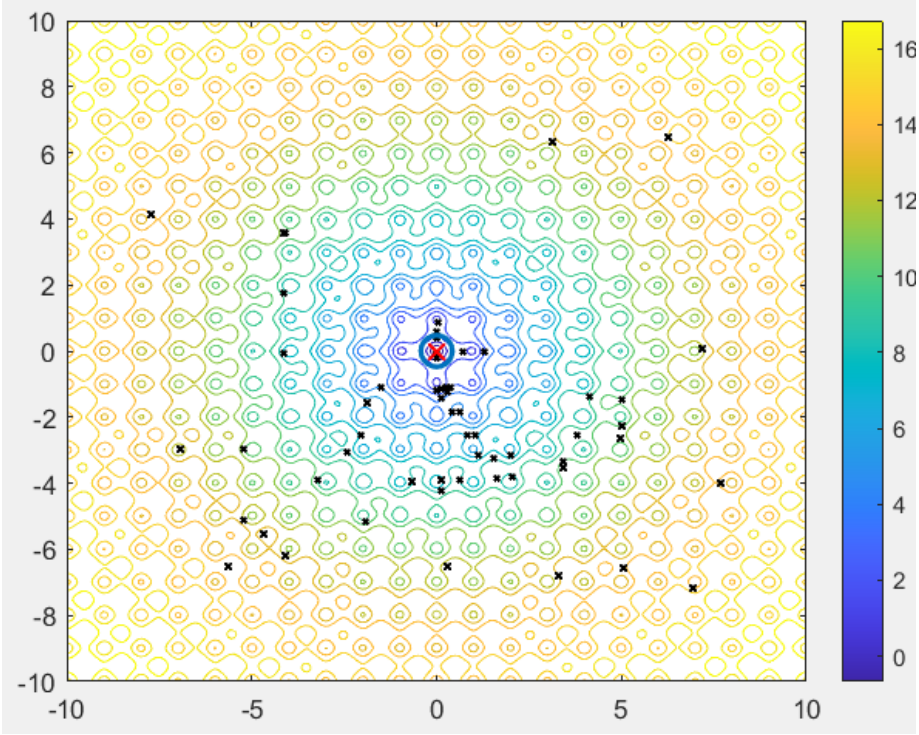
The algorithm stops when it reaches the maximum number of generations specified (max_gens). Using tournament size 5, N = 40, 60 generations, pm = 0.2 and pc = 0.7. We get this example output from the Rosenbrock function.




Iteration: 60, function evaluations: 24000 xbest: [0.9851 0.9691], fbest: 0.000386.
The algorithm seems to be converging too soon. So we adjust the mutation rate so that it increases if the population diversity, measured by standard deviation, is less than 0.2, rather than 0.1. This only helped marginally. Prioritizing population size over iteration number also helped the algorithm find the true global minimum. Using N = 100 and only 20 iterations leads the algorithm to find the global minimum more reliably but maintains the number of function evaluations. Forcing the current best solution into the new population each iteration also seemed to improve performance.
For the two dimensional p-function we get the following graphic:




And we for some reason converge immediately on the correct solution, only with N = 20 and tournament size of 3. [Iteration: 2, function evaluations: 240 xbest: [0 0], fbest: -1.000000]
Same with the two-dimensional Auckley function:




Iteration: 2, function evaluations: 240 xbest: [0 0], fbest: -1.718282

# Exercise 2

## Description

There are N cities with the distance between city i and city j given by dij. Problem: find a close route through all the cities so that minimizes the total route length. Develop and implement an evolutionary algorithm for TSP.

1. Representation: permutation of the numbers 1 to N.
2. Mutation: random swapping of two cities on the route.
3. Crossover: develop an operator that maintains feasibility of individual (i.e., an individual created by crossover is still a valid permutation of integers 1 to N).
4. Adaptive mutation rate: monitor population diversity and use it to adjust mutation rate following the general rules discussed during the lecture.

Test the algorithm on randomly generated cases of the sizes 20 to 500. Implement some sort of visualization. Compare the results with the simple random swapping algorithm developed on Day 2.

