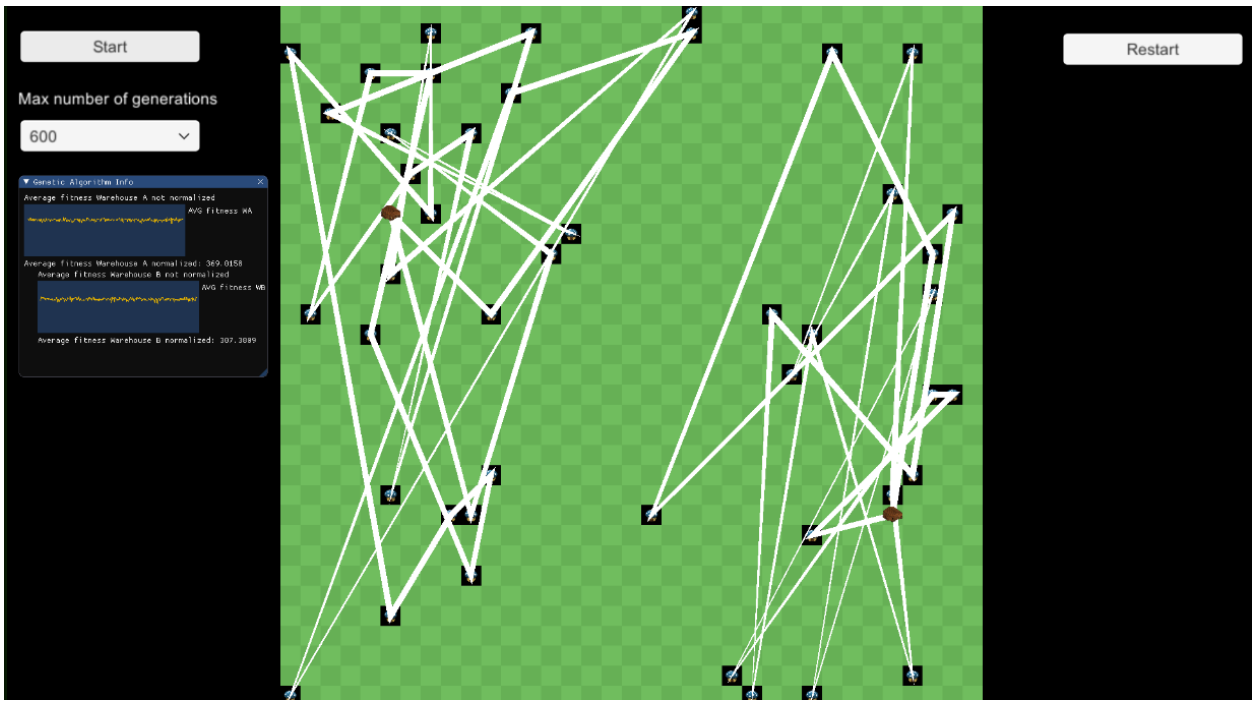


CS 657 - Assignment 3

10.25.2022

—

Andrick Mercado

Overview

In this assignment we were asked to create an algorithm based on the generic algorithm, to create paths for two warehouses for them to deliver to cities in the least amount of traveled distance, where we had the foundation to develop the algorithm only things not being specified were the crossover and the mutation algorithms.

Goals

1. Make a solution that produces optimal solutions
2. Produce good visuals

Structure

For this assignment I went for an object-oriented approach to this problem, its nature being essentially the same as the traveling salesman problem, but with an added dimension, this just being another salesman person. I deconstructed the genetic algorithm to take 4 parts Genes, Chromosomes, Population, and the solution that brings everything together. The gene class contains the location of a house, the Chromosome class contains one iteration at a time of the travel sequences of the houses traveled, as well as crossover algorithm more on that later, mutation, and has a function that calculates its fitness based on the sequence of genes the chromosome has. The population class contains all the chromosome sequences and is responsible for mutating best-fit chromosomes based on elitism.

Chromosome Mating (Elitism)

This function determines which chromosomes mutate to make two children, and the function is based on elitism where the top 10% of the fittest chromosomes get chosen to be in the next generation. The rest of the population is picked from the top 50%, where we chose two parents at random from this range and mate them to produce two offspring.

