CS 541: Artificial Intelligence, Winter 2022

Programming Assignment #2

The goal of this assignment is to implement a genetic algorithm to solve the 8-queens problem.

Individual Population: Each individual in a population is an object of the class. The individual function that I defined here has three characteristics: the _init_function of the class, when a new object is generated, it's fitness is calculated. Therefore, each individual contains the board configuration and corresponding fitness.

1. *indiv :* it's an array equivalent to the chess board with a particular configuration of the 8 queens on the board. For example, [5,8,7,4,3,2,1,6]. A successful configuration would be [6, 8, 2, 4, 1, 7, 5, 3]
2. *Fitness :* An integer with maximum value 28. The fitness of an individual is the number of pairs of non attacking queens in the indiv. Therefore, for [5, 8, 7, 4, 3, 2, 1, 6] it is 19, and for the successful configuration [6, 8, 2, 4, 1, 7, 5, 3] it is 28 (max-possible)
3. *Si* : Probability of selection. This is computed as individual's fitness / sum of fitness of individuals in the population

Initial Population: The Initial population is generated by creating new objects of the class Individual with a random ordering of numbers between 1 and 8. Each new object thus created is appended to the array population[]. Then the si value of each individual of the population is calculated and noted in each object individually in the array population.

Selection of Parents: The two parents are selected in each iteration by calling the roulette() function. This function implements the "fitness proportionate selection" (sometimes called "roulette wheel selection") as explained in class.

Crossover: The crossover point for each pair of parents is generated randomly between integers 0 and 7. At the crossover point, the 2 children are generated by combining the parent arrays, as explained in the text/slides.

***Mutation*:** Each child is sent as an argument in a call to the mutation() function. This function generates a random number, and if the number is below globally set parameter MutationPct, then the mutation occurs. At a random index of the child array, the integer is replaced with a randomly generated integer between 1 and 8.

Report:

1. The program sometimes approached a local minima of 25 and 26, in the next few iterations of configuration would be generated so that the newer population's individual fitness will be again in the range of 19-20 and so on. Very low

mutation probability led to programs finishing the while loop without finding a solution at all.Therefore, randomly mutating the child individuals avoided local maximas, and reset the path.

2. The program performs sporadically, with a constant crossover point of 4 (midpoint) and a randomly generated crossover point and either of the methods of determining the crossover point could yield similar or vastly different results.Therefore, no specific option for generating the crossover point can be said to be better than the other.
3. With a population size of time, very rarely was a goal state reached. Even when the iteration number was NumIterations = 1000000 and PopulationSize = 10, the goal state could not be reached. But, with the PopulationSize = 500, the goal state has been reached (as observed) even with just 12 iterations. Therefore, higher population sizes tend to do better.

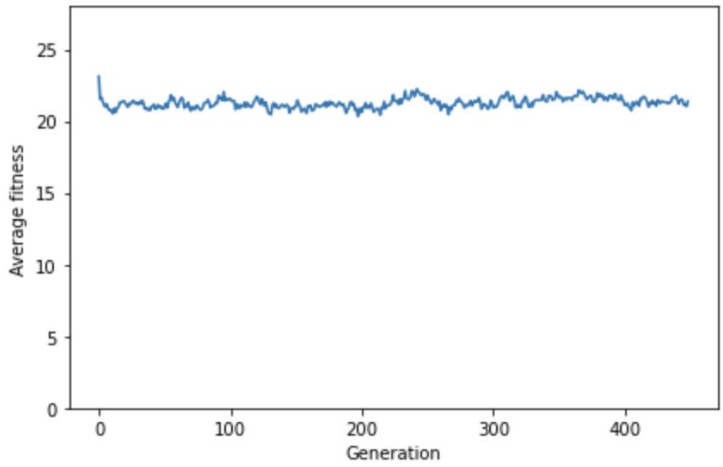
Results:

Example1:

PopulationSize = 100

NumIterations = 500

Resolved: [5, 1, 4, 6, 8, 2, 7, 3]

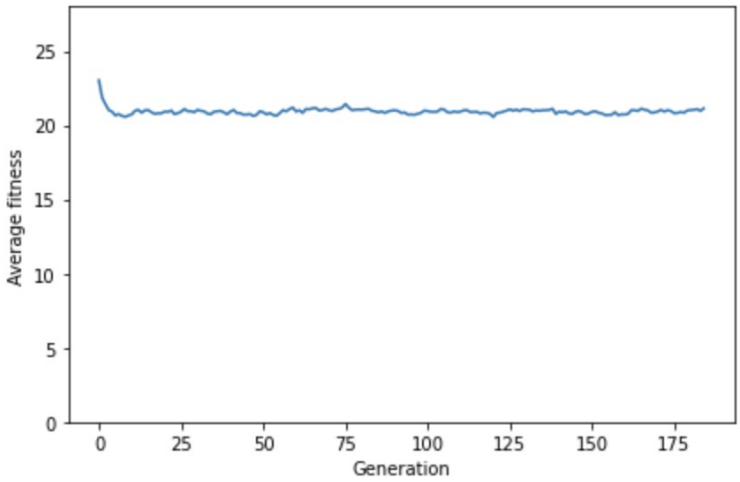


Example2:

PopulationSize = 500

NumIterations = 500

Resolved: [3, 6, 8, 1, 4, 7, 5, 2]

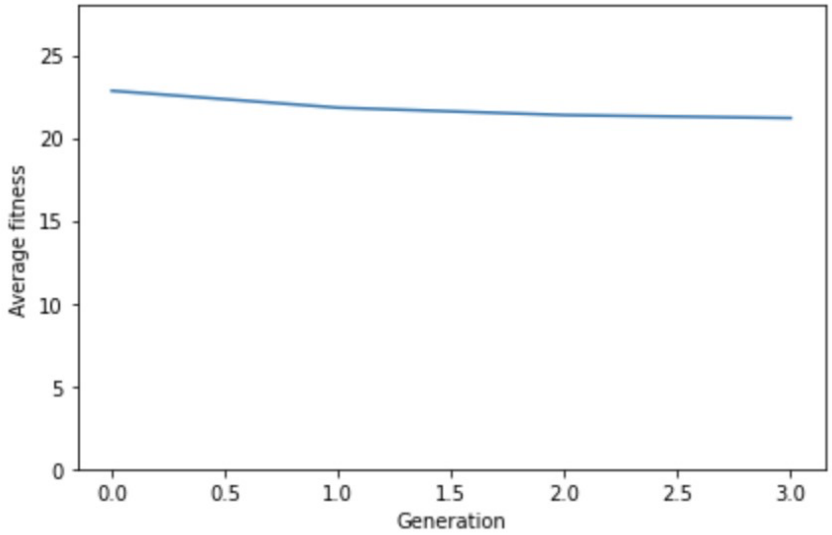


Example3:

PopulationSize = 1000

NumIterations = 500

Resolved: [3, 7, 2, 8, 6, 4, 1, 5]



Example4:

PopulationSize = 10

NumIterations = 100

The goal state is not reached before the end of iterations.