## Functionality

The Traveling Salesman Problem (TSP) evolutionary algorithm we implemented in MATLAB is designed to find an approximate solution to the problem by using genetic algorithms. The main components of the function are:

### Initialization

The function initializes with the creation of a random set of city locations and a distance matrix that holds the distances between each pair of cities. The initial population of routes is generated randomly.

- **Population Generation:** Each individual in the population represents a possible solution (route) and is initially generated randomly.
- **Distance Matrix Calculation:** The Euclidean distance between each pair of cities is calculated and stored.

### Evolutionary Process

The core of the algorithm involves simulating evolution through evaluation, selection, crossover, mutation, and elitism to evolve the population towards the best solution.

- **Fitness Evaluation:** Each route's fitness is evaluated based on the total distance traveled.
- **Tournament Selection:** A subset of the population is selected randomly, a tournament of three are selected from there and the best routes are chosen to mate.
- **Crossover:** Routes from the mating pool are combined to form new routes. Different crossover methods can be applied based on the configuration. We did three methods.
    1. Our first method developed, named "crossover" in the code and is used by inserting 0 in the crossover slot in the TSP function in the main file, is not quite a crossover, as we misunderstood the concept. It takes random indices of the parents and swaps the values under those indices in the child with the value within that same child which corresponds with the value under the same original index in the second parent. As we did realise it was not really a crossover, we developed another one.

2. The second method we implemented was to take a random number of indices and have child one inherit those first indices values from parent one and then fill up the remaining slots with the values from parent 2 in the same order they appear in parent 2, skipping all the cities already included from parent one. The same was done for child two the other way around. This function is named crossover2 in the code and is used by inserting 1 in the crossover slot in the TSP function in the main file.
3. For comparison sake, we made a function for the Ordered crossover algorithm, which starts by selecting two crossover points. The offspring inherit the order of a continuous subset of cities (the segment between the two points) from one parent. The rest of the cities are filled in from the second parent, preserving their order but starting from the second crossover point and wrapping around to the beginning of the chromosome, skipping any cities already included from the first parent. This algorithm is used by inserting 2 in the TSP crossover slot in the function in the main file.

- **Mutation:** Routes are randomly mutated to maintain genetic diversity by swapping two cities.
- **Elitism:** The best route is always carried over to the next generation to ensure that the best solution found so far is not lost.

### Adaptive Mutation Rate

The mutation rate is adjusted based on the diversity of the population, with the goal of escaping local minima and ensuring a robust search of the solution space. If the diversity is over 0.1 the mutation rate is multiplied by 0.9, if not it is increased by multiplying it with 1.2.

### Visualization

If enabled, the algorithm provides real-time visualization of the best route as it evolves through generations, along with a display of the best and average fitness over generations.

### Convergence Monitoring

At the end of the simulation, the algorithm outputs the best route found, the total distance of that route, and a plot showing the evolution of the best and average distances, which helps in analyzing the performance and convergence of the algorithm.

### Utility Functions

Several helper functions are used to support the main evolutionary process:

- `path_length` calculates the total length of a given route.
- `create_distance_matrix` generates the matrix of distances between cities.
- `adjust_mutation_rate` modifies the mutation rate based on population diversity.
- `mutate`, `OXcrossover`, and other crossover functions manipulate genetic material to generate diversity.

## Result

We decided to compare the ordered crossover to our crossover algorithm. Table 1 compares the best distance and average distance between the two with N=50 and 50 generations and figures 3 to 6 visualizes the differences.

| Algorithm | Best distance | Average distance |
|---|---|---|
| Ordered crossover | 1616.4705 | 1733.6147 |
| Our crossover | 1649.1065 | 1783.2799 |
| Difference | 32.6360 | 49.6652 |
| Improvement % | 2.02% | 2.87% |

Table 1: Ordered crossover vs. our crossover

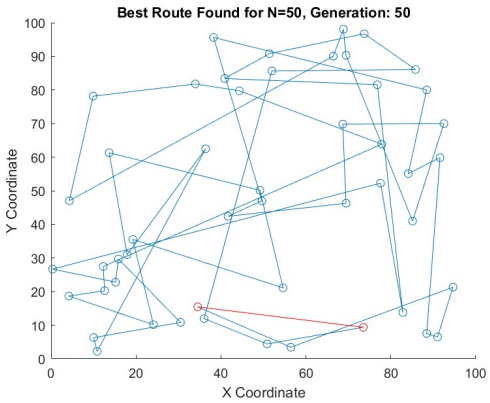



Figure 1: Best route.