Crossover (Ordercrossover)

The type of crossover I used was the order crossover, which essentially gets a part of each parent and gives to children, and the remaining empty genes get assigned a house based on what genes are empty and the other parent's genes as well.

Mutation (Swap)

For mutation, I just swapped two genes based on the probability given in this case it was hard coded to be 0.5 to produce random results.

Fitness (Distance from the warehouse)

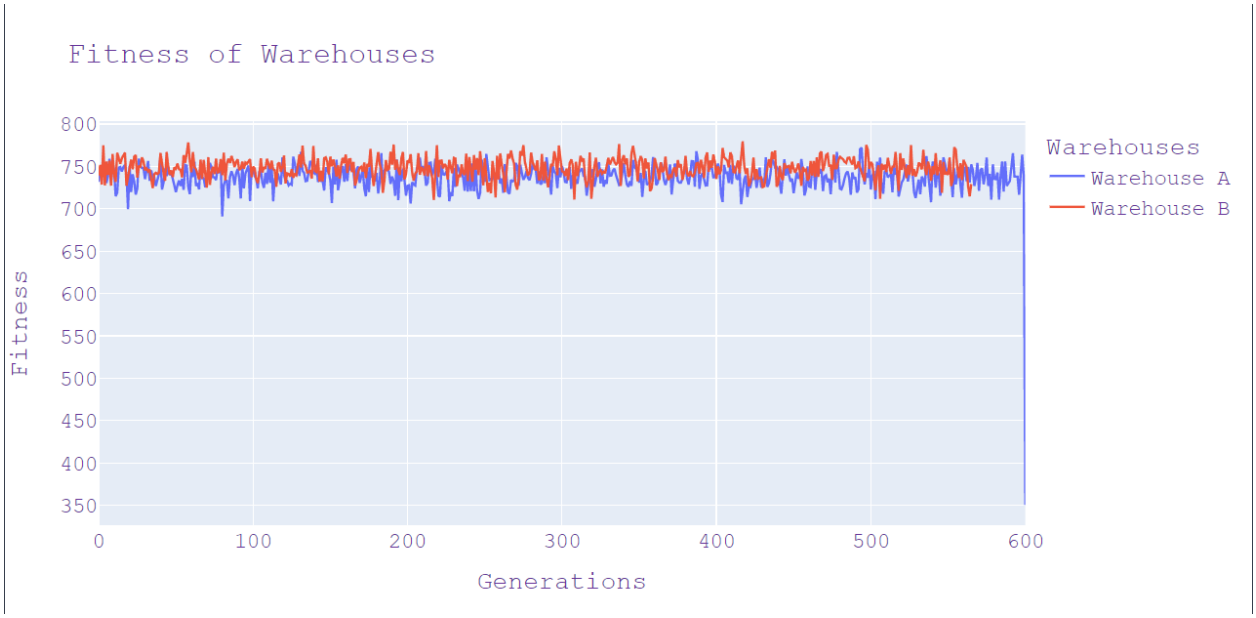
For fitness, it calculates the distance between two vectors relative to a gene (house), in this case, the two vectors are warehouses, and adds all the distance traveled between all genes, hence why fitness graphs won't go down.

