Thomas Pollard
CS – 441 Artificial intelligence, Winter 2021
Programming 2 Report

AI – Programming Assignment 2

Overview

Programming2.py is my implementation of a genetic algorithm that attempts to solve the eight queens problem. There are three hyper-parameters, POPULATION_SIZE, NUM_ITERATIONS, and MUTATE_CHANCE. The fitness function I used counts the number of non-mutually attacking pairs of queens, which is calculated by taking 28 (the maximum fitness) and subtracting the number of attacking pairs. The program starts by creating POPULATION_SIZE random starting individuals. Then, it runs through NUM_ITERATIONS generations. Each new generation is produced by running the selectiveBreeding() function on the previous population. selectiveBreeding() starts by finding the sum of the populations fitness, and then constructing a normalized fitness for each individual. These are placed in a list with and index matching their respective individual in the population list. These normalized fitness's are then used as the probability weights for a random choice for mating selection. Once two unique parents are selected, makeBabies() takes the two parents and returns a list with two offspring. makeBabies() crosses the genes of the two parents at a random crossover point, and each has an independent chance to mutate which changes one gene to a random value.

I compared the results of changing the population size, as well as the mutation chance. I kept the number of iterations constant at 500 for each experiment so that I could chart the results over the long term and compare. I included the raw output of each experiment in a separate excel spreadsheet due to the size and am only going to reflect on the charts in this report.

Thomas Pollard
CS – 441 Artificial intelligence, Winter 2021
Programming 2 Report