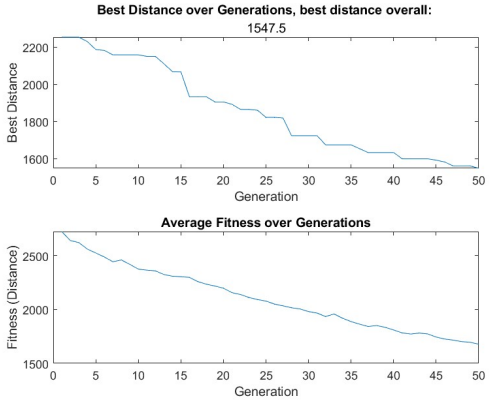



Figure 2: Graph of the best distance and the average distance of the population, both in terms of generations.

Figure 3: The ordered crossover algorithm.

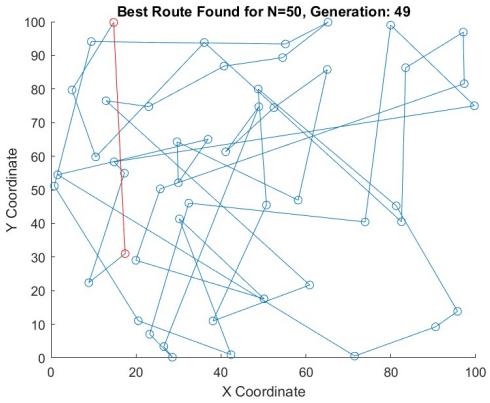



Figure 4: Best route.

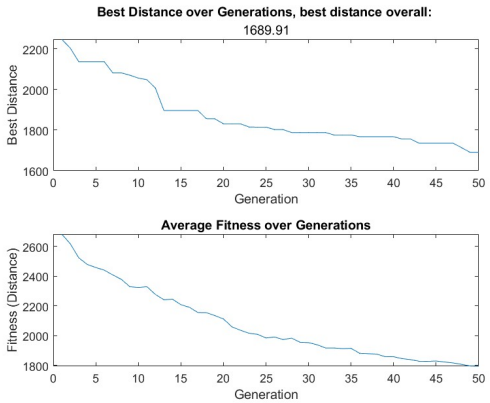



Figure 5: Graph of the best distance and the average distance of the population, both in terms of generations.

Figure 6: Our crossover algorithm.

When compared to the TSP algorithm that was developed int he first week of the course, the new algorithm developed on day 8 performed much better, this is numerically shown in table 2 and visually displayed in figures 9 to 24. As the table shows, this new algorithm is showing an improvement on average of 17.63%, it is also noticeably quicker when running, to see that please run the code that is turned it with the report.

Table 2: Comparison of TSP Solution Quality: Day 8 vs. Day 3

| N | Day 8 Best Distance | Day 3 Best Distance | Improvement % |
|---|---|---|---|
| 64 | 2324.5241 | 2881.5563 | 19.33 |
| 343 | 14254.9947 | 17759.8555 | 19.73 |
| 459 | 19543.5942 | 22678.2462 | 13.83 |

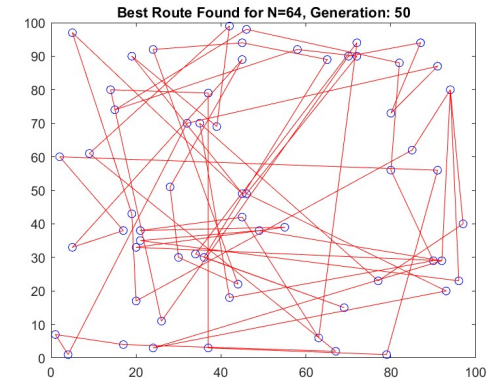



Figure 7: Best route.