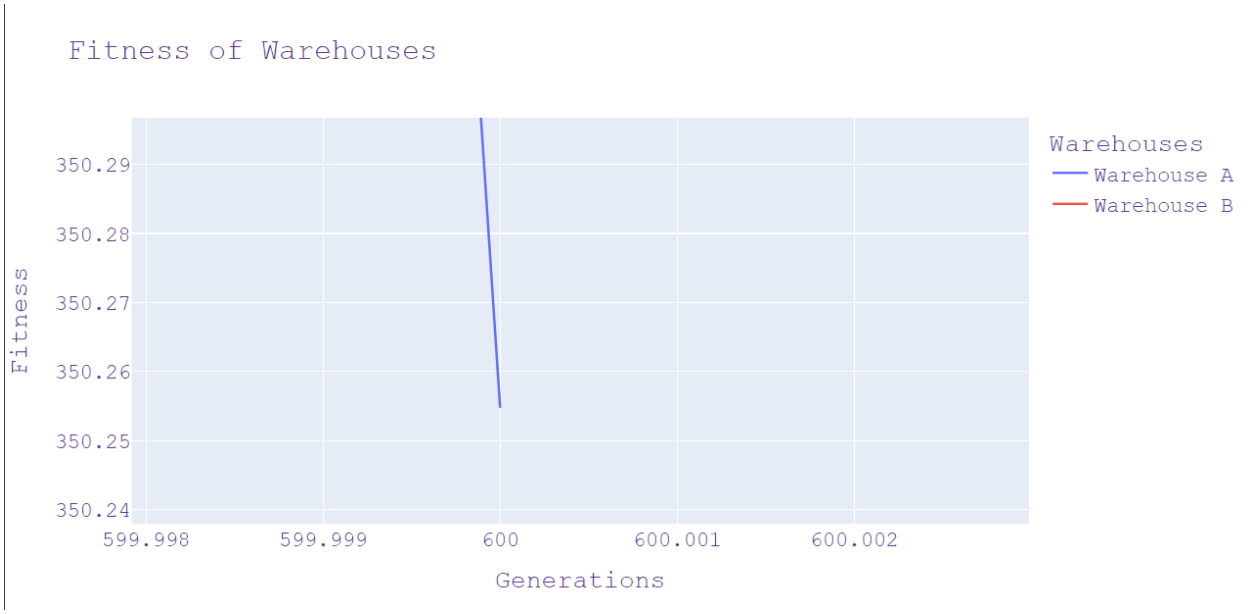
Graphs

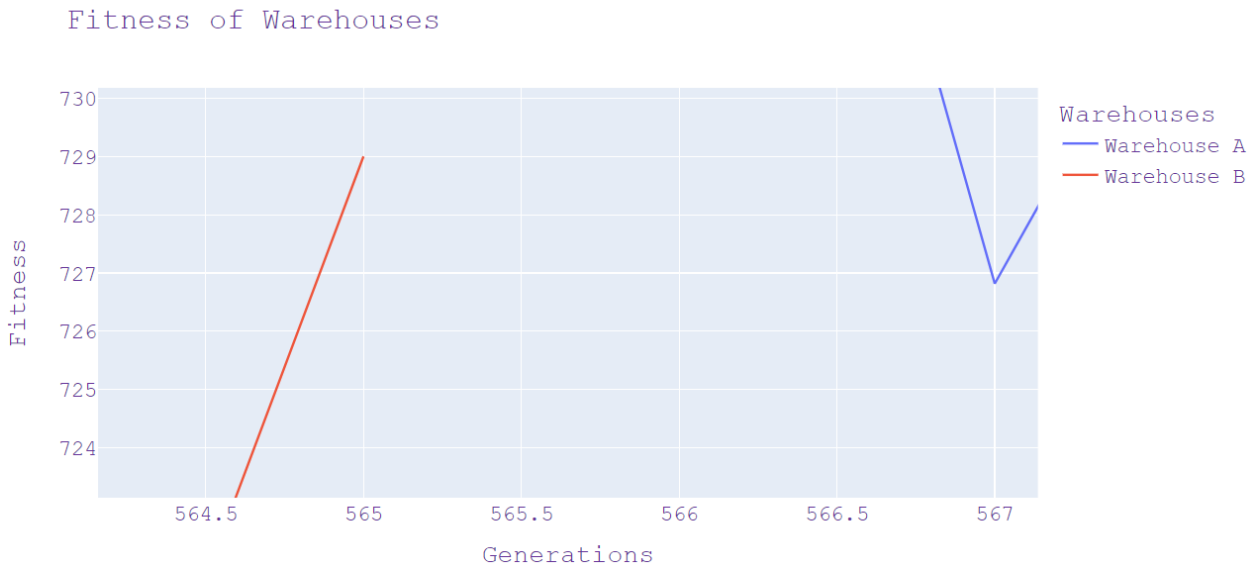
For the graphs I made them in Jupiter notebooks, it uses the output generated by each run in the unity program, and it creates a txt file each time with the fitness for each warehouse at every generation. As you might tell the values don't change too much and that is because the fitness calculated traverses through all the genes and thus doesn't ignore any at one time, thus we get a fitness curve that doesn't go down until the last values. Lastly, all tests were running generation number 600, probability crossover 0.7, probability mutation 0.5, and 20 chromosomes.