MUTATION_CHANCE = 0.01

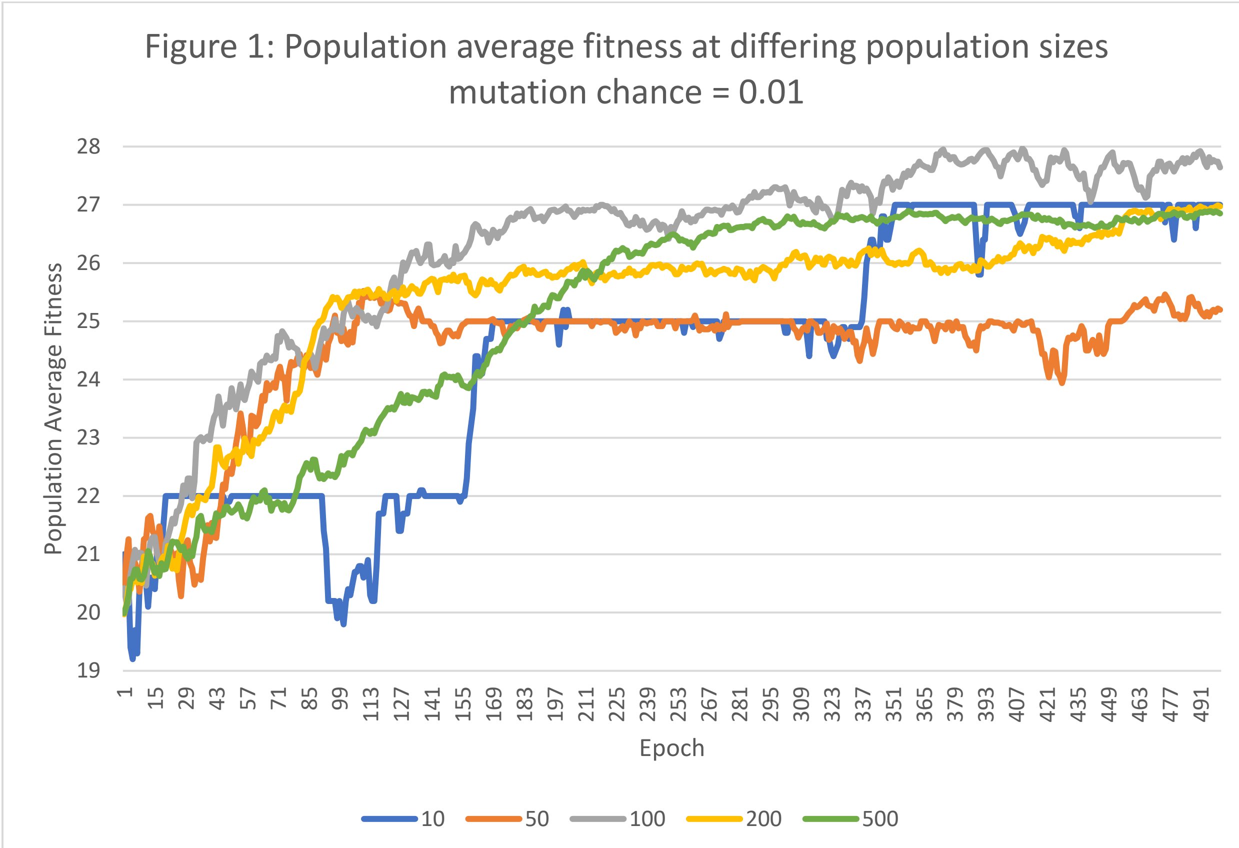

Figure 1 shows the average fitness of the population at different population sized and mutation chance of 0.01 for 500 epochs. A population of 100 had the best performance.

Thomas Pollard
CS – 441 Artificial intelligence, Winter 2021
Programming 2 Report

MUTATION_CHANCE = 0.05

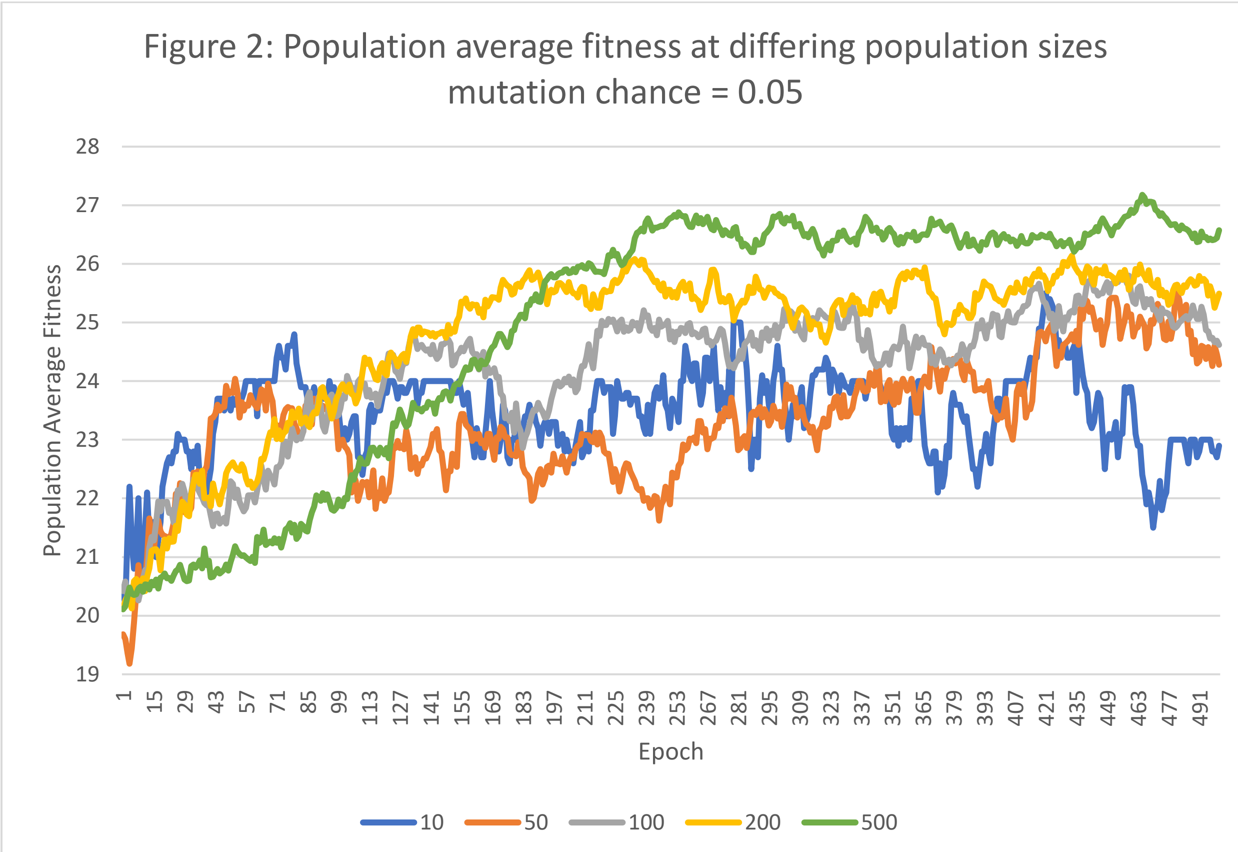

Figure 2 shows the average fitness of the population at different population sized and mutation chance of 0.05 for 500 epochs. A population of 500 had the best performance.