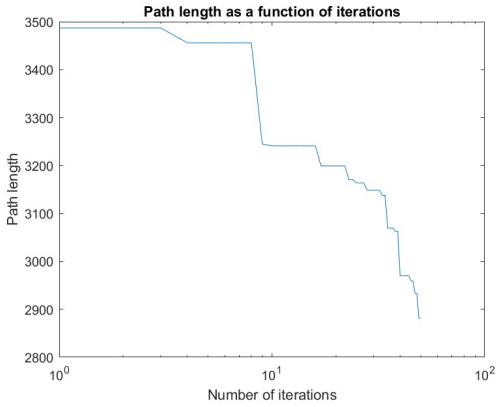



Figure 8: Graph of the best distance of the population, in terms of generations.

Figure 9: Day 3 TSP algorithm with N = 64 and 50 generations.

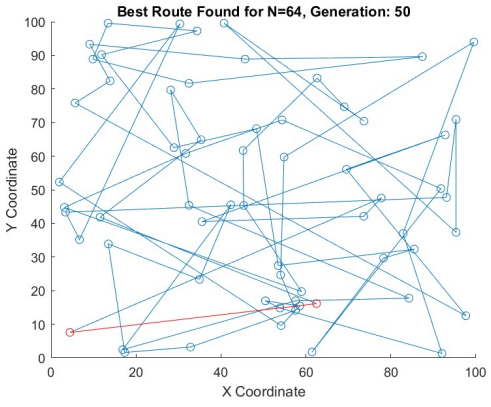



Figure 10: Best route.

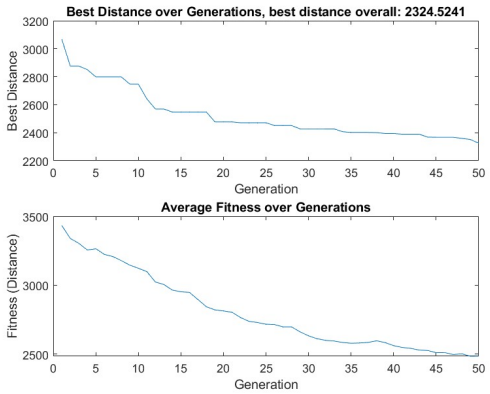



Figure 11: Graph of the best distance and the average distance of the population, both in terms of generations.

Figure 12: Day 8 TSP algorithm with N = 64 and 50 generations.

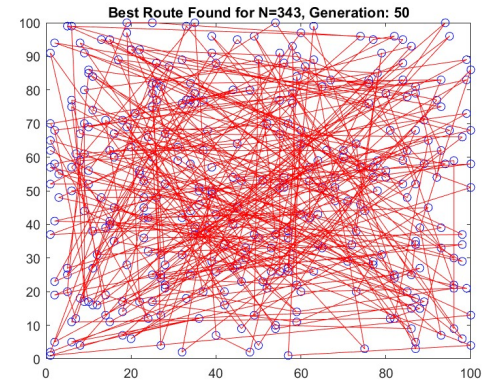



Figure 13: Best route.

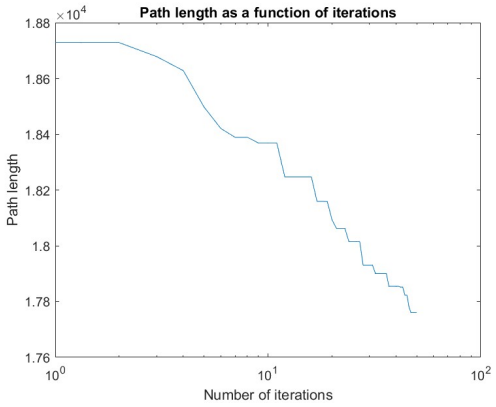



Figure 14: Graph of the best distance of the population, in terms of generations.

Figure 15: Day 3 TSP algorithm with N = 343 and 50 generations.

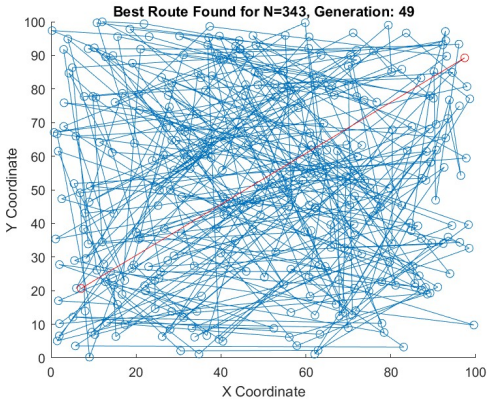



Figure 16: Best route.