Fitness of warehouses

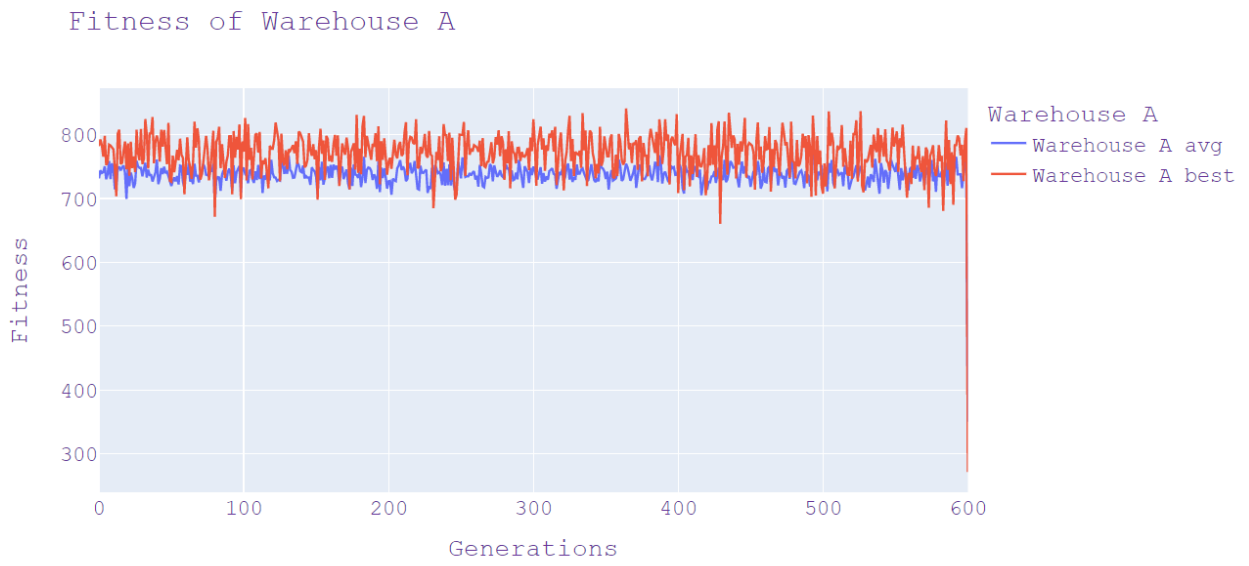
Notes: warehouse B cuts off before reaching the last number of generations, in this case, 600, reason was different dimensions of fitness for each run, and couldn't get the values to appear.