Thomas Pollard
CS – 441 Artificial intelligence, Winter 2021
Programming 2 Report

MUTATION_CHANCE = 0.1

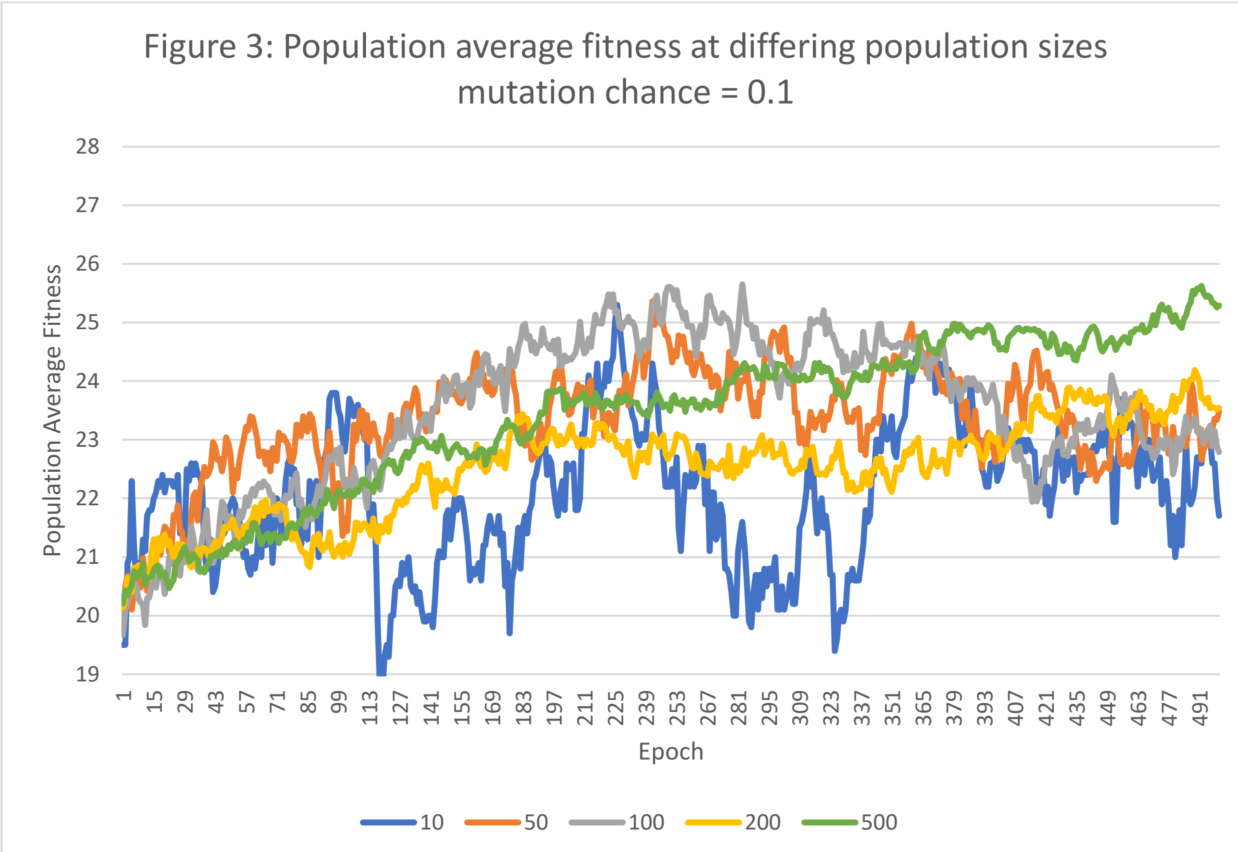

Figure 3 shows the average fitness of the population at different population sized and mutation chance of 0.1 for 500 epochs. A population of 500 had the best performance.

Conclusion

It seems as though the larger the population is, the better the average fitness of the population. It also seems, at least from these experiments, that a mutation chance larger than 10% has a large negative impact on average fitness of the population. I think, however, that it just makes the algorithm take longer to converge and that it could still produce answers if allowed to run long enough. Also, when running these algorithms, I sometimes found that a run that reached high population average fitness would not find any solutions over the 500 generations, while some would produce solutions when the average fitness was 20-24. This might indicate that the average fitness might not be the optimal solution for a proxy of effectiveness. An alternative might be the average number of generations to find a solution over a large number of runs.