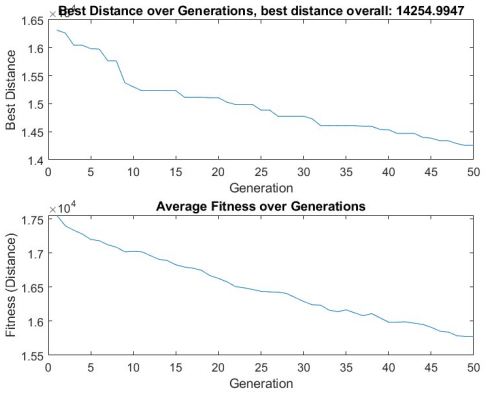



Figure 17: Graph of the best distance and the average distance of the population, both in terms of generations.

Figure 18: Day 8 TSP algorithm with N = 343 and 50 generations.

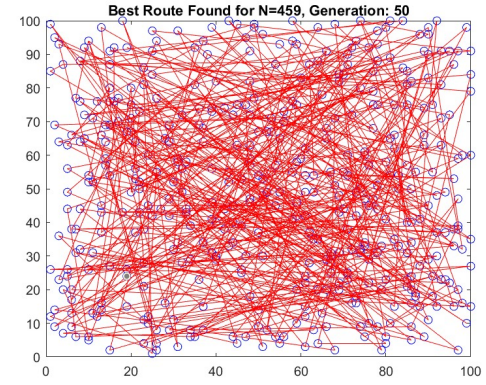



Figure 19: Best route.

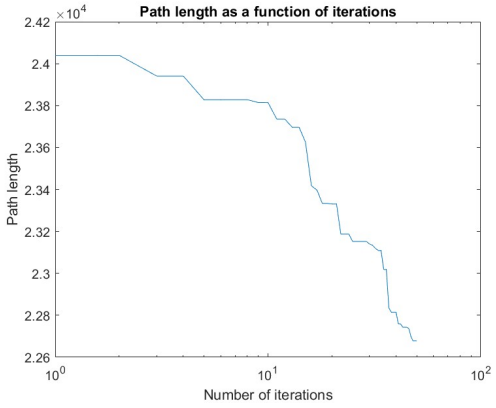



Figure 20: Graph of the best distance of the population, in terms of generations.

Figure 21: Day 3 TSP algorithm with N = 459 and 50 generations.

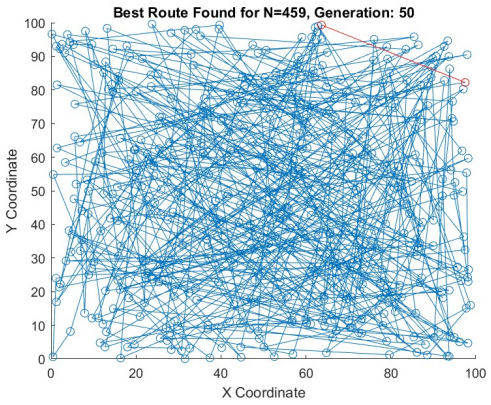



Figure 22: Best route.

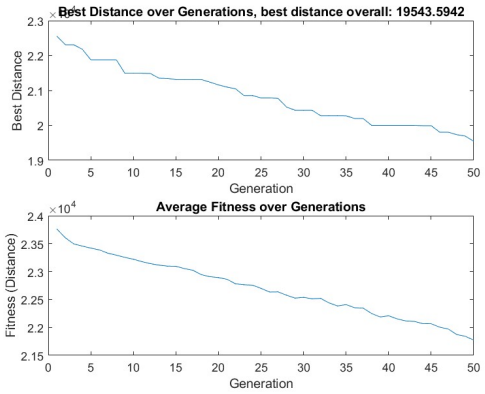



Figure 23: Graph of the best distance and the average distance of the population, both in terms of generations.

Figure 24: Day 8 TSP algorithm with N = 459 and 50 generations.