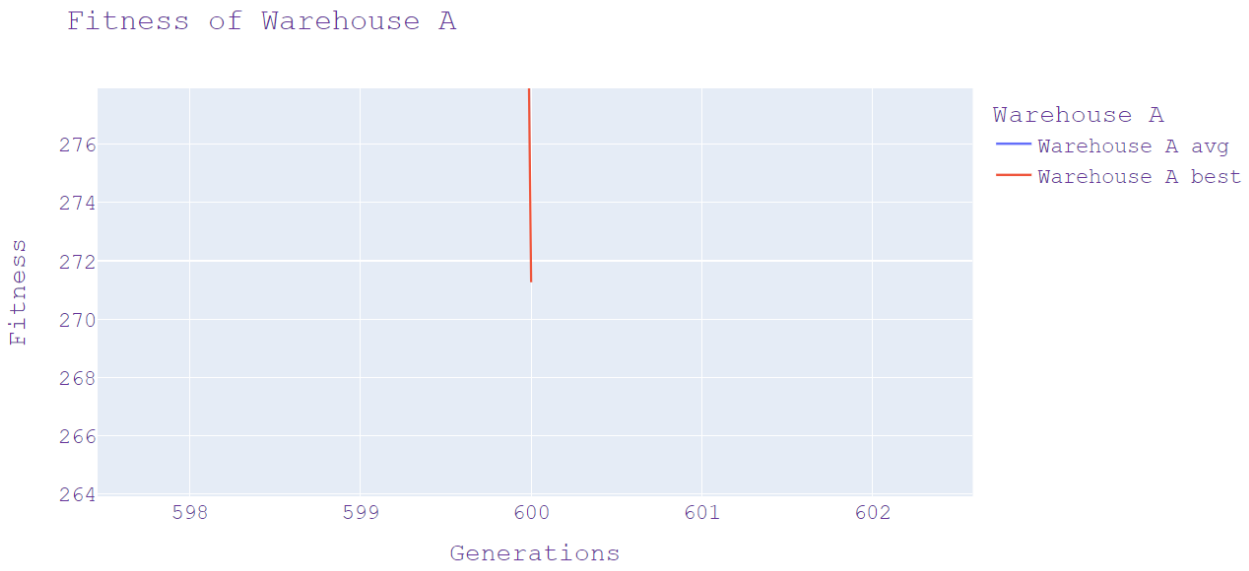



Fitness of Warehouses
Fitness
350.29
350.28
350.27
350.26
350.25
350.24
599.998
599.999
600
600.001
600.002
Generations
Warehouses
Warehouse A
Warehouse B


Fitness of Warehouses
Fitness
730
729
728
727
726
725
724
564.5
565
565.5
566
566.5
567
Generations
Warehouses
Warehouse A
Warehouse B

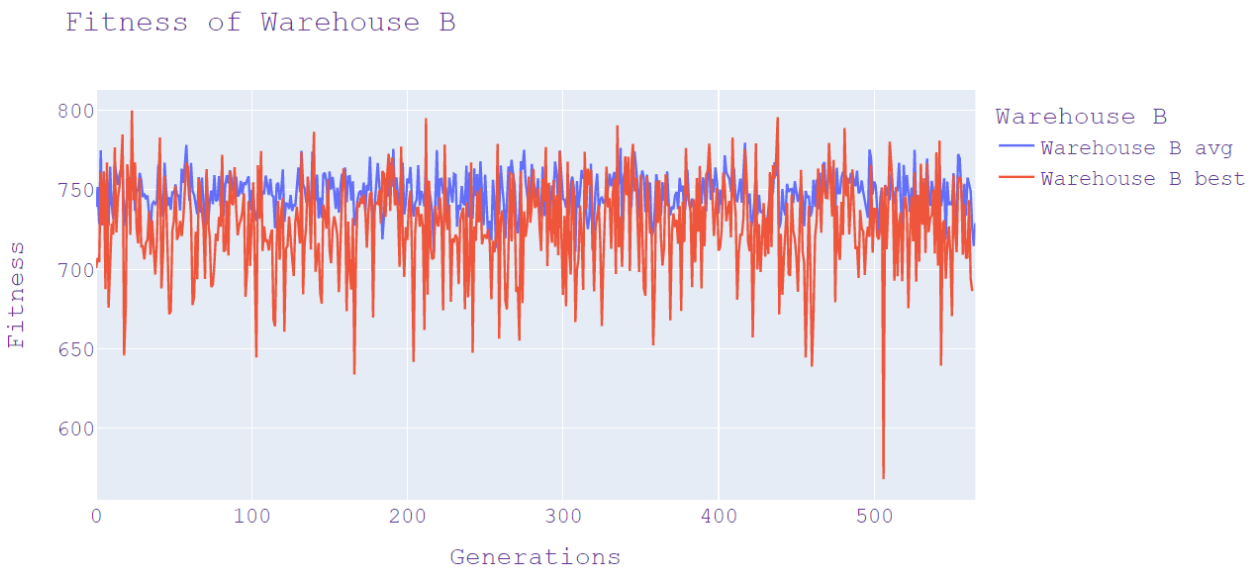
Fitness of warehouse A

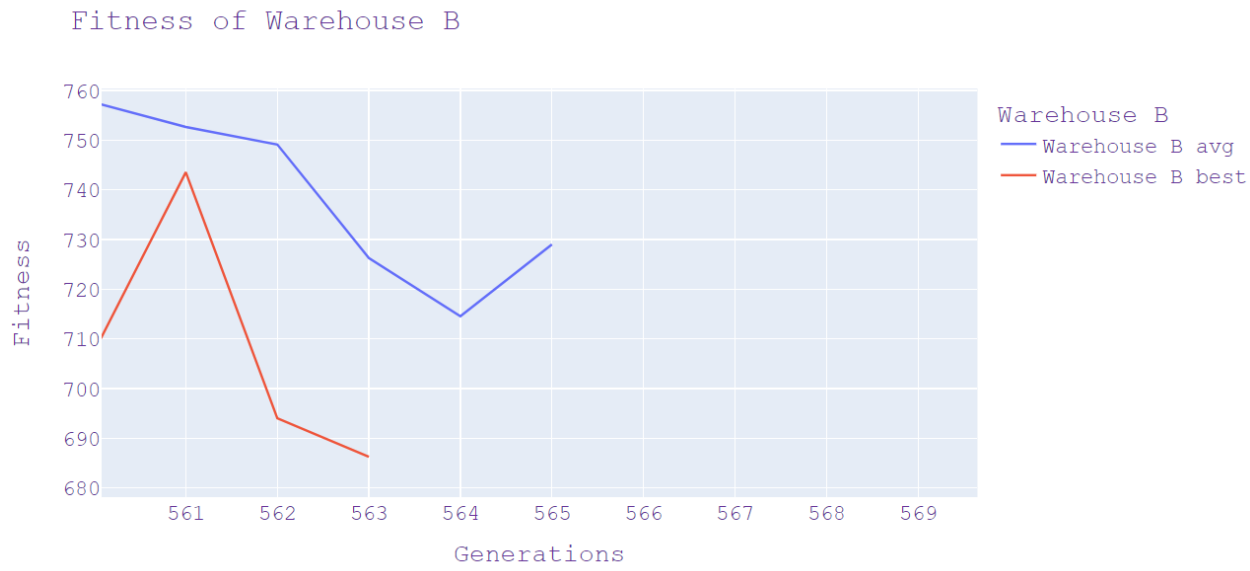




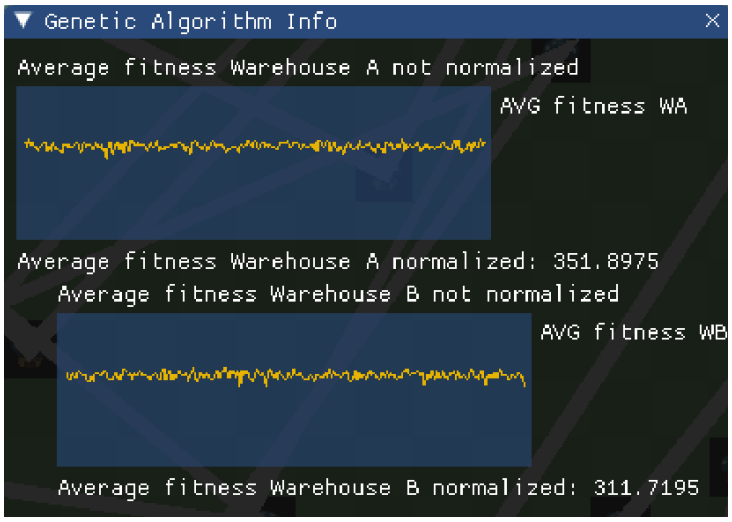
Fitness of warehouse B

Notes: warehouse B cuts off before reaching the last number of generations, in this case, 600, reason was different dimensions of fitness for each run, and couldn't get the values to appear.





Fitness of warehouses in application



Advantages / Disadvantages

I. Advantages

Produces correct solutions not necessarily optimal as well as gives feedback on the current run.

II. Disadvantages

The solutions produced are clearly not optimal ones, as well as slow if the generations range is in the thousands, no multi-threading is used in this program.

III. Comparison to exhaustive search

It produces an optimal solution in a linear amount of time compare to an exhaustive which would run at O(n^2) and even more depending on the implementation.

Conclusion

Overall my solution was clearly not the best, my reason for that was the crossover implementation did not seem to create the best children and it seemed to get stuck in local maxima, for this I set a higher mutation rate which did help, but none optimal solutions were still being produced. In terms of expectations, this program does have a UI as well as graphs to show performance which, makes this program easier to test, if time had allowed I would have implemented another type of crossover to arrive at a better solution on every run.